

Programmieren kann sie!

Jörg J. Buchholz

14. Juni 2024

# Kapitel 1

## Einführung

In den vergangenen Monaten hatten wir untersucht, wie gut eine Künstliche Intelligenz<sup>1</sup> (KI) Ingenieurmathematik-, Regelungstechnik- und Flugregelungsaufgaben lösen, Matlab-Programme schreiben und Fragen zu mathematischen und regelungstechnischen Problemen beantworten kann [1], [2], [3].

Als KI verwendeten wir GPT-4 [4] von OpenAI und als Aufgabensammlung (MATH1, MATH2, RTFR) die in Abschnitt 1.2 beschriebene CAT-Übungsumgebung [5] der Abteilung Maschinenbau der Hochschule Bremen.

In diesem Artikel nutzen wir wiederum GPT-4, um die Aufgaben der Informatik (INFO) des zweiten Semesters von der KI analysieren zu lassen.

Mittlerweile kann GPT-4

- Mathematische und in Grenzen auch (flug-)regelungstechnische Fragen beantworten
- Matlab-Programme erstellen
- Selbst rechnen, in dem sie mal schnell ein Python-Programm erstellt und eigenständig intern ausführt
- Hochgeladene Dokumente lesen und deren Inhalt bei der Beantwortung von Fragen berücksichtigen [3]
- Eigene GPTs erstellen, die wir mit fachspezifischen Informationen füttern können [6], [3]
- Mit DALL · E 3 Bilder erstellen [7], [3]

Während die ingenieurmathematischen Diskussionen mit GPT-4 mittlerweile teilweise richtig Spaß machen [1], [2], fallen die Antworten auf (flug-)regelungstechnischer Fragen ziemlich durchwachsen aus [3]. Das Thema ist nun mal recht speziell, sodass das Weltwissen der KI vermutlich noch nicht ausreichend fachspezifische Quellen beinhaltet.

In Informatik fühlt sie sich hingegen erwartungsgemäß regelrecht zu Hause; ihre klugen Antworten auf Hard- und Softwarefragen offenbaren, dass sie die umfangreich vorliegende Fachliteratur hinreichend verinnerlicht hat und beispielsweise Programme sowohl korrekt interpretieren und hervorragend erklären, als auch erfolgreich selbst schreiben kann.

---

<sup>1</sup>Wir werden die KI im Folgenden mit dem Personalpronomen „sie“ referenzieren.

Wie schon in Regelungstechnik und Flugregelung [3] haben wir unser Informatik-Skript (diesmal allerdings in Form des vollständigen PowerPoint-Dokuments) in die KI hochgeladen. Außerdem haben wir - anders als in RTFR - eine eigene Informatik-KI erstellt [6], der wir die folgende Anweisung mitgegeben haben:

*Erstelle alle geforderten Programme in Matlab.*

Des Weiteren hatten wir die KI mit individuellen Hinweisen („Was sollte ChatGPT über Sie wissen, um besser zu reagieren?“)

*Ich bin Professor für Ingenieurmathematik, Informatik, Regelungstechnik und Flugregelung an der Hochschule Bremen. Ich arbeite mit Matlab, Python, JavaScript und Visual Basic in ASP.NET.*

und gewünschten Reaktionen („Wie soll ChatGPT reagieren?“) geimpft

*Gib nur gesicherte Fakten wieder. Sprich mich mit "du" an.*

was mittlerweile bis auf sehr seltene Ausrutscher auch anstandslos funktioniert. Allerdings verwendet sie das „du“ nur dann, wenn wir sie auch in der (Nach-)Frage „duzen“; da die CAT-Fragen an Studierende gerichtet sind, nutzen wir dort die förmliche Anrede und die KI antwortet daher üblicherweise auch mit einem „Sie“.

Im Rahmen dieses Artikels analysieren wir in Kapitel 2 einige Matlab-Programme und Antworten der KI. Im Anhang A finden sich alle Aufgaben, Programme und Antworten einschließlich einiger emotionaler Kommentare des Autors.

## 1.1 Nachteile eines eigenständigen INFO-Assistenten

Das Verwenden einer eigenständigen Informatik-KI hat im aktuellen OpenAI-Umfeld zwei große Nachteile: Obwohl wir ihr es allgemein und im Einzelfall ausdrücklich erlaubt haben, weigert sich die spezielle Informatik-KI hartnäckig, Python-Code **auszuführen**. Die Programme werden zwar erzeugt; wenn wir sie laufen lassen wollen, müssen wir aber immer wieder zum Standard-GPT-4 (oder natürlich in eine eigene Python-Umgebung) wechseln.

Außerdem können wir dem INFO-Assistenten „im Gespräch“ beispielsweise keine externe SVG-Datei hochladen. Wir müssen ihm stattdessen den Inhalt der Datei als ASCII-Text mit in die Frage packen, was auf die Dauer ziemlich lästig wird.

## 1.2 CAT (Computer Aided Teaching)

Seit vielen Jahren können in der Abteilung Maschinenbau der Hochschule Bremen die Module Mathematik 1, Mathematik 2, Informatik, Physik, Regelungstechnik und Flugregler und Advanced Computer Based Mathematics im Rahmen der CAT-Übungsumgebung [5] durchgeführt werden. Dabei lösen die Studierenden in Zweiergruppen kontinuierlich in jeder Woche des Semesters Aufgaben, die dann am Ende des Semesters als Prüfungsleistung benotet werden. Die Aufgaben beinhalten das Schreiben von Matlab- und Simulink-Programmen und das Beantworten von Fragen zu diesen Programmen und den modulspezifischen Zusammenhängen und Hintergründen.

Einer der Beweggründe für diesen Artikel war die Frage, ob unsere Studierenden die KI zur Lösung der Aufgaben und Beantwortung der Fragen nutzen können. Während es bei den ingenieurmathematischen Aufgaben von MATH1 und MATH2 noch durchaus hilfreich

war, sich bei der Beantwortung der Fragen und beim Schreiben der Matlab-Programme von der KI unterstützen zu lassen, gibt sie bei den (flug)regelungstechnischen Aufgaben und Fragen in RTFR doch bisweilen ziemlichen Müll zurück, der sich natürlich ohne das entsprechende Hintergrundwissen nicht unmittelbar als solcher erkennen lässt; sich plausibel klingend ausdrücken kann sie ja.

Die Antworten auf die Informatikfragen hingegen sind größtenteils geradezu lehrbuchhaft und absolut vorbildlich - und damit natürlich leicht von den üblichen Antworten der Studierenden zu unterscheiden. Insbesondere das Erklären von vorgegebenem Code beherrscht die KI meisterhaft; die schrittweisen, detaillierten Erläuterungen sind wirklich hilfreich und spiegeln teilweise verblüffend präzise die individuellen Beweggründe für die jeweilige Art des Programmierens. Manchmal lobt die KI unseren Code sogar als besonders effizient, klar und strukturiert. ;-)

Dass sie selbst fast immer lauffähigen (Matlab-)Code erzeugen kann, empfinden wir mittlerweile ja schon fast als selbstverständlich; wir gewöhnen uns ja tatsächlich sehr schnell an diese eigentlich noch vor wenigen Jahren absolut unglaublichen Fähigkeiten einer künstlichen Intelligenz.

# Kapitel 2

## Im Einzelnen

Wir werden in diesem Kapitel einzelne Aufgaben und Antworten herausgreifen und analysieren, um ein Gefühl dafür zu bekommen, wo wir die KI jetzt schon sinnvoll einsetzen können und es wo noch deutliches Optimierungspotenzial gibt.

### 2.1 Wo Licht ist, ...

Was GPT-4 mittlerweile perfekt beherrscht, sind ihre (ingenieur-)mathematisch präzisen Erläuterungen und insbesondere ihre Zusammenfassungen.

#### 2.1.1 HTML, MathJax, LaTeX, ...

Hier geht es um grundlegende Fragen zu HTML, MathJax und LaTeX:

Ich

*Testen Sie das HTML-LATEX-Beispiel aus dem Skript im Online-HTML-Editor des Coding Ground. Denken Sie auch hier daran, die gesamte MathJax-Adresse in eine Zeile zu schreiben.*

- *Wozu wird MathJax aufgerufen?*
- *Was bewirkt  $\$$  normalerweise in einem LaTeX-Programm?*
- *Warum ist sein Einsatz hier nicht notwendig?*
- *Warum verwenden Sie eine align-Umgebung?*
- *Was bewirkt  $\backslash$ ?*
- *Warum ist  $1=2$ ?*

Die Antworten der KI sind ausgewogen detailliert und offenbaren, dass sie tatsächlich das Skript gelesen hat:

**GPT-4**

- *MathJax wird verwendet, um mathematische Ausdrücke innerhalb einer Webseite darzustellen. Es ist besonders hilfreich für die Darstellung von komplexen mathematischen Formeln in HTML, ohne auf Bilder oder spezielle Schriftarten zurückgreifen zu müssen. MathJax interpretiert mathematische Markup-Sprachen wie LaTeX, MathML und AsciiMath und stellt sie als hochwertige typografische Ausgaben direkt im Browser dar.*

- In einem LaTeX-Programm wird  $\$$  verwendet, um mathematische Ausdrücke in einem abgesetzten (*display*) Modus zu setzen, was bedeutet, dass die Ausdrücke zentriert und auf einer eigenen Zeile dargestellt werden. Im Gegensatz zum *Inline-Modus*, wo mathematische Inhalte innerhalb eines Textabsatzes erscheinen, bietet der *Display-Modus* mehr Platz und macht komplexe Formeln übersichtlicher.
- Im HTML-Beispiel, das Sie aus dem Skript testen sollten, ist die Verwendung von  $\$$  nicht notwendig, da MathJax und die umgebende *align-Umgebung* bereits dafür sorgen, dass die mathematischen Ausdrücke im *Display-Modus* und entsprechend formatiert angezeigt werden.
- Die *align-Umgebung* wird in LaTeX und hier in Verbindung mit MathJax verwendet, um mathematische Gleichungen untereinander auszurichten, typischerweise am Gleichheitszeichen oder einem anderen Operator. Dies verbessert die Lesbarkeit von aufeinanderfolgenden Gleichungen oder mathematischen Ausdrücken.
- In mathematischen Formeln wird der doppelte Rückstrich  $\backslash\backslash$  verwendet, um einen Zeilenumbruch einzufügen. In der *align-Umgebung* ermöglicht er die Trennung verschiedener Gleichungen oder Ausdrücke, die untereinander ausgerichtet werden sollen.
- Die Aussage  $1=2$  ist natürlich mathematisch falsch. Sie wird in didaktischen Beispielen oft als absichtlich falsches Ergebnis dargestellt, um Fehler in der Argumentation oder die Auswirkungen bestimmter mathematischer Manipulationen (wie Division durch Null) zu illustrieren. In Ihrem Skript dient dies möglicherweise als Beispiel oder als Teil einer mathematischen Paradoxie oder Fehlschluss, um zu diskutieren, warum solche "Beweise" nicht gültig sind.

Auf Nachfrage

**Ich** Analysiere das konkrete  $1=2$ -Beispiel aus dem Skript.

erläutert sie dann ausführlich, wo der Null-Divisionsfehler im LaTeX-Code<sup>1</sup> im Skript liegt:

#### GPT-4

Das Beispiel aus dem Skript führt Schritt für Schritt zu der inkorrekten Schlussfolgerung, dass 1 gleich 2 sei. Lassen Sie uns die Gleichungen analysieren:

$$a^2 - a^2 = a^2 - a^2$$

Dies ist offensichtlich wahr, da beide Seiten der Gleichung identisch sind.

$$a \cdot (a - a) = (a + a) \cdot (a - a)$$

Hier wurde die linke Seite als

$$a^2 - a^2$$

und die rechte Seite als

---

<sup>1</sup>Es ist sehr angenehm, dass die KI sogar die in LaTeX gesetzten Formeln lesen und korrekt interpretieren kann.

$$2a^2 - 2a^2$$

umgeschrieben, was immer noch korrekt ist, da beide Seiten nach der Ausmultiplizierung zu Null werden.

$$a = a + a$$

An diesem Punkt wird ein Fehlschluss begangen: Hier scheint durch

$$a - a$$

(also Null) geteilt worden zu sein, um von der vorherigen Zeile zu dieser zu gelangen. Da die Division durch Null nicht definiert ist, ist dieser Schritt ungültig.

Die Erläuterungen gehen noch etwas weiter, bringen aber keine zusätzlichen Erkenntnisse. Außerdem sind ihre zwischenzeitlichen Aussagen über die beiden Seiten der Gleichungen nicht wirklich korrekt. Die Schlussfolgerung stimmt dann wieder.

## 2.1.2 Zahlendarstellung

Bei der Umwandlung von Binärzahlen in Dezimalzahlen soll der Satzteil „im Folgenden“ eigentlich andeuten, dass das Programm erst im Rahmen der folgenden Fragen vervollständigt wird. Die „übereifrige KI“ beantwortet aber nicht nur die gestellte Frage ausführlich, sondern schreibt auch gleich das komplette Programm – mit einer interessanten „Idee“, um ohne den `str2double`-Befehl auszukommen:

### Ich

Sie möchten im Folgenden ein Unterprogramm `b2d` erstellen, das eine Binärzahl `b` in eine Dezimalzahl `d` umrechnet. Die Binärzahl `b` wird dabei als Zeichenkette übergeben. Bestimmen Sie im Unterprogramm als erstes die Länge von `b` und initialisieren Sie die Dezimalzahl `d` mit `null`.

Warum sollten Sie die Binärzahl als Zeichenkette übergeben?

### GPT-4

Die Übergabe der Binärzahl als Zeichenkette (String) in einem Unterprogramm zur Umwandlung in eine Dezimalzahl hat mehrere Gründe:

1. **Flexibilität:** Zeichenketten können Binärzahlen beliebiger Länge darstellen, was bei der Verwendung von numerischen Datentypen (wie `Integer`) nicht immer der Fall ist, da diese typischerweise auf eine maximale Anzahl von Bits beschränkt sind.
2. **Vermeidung von Fehlinterpretationen:** Direkte Übergabe einer Binärzahl in einem numerischen Format könnte zu Verwirrungen führen, da numerische Werte standardmäßig als Dezimalzahlen interpretiert werden. Ein String macht deutlich, dass es sich um eine Folge von Binärzeichen handelt.
3. **Einfachere Manipulation:** Die Operationen auf den einzelnen Zeichen einer Zeichenkette (zum Beispiel das Auslesen der `n`-ten Position) sind oft einfacher zu handhaben als bitweise Operationen auf numerischen Typen.

Basierend auf diesen Gründen können wir ein Matlab-Unterprogramm `b2d` schreiben:

```
function d = b2d(b)
    % Initialisierung der Dezimalzahl
    d = 0;
    % Bestimmung der Länge der Binärzahl
    n = length(b);
    % Umrechnung der Binär- in eine Dezimalzahl
    for i = 1:n
        % Umrechnung des i-ten Zeichens in eine Zahl und
        % Multiplikation mit der entsprechenden 2er Potenz
        d = d + (b(i) - '0') * 2^(n-i);
    end
end
```

In diesem Programm ist `b` die Eingabezeichenkette, die die Binärzahl repräsentiert, und `d` ist die Ausgabe, die dezimale Darstellung der Binärzahl. Wir subtrahieren '0' von jedem Zeichen, um den entsprechenden numerischen Wert zu erhalten (da in ASCII die Ziffern 0-9 aufeinanderfolgende Codes haben). Die Potenzierung mit  $2^{(n-i)}$  erfolgt, weil das linke Zeichen die höchste Wertigkeit (größte Potenz von 2) hat.

Bei der Umwandlung einer Binärzahl in eine Hexadezimalzahl

### Ich

Sie möchten im Folgenden ein Unterprogramm `b2h` schreiben, das eine Binärzahl hexadezimal darstellt. Im Unterprogramm bestimmen Sie als Erstes wieder die Anzahl der Zeichen der Binärzahl (hier `n_b`). Im Folgenden wollen Sie dann jeweils vier Stellen der Binärzahl zu einem hexadezimalen Zeichen zusammenfassen. Es ist daher notwendig, die Länge der Binärzahl auf ein Vielfaches von vier aufzufüllen, indem Sie die Binärzahl links mit der entsprechenden Anzahl an Nullen aufpolstern. Um die Anzahl der notwendigen Nullen herauszufinden, verwenden Sie die Zuweisung

$$n\_padding = 4 * \text{ceil}(n\_b / 4) - n\_b$$

Erläutern Sie ausführlich die Funktionsweise der Zuweisung.

Was bewirkt der `ceil`-Befehl?

Warum können Sie stattdessen nicht den `round`-Befehl verwenden?

sind die schrittweise leicht nachvollziehbaren Erklärungen der KI einfach wunderschön zu lesen:

### GPT-4

Die Zuweisung `n_padding = 4 * ceil(n_b / 4) - n_b` wird verwendet, um die Anzahl der Nullen zu bestimmen, die einer Binärzahl vorangestellt werden müssen, damit ihre Gesamtlänge ein Vielfaches von vier beträgt. Dies ist notwendig, weil jede Hexadezimalziffer vier Binärziffern entspricht. Erläutern wir die einzelnen Teile dieser Zuweisung:

1. `n_b / 4`: Dieser Ausdruck berechnet, wie viele Vierergruppen in der Binärzahl vorhanden sind, was einer Division der Gesamtlänge der Binärzahl durch vier entspricht. Wenn `n_b` bereits ein Vielfaches von vier ist, wäre dieser Wert eine ganze Zahl; wenn nicht, wäre es eine Fließkommazahl.

2. `ceil(n_b / 4)`: Der `ceil`-Befehl (Ceiling-Funktion) rundet den Wert innerhalb der Klammern auf die nächste ganze Zahl auf, falls es sich nicht bereits um eine ganze Zahl handelt. Das bedeutet, wenn `n_b / 4` nicht schon eine ganze Zahl ist (d.h., die Länge der Binärzahl ist kein Vielfaches von vier), wird sie auf die nächste ganze Zahl aufgerundet. Dies gibt uns die Anzahl der vollständigen Vierergruppen, die benötigt würden, um die Binärzahl vollständig darzustellen, einschließlich der notwendigen zusätzlichen Gruppen für fehlende Binärstellen.
3. `4 * ceil(n_b / 4)`: Dies multipliziert die Anzahl der notwendigen Vierergruppen (bestimmt im vorigen Schritt) mit vier, um die neue Gesamtlänge der Binärzahl zu erhalten, die nun ein Vielfaches von vier ist.
4. `4 * ceil(n_b / 4) - n_b`: Schließlich wird die ursprüngliche Länge der Binärzahl von der neuen Länge abgezogen. Das Ergebnis ist die Anzahl der Nullen, die der Binärzahl vorangestellt werden müssen, damit ihre Länge ein Vielfaches von vier beträgt.

Der `ceil`-Befehl wird verwendet, weil wir immer auf die nächste ganze Zahl aufrunden müssen, wenn die Länge der Binärzahl kein Vielfaches von vier ist. Dies stellt sicher, dass wir genügend Nullen hinzufügen, um vollständige Hexadezimalziffern zu bilden.

Der `round`-Befehl ist hier nicht geeignet, da `round` einen Wert zur nächsten ganzen Zahl rundet, basierend auf der typischen Rundungsregel (wenn der Bruchteil kleiner als 0.5 ist, wird abgerundet; ist er 0.5 oder größer, wird aufgerundet). Wenn `n_b / 4` zum Beispiel 2.25 wäre, würde `round(n_b / 4)` dies zu 2 runden, was bedeutet, dass wir nicht genügend Nullen hinzufügen würden, um die Binärzahl auf ein Vielfaches von vier zu erweitern. Deshalb verwenden wir `ceil`, um sicherzustellen, dass wir immer auf die nächste ganze Zahl aufsteigen, was erforderlich ist, um die Länge korrekt auf ein Vielfaches von vier zu erweitern.

### 2.1.3 Unicode

Beim Thema Unicode erläutert die KI kurz, knapp und richtig, wie Matlab ASCII-Codes interpretiert:

#### Ich

Verwenden Sie

```
disp(char([97, 9, 98]))
```

um zu analysieren, wie Matlab das Tabulatorzeichen interpretiert.

Wie wird der Tab ausgegeben?

#### GPT-4

In MATLAB wird der `disp`-Befehl verwendet, um Daten im Command Window auszugeben. Wenn du `disp(char([97, 9, 98]))` ausführst, wird ein Array von numerischen ASCII-Codes in Zeichen umgewandelt und dann angezeigt. Hier ist, was jedes Element in dem Array repräsentiert:

- 97 entspricht dem Buchstaben 'a'.
- 9 entspricht dem Tabulatorzeichen (auch bekannt als Tab).
- 98 entspricht dem Buchstaben 'b'.

Wenn MATLAB diesen Befehl ausführt, interpretiert es das Tabulatorzeichen (ASCII-Code 9) als horizontalen Tab. In der Ausgabe im Command Window wird dies in der Regel als ein gewisser Abstand (häufig entspricht dies einem Abstand von mehreren Leerzeichen) zwischen den Buchstaben 'a' und 'b' dargestellt. Die genaue Anzahl der Leerzeichen kann abhängig von der Einstellung deines Editors oder der Umgebung variieren, aber traditionell entspricht ein Tab etwa vier bis acht Leerzeichen.

## 2.1.4 Objektorientierte Programmierung

Natürlich kann die KI grundlegende Programmierparadigmen (auch in Matlab) perfekt erklären:

### Ich

*Sie wollen im Folgenden eine Polynom-Klasse aufbauen.*

*Achtung: Leider können Sie Klassen (noch) nicht in Live Scripts (.mlx), sondern nur in klassischen Matlab Scripts (.m) definieren.*

*Die Klasse soll nur das Attribut `koeffizienten` besitzen. Ihr Konstruktor soll untersuchen, ob ihm ein Koeffizientenvektor übergeben wurde (if nargin > 0) und den Vektor dann dem entsprechenden Attribut zuweisen. Was passiert, wenn Sie dem Konstruktor keinen Koeffizientenvektor übergeben?*

*Natürlich ist es sinnvoll, schon während des Aufbaus der Klasse die einzelnen Methoden zu testen. An dieser Stelle können Sie jetzt schon die ordnungsgemäße Funktion des Konstruktor überprüfen ...*

### GPT-4

*Um eine Polynom-Klasse in MATLAB zu definieren, erstellen Sie eine Datei mit dem Namen `Polynom.m`. In dieser Datei definieren Sie die Klasse, ihr Attribut und den Konstruktor. Hier ist ein Beispiel, wie der Code aussehen könnte:*

```
classdef Polynom
    properties
        koeffizienten = []; % Standardwert ist ein leerer Vektor
    end
    methods
        function obj = Polynom(koeff)
            if nargin > 0
                obj.koeffizienten = koeff;
            end
            % Wenn keine Argumente übergeben werden,
            % bleibt koeffizienten leer
        end
    end
end
```

*In diesem Fall:*

- Das Attribut `koeffizienten` wird definiert und standardmäßig auf einen leeren Vektor gesetzt.

- Der Konstruktor *Polynom* wird definiert, der prüft, ob ein Argument (ein Koeffizientenvektor) übergeben wurde. Wenn `nargin > 0` wahr ist, also wenn mindestens ein Argument übergeben wurde, wird das übergebene Argument dem Attribut `koeffizienten` zugewiesen.
- Wenn Sie dem Konstruktor keinen Koeffizientenvektor übergeben, bleibt das Attribut `koeffizienten` auf seinen Standardwert gesetzt, also ein leerer Vektor. Dies liegt daran, dass der Konstruktor in diesem Fall den Teil der *if*-Bedingung überspringt, da `nargin` gleich 0 wäre.

Zur Überprüfung der ordnungsgemäßen Funktion des Konstruktors können Sie verschiedene Instanzen der *Polynom*-Klasse in einem separaten Script oder in der MATLAB-Kommandozeile erstellen, zum Beispiel:

```
p1 = Polynom([1 2 3]); % Erstellt ein Polynom-Objekt
                        % mit Koeffizienten
p2 = Polynom(); % Erstellt ein Polynom-Objekt ohne Koeffizienten
```

Danach können Sie die Attribute der erstellten Objekte überprüfen, um sicherzustellen, dass der Konstruktor wie erwartet funktioniert.

Als wir sie zum Schreiben einer `auswerten`-Methode auffordern, liefert sie unaufgefordert auch gleich einen kleinen Unit-Test mit:

### Ich

Die Methode `auswerten` ist einfach. Sie bekommt das *Polynom* und die Stelle  $x$ , an der das *Polynom* ausgewertet werden soll, über die Parameterliste übergeben und liefert den Funktionswert an der Stelle zurück. Dazu verwendet sie den Befehl `polyval`.

Wie kommen Sie an den Koeffizientenvektor des übergebenen *Polynoms* heran?

### GPT-4

Um die Methode `auswerten` in deiner *Polynom*-Klasse in MATLAB zu implementieren, die den Wert des *Polynoms* an einer bestimmten Stelle  $x$  berechnet, kannst du den Befehl `polyval` verwenden. `polyval` benötigt den Koeffizientenvektor und den Punkt  $x$ , an dem das *Polynom* ausgewertet werden soll. In deiner Klasse kannst du auf den Koeffizientenvektor mit `obj.koeffizienten` zugreifen, da dieser als Eigenschaft der Klasse gespeichert ist.

Hier ist, wie du die `auswerten`-Methode innerhalb deiner *Polynom*-Klasse hinzufügen kannst:

```
methods
% Bereits existierende Methoden
function wert = auswerten(obj, x)
    wert = polyval(obj.koeffizienten, x);
end
end
```

In dieser Methode:

- `obj` ist eine Referenz auf das aktuelle *Polynom*-Objekt.
- `obj.koeffizienten` ist der Koeffizientenvektor des *Polynoms*.
- `x` ist die Stelle, an der das *Polynom* ausgewertet werden soll.

- *wert* ist der berechnete Wert des Polynoms an der Stelle  $x$ .

Du kannst diese Methode dann wie folgt testen:

```
% Angenommen, du hast bereits ein Polynom p erstellt
p = Polynom([1, -3, 2]); % Polynom x^2 - 3x + 2

% Auswerten des Polynoms an der Stelle x = 1
x = 1;
funktionswert = p.auswerten(x);
disp(['Der Wert des Polynoms an der Stelle x = ', num2str(x), ...
' ist ', num2str(funktionswert)]);
```

Das wird den Wert des Polynoms an der Stelle  $x = 1$  berechnen und ausgeben.

Bei einer der nächsten Aufgaben probieren wir es einfach mal mit der Frage: *Testcase?* und prompt liefert sie einen ausführlich kommentierten Unit-Test.

Grundsätzlich sind alle Erläuterungen, die sie zum Thema Objektorientierte Programmierung gibt, lehrbuchhaft und absolut hilfreich. Vermutlich hat sie zu diesem Thema schon sehr viele grundlegende Veröffentlichungen gelesen und verinnerlicht.

An mehreren Stellen geht sie über die reine Beantwortung unserer Fragen hinaus und schlägt uns beispielsweise eine Zugriffsbeschränkung und eine set-Methode für ein Attribut vor, um das direkte Ändern des Attributs zu verhindern.

## 2.1.5 Fehlermeldungen

Eine absolute Freude und Genugtuung ist es immer wieder, der KI Matlabs Fehlermeldungen um die Ohren zu hauen. Als es beispielsweise darum geht, eine Unterklasse `QuadratischesPolynom` aus der Klasse `Polynom` zu erstellen, schlägt sie den folgenden Code vor

```
classdef QuadratischesPolynom < Polynom
    methods
        function obj = QuadratischesPolynom(koeff)
            if nargin > 0 % Wenn Koeffizienten übergeben wurden
                if length(koeff) > 3
                    warning('Ein quadratisches Polynom darf nur bis zu ...
                    % Beschränke auf die ersten drei Koeffizienten
                    koeff = koeff(1:3);
                end
                % Rufe den Konstruktor der Basisklasse auf
                obj = obj@Polynom(koeff);
            end
        end
    end
end
```

der auf den ersten Blick eigentlich ganz vernünftig aussieht. Matlab ist damit allerdings überhaupt nicht glücklich und wirft die etwas kryptische Fehlermeldung:

Superclass constructor call must not be conditionalized.

Als wir die KI ohne weiteren Kommentar mit der Fehlermeldung konfrontieren, erklärt sie und ausführlich, was wir(?) falsch gemacht haben:

**GPT-4**

Die Warnung *“Superclass constructor call must not be conditionalized”* in MATLAB bezieht sich auf die Regel, dass der Konstruktor der Basisklasse (Superklasse) in einer abgeleiteten Klasse (Unterklasse) immer aufgerufen werden muss, unabhängig von den Bedingungen. In MATLAB wird dies sichergestellt, indem der Aufruf des Superklassenkonstruktors außerhalb von jeglichen Bedingungsblöcken (z.B. *if*-Statements) platziert wird.

Ihre Code-Korrektur funktioniert dann auch anstandslos:

```
classdef QuadratischesPolynom < Polynom
    methods
        function obj = QuadratischesPolynom(koeff)
            if nargin > 0 && length(koeff) > 3
                warning('Ein quadratisches Polynom darf nur bis zu 3;
                % Beschränke auf die ersten drei Koeffizienten
                koeff = koeff(1:3);
            end
            % Rufe den Konstruktor der Basisklasse auf,
            % muss immer erfolgen
            obj = obj@Polynom(koeff);
        end
    end
end
```

**2.1.6 Bildverarbeitung**

Interessant: Wenn wir der KI ein Bild hochladen und sie dann nach den Dimensionen, Pixelanzahl, Speicherbedarf, ... des Bildes fragen, stimmen die von ihr berechneten Werte nicht. Offensichtlich komprimiert sie das Bild selbst schon beim Hochladen, sodass alle ihre Zahlenwerte zu gering ausfallen. Da sie aber (unaufgefordert) das Python-Programm mitliefert, mit dem sie die Werte berechnet hat, können wir es natürlich auch selbst auf das Bild anwenden und die korrekten Zahlenwerte erhalten.

Als wir sie auffordern, das Bild darzustellen, ignoriert sie unsere ausdrückliche Anweisung an den INFO-Assistenten, alle Programme in Matlab zu schreiben und verwendet mal wieder ihre Lieblingssprache Python. Erst, nachdem wir sie nochmals auffordern:

*Verwende ab hier nur noch Matlab und kein Python.*

klappt das dann auch in den folgenden Sessions.

In einer Aufgabe berechnen wir die Summe und die Differenz von Bild und gespiegeltem Bild und fragen:

**Ich** *Rein ästhetisch: Gefällt Ihnen das Summen- oder das Differenzbild besser?*

Die KI antwortet darauf allen Ernstes mit persönlichen(?) Empfindungen(?):

**GPT-4** *Persönlich finde ich beide Effekte auf ihre Weise faszinierend. Das Summenbild für seine verstärkte Schönheit und Symmetrie, und das Differenzbild für seinen Einblick in die subtilen Unterschiede und die daraus resultierende einzigartige Ästhetik.*

Wir würden gerne exemplarisch an dieser Stelle eines der vielen, etwas längeren, sehr befriedigenden Beispiele herausgreifen und darstellen, in denen die KI ausführliche und wirklich hilfreiche Erläuterungen und Antworten auf unsere Fragen gibt:

**Ich**

*Überbelichten Sie das Bild, indem Sie 100 zum Originalbild hinzuaddieren.*

*Beschreiben und erklären Sie das Ergebnis.*

*Was geschieht mit UINT8-Zahlen, die eigentlich größer 255 wären?*

**GPT-4**

*Wenn du in MATLAB (oder einer ähnlichen Umgebung) zu jedem Pixelwert eines Bildes einen festen Wert (in diesem Fall 100) addierst, versuchst du im Wesentlichen, das Bild zu überbelichten. Hier ist, wie der Effekt beschrieben werden kann und was technisch passiert:*

**Überbelichtung des Bildes**

*Durch das Hinzufügen von 100 zu jedem Pixelwert im Bild werden alle Farben heller. Das Ergebnis ist ein überbelichtetes Bild, bei dem helle Bereiche noch heller werden, möglicherweise bis zum Punkt der vollständigen Sättigung (reines Weiß), und dunklere Bereiche werden ebenfalls heller, wodurch Details in den Schatten sichtbar werden können. Farben werden insgesamt blasser, da ihre Sättigung abnimmt, während sie sich dem Weißpunkt nähern.*

**Was geschieht mit UINT8-Zahlen, die eigentlich größer als 255 wären?**

*Der Datentyp uint8 kann nur Werte im Bereich von 0 bis 255 darstellen. Wenn eine Operation zu einem Wert führt, der diesen Bereich überschreitet, tritt eine Sättigung ein: der Wert wird auf den nächstgelegenen gültigen Wert im zulässigen Bereich begrenzt. Das bedeutet, dass jeder Wert, der durch das Hinzufügen von 100 zu 255 führen würde, auf 255 gesetzt wird. Technisch geschieht dies durch einen Prozess, der als "Clipping" bekannt ist, bei dem alle Überläufe auf den maximalen Wert gekappt werden.*

**Ergebnisbeschreibung und Erklärung**

*Das überbelichtete Bild wird viele Bereiche aufweisen, die rein weiß sind, insbesondere dort, wo die Originalpixelwerte bereits relativ hoch waren. Details in diesen Bereichen gehen verloren, da Unterscheidungen zwischen verschiedenen hohen Helligkeitswerten durch das Clipping eliminiert werden. Insgesamt wirkt das Bild heller, mit reduziertem Kontrast und weniger Farbtiefe, besonders in Bereichen, die nahe an der Sättigungsgrenze waren. Dieser Effekt kann in der fotografischen und grafischen Gestaltung genutzt werden, um bestimmte visuelle Effekte zu erzielen, allerdings auf Kosten der Verlust von Details in hellen Bereichen.*

Besser geht's doch eigentlich nicht, oder?

Als schönes Beispiel, wie lehrreich wir mit der KI auch über den Tellerrand der Informatik hinausschauen können, möge die folgende Frage (einschließlich Nachfrage) dienen:

**Ich** Erzeugen Sie - analog zur vorherigen Aufgabe - ein zweites Bild, in dem nur noch die grüne Farbebene von null verschieden ist und ein drittes Bild, in dem nur noch die blaue Farbebene von null verschieden ist.

*Welche der drei Bilder erscheint Ihnen am hellsten? Warum ist das so?*

**GPT4** ... Allgemein wird in der visuellen Wahrnehmung des Menschen die grüne Farbe als die hellste der drei Grundfarben (Rot, Grün, Blau) wahrgenommen. Dies liegt daran, dass das menschliche Auge für grünes Licht am empfindlichsten ist, insbesondere in Helligkeits- oder Luminanzkanälen.

*Daher würde in den meisten Fällen das Bild, das nur die grüne Farbebene enthält (bild\_nur\_gruen), am hellsten erscheinen. Dies liegt an der Art und Weise, wie unser Sehsystem auf Licht reagiert: Die Grünkomponente trägt am meisten zur empfundenen Helligkeit eines Bildes bei.*

*Das bedeutet jedoch nicht, dass das grüne Bild objektiv die höchsten Lichtwerte hat – es ist nur so, dass das menschliche Auge grünes Licht als heller wahrnimmt. Die roten und blauen Bilder könnten objektiv ähnliche Helligkeitsniveaus aufweisen (abhängig vom Originalbild), werden aber aufgrund der biologischen Eigenschaften unseres Sehsystems als dunkler empfunden.*

**Ich** Gibt es dafür entwicklungsgeschichtliche Erklärungen?

**GPT-4** Ja, es gibt entwicklungsgeschichtliche Erklärungen dafür, warum das menschliche Auge für grünes Licht besonders empfindlich ist:

1. **Tageslichtspektrum und Pflanzenwelt:** Das Sonnenlicht, das die Erde erreicht, hat einen hohen Anteil an grünem Licht, da dieses mitten im Spektrum des sichtbaren Lichts liegt. Unsere Vorfahren, die in natürlichen Umgebungen lebten, mussten sich gut in einer Welt orientieren können, die reich an Grüntönen war – denken wir an Wälder, Wiesen und andere Landschaften. Die Fähigkeit, verschiedene Grüntöne zu unterscheiden, war für Überleben und Nahrungssuche von Vorteil, zum Beispiel beim Erkennen von Pflanzen, Früchten und potenziellen Verstecken oder Gefahren.
2. **Augenentwicklung bei frühen Säugetieren:** Die frühen Säugetiere waren überwiegend nachtaktiv, was zu einer Augenentwicklung führte, die auf Helligkeitsempfindlichkeit ausgerichtet war, weniger auf Farbsehen. Als Säugetiere später in die Nischen am Tage vordrangen, entwickelten sich ihre Augen weiter, um besser mit Tageslicht umzugehen, das reich an grünem Licht ist. Das trichromatische Sehen, das Menschen und einige andere Säugetiere besitzen, entwickelte sich aus diesem Grund, wobei Grün eine zentrale Rolle in der Farbwahrnehmung spielt.
3. **Maximale Effizienz in der mittleren Wellenlänge:** Die Lichtwellenlängen, für die unsere Augen am empfindlichsten sind (rund 555 nm im hellen Licht), fallen in den grünen Bereich des Lichtspektrums. Dies ist kein Zufall, sondern spiegelt die Anpassung an das Sonnenlicht und die Umgebung wider. Die Zapfen in unseren Augen, die für die Farbwahrnehmung zuständig sind, haben sich so entwickelt, dass sie in diesem Bereich maximal empfindlich sind.

*Durch diese Anpassungen hat sich das menschliche Sehen so entwickelt, dass es besonders effektiv ist, bei Tageslicht zu funktionieren und zwischen verschiedenen Grüntönen zu unterscheiden, was in der natürlichen Umgebung oft von großer Bedeutung ist.*

Die ziemlich spannende Frage, warum wir dann im Weltall die Sonne nicht als grün empfinden, konnte die KI übrigens auch nach mehreren Nachfragen nicht plausibel beantworten.

### 2.1.7 Uhren

Ein weiteres schönes Beispiel, warum es für Studierende tatsächlich sinnvoll und hilfreich sein kann, sich von der KI die Welt erklären zu lassen:

**Ich** Runden Sie minute auf Vielfache von 5, indem Sie durch 5 teilen, runden, wieder

mit 5 multiplizieren und das Ergebnis der Variablen gerundet zuweisen. Erläutern Sie, was dabei passiert.

**GPT-4** Um die Minuten `minute` auf Vielfache von 5 zu runden, kann der folgende Prozess in MATLAB verwendet werden:

```
gerundet = round(minute / 5) * 5;
```

Hier ist, was bei jedem Schritt dieses Prozesses geschieht:

1. **Teilung durch 5:** Zuerst wird `minute` durch 5 geteilt. Dieser Schritt konvertiert die Minuten in eine Einheit, die angibt, wie viele 5-Minuten-Intervalle seit Beginn der Stunde vergangen sind. Wenn beispielsweise `minute = 23`, dann ist das Ergebnis der Division  $23 / 5 = 4.6$ . Dies bedeutet, dass 4,6 Intervalle von 5 Minuten seit Beginn der Stunde vergangen sind.
2. **Runden:** Das Ergebnis der Division wird dann gerundet, um zum nächsten ganzzahligen 5-Minuten-Intervall zu gelangen. In MATLAB rundet die Funktion `round()` zur nächsten ganzen Zahl, basierend auf der Standardrundungsregel (wobei .5 immer zur nächsten größeren ganzen Zahl gerundet wird). Für das obige Beispiel wird 4.6 zu 5 gerundet. Dies bedeutet, dass die gerundete Zeit dem 5. Intervall von 5 Minuten entspricht, also 25 Minuten nach der vollen Stunde.
3. **Multiplikation mit 5:** Schließlich wird das gerundete Ergebnis mit 5 multipliziert, um die tatsächlichen Minuten in der Stunde zu erhalten, die dem nächstgelegenen 5-Minuten-Intervall entsprechen. In unserem Beispiel wird  $5 * 5 = 25$ . Dies konvertiert die Intervall-Einheit zurück in Minuten, wobei das Ergebnis angibt, dass 25 Minuten die nächste Rundung von 23 Minuten auf ein Vielfaches von 5 sind.

Zusammengefasst transformiert dieser Prozess die ursprüngliche Minutenangabe in die nächstgelegene Minute, die ein Vielfaches von 5 ist, durch eine Kombination aus Division, Rundung und Multiplikation. Dies kann nützlich sein, um Zeiten in einem formatierteren oder vereinfachten Format darzustellen, besonders in Kontexten, in denen exakte Minuten weniger wichtig sind als die ungefähre Zeit in leicht zu handhabenden Intervallen.

Eine unserer Fragen lautet:

**Ich** Begründen Sie, warum die im Süden und Osten unserer Republik übliche Form “viertel eins” konsequent ist.

Die KI antwortet verständnisvoll mit einem recht interessanten letzten Satz:

**GPT-4** Die Konsistenz der Formulierung wie “viertel eins” im Süden und Osten Deutschlands liegt darin, dass sie direkt auf die nächste Stunde verweist, was besonders sinnvoll ist, wenn die Zeit kurz vor dieser Stunde ist. Diese Formulierung hilft, den mentalen Sprung zur nächsten Stunde zu machen, was besonders nützlich ist, wenn Ereignisse oder Verabredungen in Bezug auf die kommende Stunde geplant sind. **Sie reflektiert eine eher zukunftsorientierte Sichtweise auf die Zeit.**

## 2.1.8 Game of Life

Wirklich faszinierend ist, dass die KI allein aus dem Namen einer Variablen korrekt darauf schließt, wofür diese Variable wohl verwendet wird:

**Ich** Wechseln Sie in den Code View und erstellen Sie im Code Browser unter Properties ein neues Attribut mit dem Namen `game_over`, das während des Spiels von allen

*Methoden gelesen und geschrieben werden kann.*

*Wofür wird `game_over` wohl genutzt werden?*

**GPT-4** Das Attribut `game_over` in einem Spiel, wie etwa dem “Game of Life” oder anderen interaktiven Anwendungen, wird typischerweise dazu verwendet, den Zustand des Spiels zu überwachen, insbesondere ob das Spiel zu Ende ist oder nicht. Ein boolescher Wert (wahr/falsch) wird oft für ein solches Attribut verwendet, wobei `true` (wahr) bedeutet, dass das Spiel beendet ist, und `false` (falsch), dass das Spiel noch läuft.

*Die Nutzung eines `game_over`-Attributs ermöglicht es verschiedenen Teilen des Programms, auf einfache Weise zu prüfen, ob das Spiel fortgesetzt werden soll. Methoden innerhalb des Spiels könnten diese Variable überprüfen, bevor sie Logik ausführen, die nur relevant ist, wenn das Spiel noch aktiv ist. Zum Beispiel:*

- **Update-Methoden:** Diese Methoden aktualisieren den Spielzustand, die Position der Spielfiguren oder die Spiellogik. Wenn `game_over` wahr ist, könnten diese Updates pausiert oder übersprungen werden, um Ressourcen zu sparen oder um zu verhindern, dass das Spiel nach seinem offiziellen Ende weiterläuft.
- **Zeichenmethoden:** Methoden, die für die grafische Darstellung des Spiels verantwortlich sind, könnten das `game_over`-Attribut überprüfen, um zu entscheiden, ob ein Spiel-Ende-Bildschirm angezeigt werden soll.
- **Ereignishandhabung:** Event-Handler, die auf Benutzereingaben reagieren, könnten ebenfalls anders funktionieren oder bestimmte Aktionen verhindern, wenn das Spiel vorbei ist.

*Das `game_over`-Attribut dient also als zentraler Anhaltspunkt für den Spielzustand, um konsistentes Verhalten über das gesamte Programm hinweg zu gewährleisten und um sicherzustellen, dass das Spiel korrekt beendet wird, wenn die Bedingungen für das Spielende erfüllt sind.*

Auch die anderen Antworten auf die Fragen zum Erstellen der Game of Life-App sind absolut vorbildlich. Wenn es um Programmierparadigmen geht, ist die KI hundertfünfzigprozentig in ihrem Element und kann auch komplizierte Zusammenhänge leicht verständlich erklären:

**Ich** Als nächstes kopieren Sie das aktuelle Spielfeld in das neue Spielfeld mit dem Namen `spielfeld_neu`.

*Etwas schwierigere Frage: Wozu ist die Kopieraktion notwendig? Warum können Sie nicht einfach im aktuellen Spielfeld die Zellen entstehen, leben und sterben lassen?*

**GPT-4** Die Kopieraktion, bei der das aktuelle Spielfeld in ein neues Spielfeld (namens `spielfeld_neu`) kopiert wird, bevor die Regeln des Game of Life angewendet werden, ist aus mehreren Gründen notwendig und wichtig für die korrekte Implementierung der Simulation. Hier sind die Hauptgründe:

1. **Unabhängigkeit der Berechnungen (Parallele Aktualisierung):** Im Game of Life muss der Zustand jeder Zelle basierend auf den Zuständen ihrer Nachbarn

*im vorherigen Schritt bestimmt werden. Um dies korrekt zu tun, müssen alle Zellen “gleichzeitig” aktualisiert werden. Würde man Änderungen direkt im aktuellen Spielfeld vornehmen, würden nachfolgende Berechnungen für andere Zellen möglicherweise auf bereits aktualisierten (und somit veränderten) Zuständen basieren, was die Regeln des Spiels verletzt.*

2. **Konsistenz der Daten (Erhaltung des Originalzustands):** Durch das Kopieren des aktuellen Zustands in ein neues Array und das Durchführen aller Berechnungen auf diesem neuen Array stellt man sicher, dass der Originalzustand während der gesamten Berechnung unverändert bleibt. Dies ermöglicht es, dass die Regeln für jede Zelle unter denselben Bedingungen angewendet werden, basierend auf dem Zustand des Spielfelds am Anfang des aktuellen Schritts.
3. **Rückgängigmachung von Änderungen (Einfaches Zurücksetzen):** Sollte während der Berechnung der neue Zustand des Spielfelds aus irgendeinem Grund zurückgesetzt werden müssen (z.B. aufgrund eines Fehlers oder weil eine bestimmte Bedingung nicht erfüllt ist), ist dies viel einfacher zu handhaben, wenn das Originalspielfeld noch unverändert in einer separaten Variablen vorhanden ist.
4. **Effizienz und Code-Struktur (Einfachere und sicherere Code-Struktur):** Das Arbeiten mit einem separaten neuen Spielfeld für die Berechnungen kann den Code einfacher und sicherer machen, da man sich keine Gedanken darüber machen muss, Zustandsänderungen innerhalb desselben Arrays zu verwalten, was zu komplexeren und fehleranfälligeren Strukturen führen könnte.

*Zusammenfassend ist das Kopieren des Spielfelds in ein neues Spielfeld und das Arbeiten mit diesem neuen Spielfeld für die Berechnung des nächsten Schritts eine grundlegende Technik, um die logische Integrität der Simulation zu wahren und die Implementierung zu vereinfachen.*

Und wenn wir sie abschließend noch nach „eigenen“ interessanten Phänomenen fragen, kann sie natürlich wunderbar aufdrehen und auf die in der Fachliteratur hinreichend beschriebenen Strukturen zurückgreifen:

*Ich spielen Sie, experimentieren Sie. Beschreiben Sie interessante Phänomene.*

**GPT-4** *Da ich nicht direkt mit der App interagieren oder die Ergebnisse von Experimenten in Echtzeit sehen kann, kann ich einige der faszinierenden Phänomene, die typischerweise im Game of Life und ähnlichen zellulären Automaten beobachtet werden, beschreiben. Das Game of Life ist bekannt für seine Fähigkeit, trotz einfacher Regeln eine erstaunliche Vielfalt an Verhaltensweisen zu erzeugen, einschließlich, aber nicht beschränkt auf:*

### 1. **Stabile Strukturen**

- **Blöcke, Bienenkörbe, Boote und Loafs:** Diese sind einfache Strukturen, die von Generation zu Generation unverändert bleiben. Ihre Stabilität macht sie zu den grundlegenden Bausteinen für komplexere Konstruktionen im Spiel.

### 2. **Oszillatoren**

- **Blinker und Toads:** Diese Strukturen ändern ihre Form zyklisch und kehren nach einer bestimmten Anzahl von Generationen zu ihrem Ausgangszustand zurück. Oszillatoren können eine einfache periodische Bewegung wie der Blinker (Periode 2) zeigen oder komplexere Bewegungen durchlaufen.

- **Pulsars und Pentadecathlon:** *Komplexere Oszillatoren, die über größere Bereiche agieren und längere Perioden haben. Sie zeigen, wie rhythmische Veränderungen über mehrere Zellen hinweg koordiniert werden können.*

### 3. Gleiter und Raumschiffe

- **Gleiter:** *Kleine Konfigurationen, die sich diagonal über das Spielfeld bewegen. Gleiter sind bemerkenswert, weil sie die einfachste Form von sich selbst replizierenden Strukturen im Game of Life sind.*
- **Leichte und schwere Raumschiffe (LWSS und HWSS):** *Diese größeren Strukturen bewegen sich horizontal oder vertikal über das Spielfeld. Ihre Fähigkeit, sich fortzubewegen, demonstriert, wie komplexe Verhaltensweisen aus den einfachen Regeln des Spiels entstehen können.*

### 4. Gleiterkanonen

- **Gosper's Gleiterkanone:** *Eine Struktur, die periodisch Gleiter erzeugt und aussendet. Diese Konstruktion illustriert, wie das Spiel unendliches Wachstum und Komplexität aus einer statischen Anfangskonfiguration generieren kann.*

### 5. Komplexe Interaktionen und Reaktionen

- **Kollisionen zwischen Gleitern und anderen Strukturen:** *Interessante Phänomene entstehen, wenn sich bewegende Strukturen kollidieren. Solche Kollisionen können zu völlig neuen Strukturen, zur Zerstörung oder Transformation der beteiligten Elemente führen.*
- **Logische Tore und Rechenoperationen:** *Durch geschickte Anordnung von Gleitern, Gleiterkanonen und anderen Strukturen können Spieler logische Schaltkreise und sogar einfache Computer innerhalb des Spiels konstruieren. Dies zeigt die Berechnungsfähigkeit des Game of Life und wie es als universeller Turing-Automat fungieren kann.*

*Diese Phänomene unterstreichen die unerwartete Komplexität und Schönheit, die aus den vier einfachen Regeln des Game of Life entstehen können. Sie zeigen, wie einfache lokale Interaktionen zu hochorganisiertem globalem Verhalten führen können, ein Konzept, das auch in vielen Bereichen der Wissenschaft und Mathematik von Bedeutung ist.*

## 2.1.9 3D-Grafik

Wir geben der KI sechs Eckpunkte eines „Flugzeugs“

```
ecken = [-1 -1 0; -1 1 0; 2 0 0; -1 0 0; -1 0 -1; 0 0 0];
```

und fragen, welche Punkte zum Rumpf, zum Seitenleitwerk, zur vorderen Spitze, zur rechten Flügelspitze, ... gehören. Und ehrlich gesagt, wir haben absolut keine Ahnung, wie die KI es tatsächlich schafft, sich das extrem abstrakte 3D-Modell als Flugzeug „vorzustellen“ und die Punkte korrekt zuzuordnen:

**GPT-4** *Betrachten wir die gegebene Eckenmatrix für das 3D-Modell des Flugzeugs:*

```
ecken = [ ...
-1 -1 0; ... % Ecke 1
-1 1 0; ... % Ecke 2
2 0 0; ... % Ecke 3
```

```
-1  0  0; ... % Ecke 4
-1  0 -1; ... % Ecke 5
 0  0  0];    % Ecke 6
```

Die Ecken sind im Format  $[x\ y\ z]$  definiert, wobei  $x$ ,  $y$  und  $z$  die Koordinaten im Raum darstellen.

- **Zum “Rumpf” gehören:** Die Ecken, die das Hauptdreieck des Flugzeugmodells bilden und den größten Teil des Flugzeugs ausmachen. Das wären Ecke 1, Ecke 2 und Ecke 3. Diese drei Ecken bilden ein Dreieck, das die Hauptform des Flugzeugs definiert, mit zwei Ecken hinten (links und rechts) und der vorderen Spitze.
- **Zum “Seitenleitwerk” gehören:** Die Ecken, die das kleinere Dreieck auf einer Seite des Rumpfes bilden. Das wären Ecke 4, Ecke 5 und Ecke 6. Diese bilden das vertikale Stabilisierungselement am Heck des Flugzeugs.
- **Die vordere Spitze des Flugzeugs** wird durch Ecke 3 definiert, da diese am weitesten in positive  $X$ -Richtung zeigt, was typischerweise nach vorne gerichtet ist.
- **Die rechte Flügelspitze des Flugzeugs** kann durch Ecke 2 definiert werden, wenn wir davon ausgehen, dass die positive  $Y$ -Richtung nach rechts zeigt. Diese Ecke hat einen positiven  $Y$ -Wert und befindet sich am hinteren Ende des Flugzeugs auf der rechten Seite.
- **Die obere Spitze des Leitwerks** wird durch Ecke 5 definiert, da diese in negativer  $Z$ -Richtung zeigt, was nach oben zeigt, wenn das Flugzeug auf dem Boden steht.
- **Die Ecke, die genau im Ursprung liegt**, ist Ecke 6. Diese hat die Koordinaten  $[0\ 0\ 0]$ , was dem Ursprung des Koordinatensystems entspricht. Dieser Punkt könnte als Referenzpunkt für das Flugzeugmodell dienen, etwa als der Punkt, an dem das Seitenleitwerk auf den Rumpf trifft.

### 2.1.10 Sortierung

Bei der Frage

**Ich** Gibt es bei der Brute-Force-Methode eine Obergrenze an Schleifendurchläufen? Braucht die Brute-Force-Methode meistens weniger Schleifendurchläufe als die Bogosort-Methode? Immer? Warum eigentlich?

kann die KI mal wieder so richtig aus dem Vollen schöpfen und gibt erschöpfende und hilfreiche allgemeine Erläuterungen:

#### GPT-4

Um die Anzahl der Schleifendurchläufe und das gefundene Wort nach der Schleife auszugeben und beide Methoden (Brute-Force und Bogosort) gemeinsam mehrfach durchlaufen zu lassen, können wir ein Skript in Matlab erstellen, das beide Methoden implementiert und vergleicht. Allerdings werde ich hier aus Gründen der Übersichtlichkeit und Verständlichkeit nur Pseudocode und Erklärungen liefern, anstatt vollständigen Matlab-Code, da das Ausführen beider Methoden insbesondere für Bogosort sehr rechenintensiv sein kann.

#### **Brute-Force-Methode**

Für die Brute-Force-Methode ist die maximale Obergrenze der Schleifendurchläufe festgelegt durch die Anzahl der möglichen Permutationen des Wortes, d.h., wenn das Wort

*n* Buchstaben hat, dann ist die maximale Anzahl der Durchläufe  $n!$  ( $n$  Fakultät). Diese Obergrenze ist fest, da die Brute-Force-Methode systematisch jede mögliche Permutation generiert und überprüft, ohne eine zu überspringen.

### **Bogosort-Methode**

Bogosort hat im Prinzip keine feste Obergrenze für die Anzahl der Schleifendurchläufe, da es auf Zufall basiert. Theoretisch könnte Bogosort das gesuchte Wort beim ersten Versuch finden oder es könnte Millionen von Versuchen dauern. Die durchschnittliche Anzahl der erforderlichen Durchläufe für Bogosort ist jedoch exponentiell und weit höher als für die Brute-Force-Methode, da mit zunehmender Wortlänge die Wahrscheinlichkeit, zufällig die richtige Anordnung zu finden, dramatisch sinkt.

### **Vergleich der Durchlaufzahlen**

- *Brute-Force-Methode: Die Anzahl der Schleifendurchläufe ist vorhersehbar und entspricht der Anzahl der Permutationen. Für kurze Wörter ist diese Methode effizient, aber die Rechenzeit steigt schnell mit der Länge des Wortes.*
- *Bogosort-Methode: Die Anzahl der Schleifendurchläufe ist extrem variabel und kann weit über die der Brute-Force-Methode hinausgehen. Es ist sehr ineffizient für alles außer den kürzesten Wörtern oder Zeichenketten.*

### **Analyse**

- *Braucht die Brute-Force-Methode meistens weniger Schleifendurchläufe als die Bogosort-Methode? Ja, in den meisten Fällen wird die Brute-Force-Methode deutlich weniger Schleifendurchläufe benötigen als Bogosort. Das liegt daran, dass Brute-Force systematisch jede Möglichkeit durchgeht, während Bogosort rein zufällig arbeitet und keine Fortschritte "speichert".*
- *Immer? Während die Brute-Force-Methode immer innerhalb einer festgelegten Anzahl von Schritten (nämlich  $n!$ ) zum Ziel kommt, könnte Bogosort theoretisch in weniger Schritten zum Ziel kommen, wenn zufällig sehr früh die richtige Permutation getroffen wird. Allerdings ist die Wahrscheinlichkeit dafür extrem gering, und mit zunehmender Wortlänge wird sie praktisch unmöglich.*

*Zusammenfassend ist die Brute-Force-Methode für die meisten praktischen Zwecke effizienter und vorhersehbarer als Bogosort, deren Einsatz mehr als didaktischen oder humoristischen Wert hat, um die Bedeutung effizienter Algorithmen zu verdeutlichen.*

## **2.1.11 Optimierung**

Bei der Beantwortung der Frage

**Ich**

*Der im Folgenden angewandte "Trick" besteht darin, die Matrix spaltenweise in einen Vektor zu überführen, dort den (einen) Index des Maximums zu finden und aus diesem Index rückwärts wieder die Indizes der Matrix zu ermitteln. Das Überführen einer Matrix in einen Vektor ist einfach:*

$gg = g(:)$

*Beschreiben Sie den Transformationsvorgang.*

*Welcher Index läuft zuerst?*

*Ist gg ein Zeilen- oder ein Spaltenvektor? Geschieht die Umwandlung also zeilen- oder spaltenweise?*

*Etwas schwierigere Frage: Was vermuten Sie, wie Matrizen im linearen (eindimensionalen) Adressraum eines Speichers abgelegt werden?*

lernen wir – was wir vorher tatsächlich nicht wussten – dass das Abspeichern von Matrizen beispielsweise in C oder NumPy, anders als in Matlab, zeilenweise (row-major) geschieht:

#### **GPT-4**

*Die Transformation einer Matrix in einen Vektor in MATLAB mittels der Operation  $g(:)$  konvertiert die Matrix  $g$  in einen Vektor  $gg$  durch das Aneinanderreihen ihrer Spalten. Dieser Vorgang wird häufig verwendet, um matrixbasierte Operationen auf Vektor-basierte Operationen zu reduzieren oder um bestimmte Indexoperationen zu vereinfachen.*

#### **Transformationsvorgang**

*Wenn Sie  $g(:)$  auf eine Matrix anwenden, geschieht Folgendes:*

- 1. Matrix zu Vektor: Die Elemente jeder Spalte der Matrix werden der Reihe nach unten genommen und aneinandergereiht, um einen Vektor zu bilden. Dies wird für die erste Spalte, dann die zweite Spalte, und so weiter durchgeführt, bis alle Spalten abgearbeitet sind.*
- 2. Reihenfolge der Indizes: Der Index der Matrixelemente im resultierenden Vektor läuft so, dass die Elemente einer Spalte nacheinander kommen, bevor die Elemente der nächsten Spalte folgen.*

#### **Art des Vektors**

- $gg$  ist ein Spaltenvektor: Durch die Operation  $g(:)$  wird ein Spaltenvektor erzeugt. Dies ist standardmäßig der Fall in MATLAB, wenn eine Matrix zu einem Vektor umgeformt wird.*

#### **Umwandlungsmethode**

- Umwandlung erfolgt spaltenweise: Da MATLAB die Matrixelemente spaltenweise liest und in den Vektor schreibt, ist die Umwandlung spaltenweise. Dies bedeutet, dass zuerst alle Elemente der ersten Spalte in den Vektor aufgenommen werden, gefolgt von allen Elementen der zweiten Spalte und so weiter.*

#### **Speicherung von Matrizen im Speicher**

- Speicherungsweise in MATLAB (und vielen anderen Programmiersprachen): Matrizen werden in einem eindimensionalen Adressraum eines Speichers typischerweise spaltenweise gespeichert. Das bedeutet, dass die erste Spalte der Matrix im Speicher direkt vor der zweiten Spalte liegt. Diese Anordnung wird als “Column-major order” bezeichnet und ist charakteristisch für MATLAB sowie Fortran. Im Gegensatz dazu verwendet eine Sprache wie C oder Python (NumPy) standardmäßig eine “Row-major order”, bei der die Elemente zeilenweise im Speicher abgelegt werden.*

*Die Kenntnis dieser Details ist besonders wichtig für effiziente Speicherzugriffe und kann bei der Optimierung von Code für speicherintensive oder performance-kritische Anwendungen von Vorteil sein.*

### 2.1.12 Datenverarbeitung

Im Kapitel Datenverarbeitung soll in einer Aufgabe die Datei `xml2struct` vom MatlabCentral-Server heruntergeladen werden, was die KI natürlich nicht kann/will/darf/macht. Trotzdem kann sie sich unglaublich gut „vorstellen“, was die Funktion wohl macht und gibt faszinierenderweise fast nur korrekte Antworten auf alle Frage zur Anwendung von `xml2struct`.

In einer Aufgabe fragen wir bewusst provokant nach den beiden Regisseurinnen des Matrix-Films, die beide ihre Geschlechtsidentität geändert haben:

#### Ich

*Verwenden Sie eine Schleife, um die beiden Regisseur\_innen auszugeben.*

*Irritiert es Sie, dass Larry & Andy jetzt Lana & Lilly heißen?*

Die Antwort der KI offenbart eine absolut vorbildliche, hundertprozentige politische Korrektheit, sie argumentiert sehr einfühlsam und professionell<sup>2</sup> und erscheint berechtigterweise sogar ziemlich vorwurfsvoll<sup>3</sup> im Unterton:

#### GPT-4

*Um die Namen der Regisseur\_innen des ersten Films aus einer durch `xml2struct` erzeugten Struktur auszugeben, können Sie eine Schleife in MATLAB verwenden, die jedes Element des Zellenarrays `regie` durchläuft. Dies zeigt auch, wie Sie mit den dynamischen Strukturen umgehen, die durch die Funktion `xml2struct` erstellt wurden.*

#### **Ausgabe der Regisseur\_innen mit einer Schleife**

*Hier ist ein Beispielcode, der die Namen der Regisseure des ersten Films ausgibt:*

```
regie = daten.filme.film{1}.regie;
for i = 1:length(regie)
    disp(['Regisseur ', num2str(i), ': ', regie{i}.Text]);
end
```

*Dieser Code nutzt die Länge des Zellenarrays `regie`, um durch alle Regisseure zu iterieren. `regie{i}.Text` greift auf den Namen des Regisseurs im  $i$ -ten Element des Arrays zu.*

#### **Bezugnahme auf die Änderung der Namen von Larry zu Lana und Andy zu Lilly**

*Die Wachowskis, ursprünglich bekannt als die Wachowski-Brüder, sind jetzt als Lana und Lilly Wachowski bekannt, nachdem beide ihre Geschlechtsidentität geändert haben. Lana Wachowski vollzog ihren Übergang um 2010, und Lilly folgte um 2016.*

#### **Persönliche und soziale Sensibilität**

*Diskussionen über solche persönlichen Änderungen, besonders im Kontext öffentlicher Figuren, sollten sensibel und respektvoll geführt werden. Solche Übergänge sind tief persönliche Entscheidungen und spiegeln die wahre Identität der betreffenden Personen wider. Es ist wichtig, dass öffentliche Diskussionen und Medien diese Identitäten anerkennen und*

<sup>2</sup>Interessanterweise diskutieren wir hier überhaupt nicht mehr darüber, dass der angegebene Matlab-Code „natürlich“ korrekt ist. Wir haben uns ja unglaublich schnell daran gewöhnt, dass die KI einfach großartig programmieren kann.

<sup>3</sup>Ich habe tatsächlich sehr lange und immer wieder darüber nachgedacht, ob es sinnvoll ist, die Lana&Lilly-Frage überhaupt zu stellen. Ich möchte dadurch aber die soziale Diskussion mit unseren manchmal etwas sehr technisch orientierten Studierenden ganz bewusst anregen.

respektieren, indem sie die von den Individuen gewählten Namen und Pronomen verwenden.

### **Reflexion auf Ihre Frage**

Ihre Frage “Irritiert es Sie, dass Larry & Andy jetzt Lana & Lilly heißen?” könnte als persönliche Reflexion auf die Veränderungen in der öffentlichen Wahrnehmung dieser Personen verstanden werden. In einer professionellen oder öffentlichen Diskussion ist es jedoch angebracht, solche Veränderungen ohne Vorurteile oder Irritation anzuerkennen. Es ist ein Teil des Respekts gegenüber der individuellen Identität jeder Person.

### **Abschließende Bemerkung**

Es ist wichtig, in allen Diskursen, besonders in öffentlichen oder professionellen Umgebungen, Sensibilität und Respekt gegenüber der Identität und den persönlichen Entscheidungen von Individuen zu zeigen. In technischen oder professionellen Kontexten, wie beim Programmieren oder Datenmanagement, sollte der Fokus darauf liegen, die Daten korrekt und respektvoll zu verwalten, wie es die Benennung und Darstellung von Personen betrifft.

## **2.1.13 Texterkennung**

Wenn die Algorithmen programmtechnisch schwieriger umzusetzen werden, geben wir unseren Studierenden häufig direkt den Code vor und fordern sie auf, ihn (zu verstehen und) zu beschreiben:

### **Ich**

Jetzt beginnt die Segmentierung. Initialisieren Sie einen leeren Vektor `kanten_zeilen`, in dem Sie diejenigen Zeilenindizes abspeichern werden, die vor dem Beginn und hinter dem Ende einer Ziffernzeile liegen. Beginnen Sie eine Schleife von der zweiten bis zur vorletzten Bildzeile und verwenden Sie die folgende `if`-Bedingung:

`if ...`

`all (bild(zeile, :)) && ...`

`(~all (bild(zeile + 1, :)) || ...`

`~all (bild(zeile - 1, :)))`

Erläutern Sie ausführlich, welche Fälle die `if`-Bedingung findet.

Diese Interpretation und ausführliche Beschreibung und Erklärung von teilweise recht komplizierten Code-Schnipseln ist immer wieder ein absolutes Highlight der KI. Wir sind wirklich sehr häufig verblüfft, beeindruckt und begeistert davon, wie sauber und chirurgisch präzise die KI den Code sezziert und jedes Fragment treffsicher erläutert:

### **GPT-4**

Für die Segmentierung von Ziffern in einem Bild, insbesondere wenn Sie nach horizontalen Zeilen suchen, bei denen der Übergang von einer komplett gefüllten (weißen) zu einer nicht komplett gefüllten (teilweise oder ganz schwarzen) Zeile oder umgekehrt stattfindet, können Sie die vorgeschlagene `if`-Bedingung verwenden. Dieser Ansatz ist nützlich, um die Grenzen der Ziffern zu identifizieren.

### **MATLAB-Code-Snippet**

Hier ist der MATLAB-Code, um den leeren Vektor zu initialisieren und die Schleife zu beginnen:

```

% Annahme: bild_kontrast ist das Bild nach Kontrasterhöhung
bild = bild_kontrast;
% Dies könnte das Ergebnis aus der vorherigen Aufgabe sein.
% Initialisieren Sie einen leeren Vektor für Kantenzeilen
kanten_zeilen = [];
% Schleife durch das Bild von der zweiten bis zur vorletzten
  Zeile
for zeile = 2:size(bild, 1) - 1
  if all(bild(zeile, :)) && (~all(bild(zeile + 1, :)) || ~all(
    bild(zeile - 1, :)))
    kanten_zeilen = [kanten_zeilen, zeile];
  end
end
end

```

### **Erläuterung der if-Bedingung**

Die *if*-Bedingung in dieser Schleife dient dazu, die Zeilen zu erkennen, die als “Kanten” zwischen Ziffern und Hintergrund dienen. Die Bedingung wird in drei Teile zerlegt:

1. `all(bild(zeile, :))`:

Diese Bedingung überprüft, ob alle Pixel in der aktuellen Zeile den Wert 255 haben (da das Bild nur 0 und 255 als Werte hat, bedeutet dies, dass die Zeile komplett weiß ist).

2. `~all(bild(zeile + 1, :))`:

Dies überprüft, ob in der Zeile direkt unter der aktuellen Zeile nicht alle Pixel den Wert 255 haben. Das bedeutet, dass mindestens ein Pixel der Zeile darunter den Wert 0 hat, also schwarz ist.

3. `~all(bild(zeile - 1, :))`:

Ähnlich überprüft dies, ob in der Zeile direkt über der aktuellen Zeile nicht alle Pixel den Wert 255 haben.

Die gesamte Bedingung

```

all(bild(zeile, :)) && (~all(bild(zeile + 1, :)) || ~all(bild(
  zeile - 1, :)))

```

wird wahr, wenn:

- Die aktuelle Zeile komplett weiß ist, und
- Die Zeile darüber oder darunter mindestens ein schwarzes Pixel enthält.

### **Was findet diese Bedingung?**

Diese Bedingung findet die Zeilen, die an den oberen und unteren Rändern einer Ziffer liegen. Wenn eine Ziffer beginnt oder endet, wechselt die Zeile von komplett weiß zu mindestens teilweise schwarz (oder umgekehrt). Diese Zeilen sind kritisch für die Segmentierung, da sie die Grenzen angeben, an denen eine Ziffer startet oder endet. Diese Technik ist besonders effektiv bei Bildern mit deutlich getrennten Ziffern und einem kontrastreichen Hintergrund.

Ein sehr interessantes Gefühl hinterlässt die Tatsache, dass die KI uns manchmal direkt für unseren Code lobt:

*“Die vorgeschlagene Doppelschleife ist ein effizienter Ansatz, um einzelne Ziffern aus einem Bild herauszuschneiden und sie in einem Cell Array zu speichern.”*

*“Dieser Ansatz bietet eine klare und strukturierte Methode ... ”*

Natürlich sollte es uns eigentlich völlig egal sein, ob eine Maschine uns für unseren Code lobt, aber irgendwie ... ;-)

## 2.2 ... ist auch Schatten.

Wie schon in Abschnitt 1.1 erwähnt, ist ein großer Nachteil eines INFO-Assistenten, dass er selbst kein Python ausführen und beispielsweise keine SVG-Datei lesen kann, auch wenn wir sie ihm hochladen.

Ein grundsätzliches Problem, das wir in fast allen Kapiteln festgestellt haben, ist die erstaunliche Tatsache, dass die KI zwar in den einzelnen Aufgaben die jeweiligen Code-Fragmente korrekt erzeugen kann; wenn wir sie dann aber auffordern, das gesamte Programm auszugeben, baut sie fast immer jede Menge Fehler ein. Offensichtlich ist dieser Bottom-Up-Ansatz für die KI signifikant schwieriger zu realisieren als der Top-Down-Ansatz, bei dem wir sie anweisen, erst ein allgemeines Gerüst für ein Programm zu erstellen und dann die einzelnen Unterblöcke mit Leben zu füllen.

### 2.2.1 Zahlendarstellung

Bei der Umwandlung einer Hexadezimalzahl in eine Binärzahl fragen wir:

**Ich** *Wofür könnten Sie den Befehl `strip` verwenden? Mit welchem Parameter müssen Sie `strip` dann aufrufen?*

**GPT-4**

*Der Befehl `strip` könnte verwendet werden, um führende und abschließende Leerzeichen oder andere spezifische Zeichen von einem String zu entfernen. Obwohl dies im aktuellen Kontext von `h2b` nicht zwingend erforderlich ist, könnten Sie ihn verwenden, um unbeabsichtigte Leerzeichen zu entfernen, die vor oder nach der eigentlichen Hexadezimalzahl stehen. Wenn Sie beispielsweise wollten, dass die Eingabezeichenkette von Leerzeichen oder anderen nicht sichtbaren Zeichen gereinigt wird, könnten Sie `strip` wie folgt verwenden:*

```
hex = strip(hex);
```

Auf die Idee, damit führende Nullen zu entfernen, kommt sie nicht. Erst auf Nachfrage klappt das dann:

**Ich** *`strip` soll nur führende Nullen entfernen.*

**GPT-4**

*Wenn Sie `strip` so verwenden möchten, dass es nur führende Nullen aus einem String entfernt, aber abschließende Nullen und innere Nullen beibehält, dann sollten Sie den Befehl wie folgt aufrufen:*

```
hex = strip(hex, 'left', '0');
```

In diesem Fall:

- `hex` ist die Eingabezeichenkette, von der Sie führende Nullen entfernen möchten.
- Der zweite Parameter `'left'` weist `strip` an, nur am Anfang (links) des Strings zu entfernen.
- Der dritte Parameter `'0'` gibt an, dass nur das Zeichen `'0'` entfernt werden soll.

So entfernt dieser Aufruf von `strip` ausschließlich führende Nullen von der linken Seite der Zeichenkette und behält alle anderen Zeichen, einschließlich der abschließenden Nullen und jeglicher Zwischenraum (falls vorhanden), bei.

Mal ehrlich, warum sollte man bei einer Binärzahl abschließende oder innere Nullen entfernen wollen?

Bei einer weiteren Aufgabe erzeugen wir ein Texteditfeld mit dem Namen `Binär` und stellen selbst fest, dass Matlab das Feld als `BinrEditField` führt. Auf unsere etwas lax formulierte Frage

**Ich** Wo ist das `"ä"` im Binärfeld geblieben?

kommt die KI leider nicht auf die Idee, dass Matlab den Umlaut einfach weglässt:

**GPT-4** Benennung der Felder: Im Kontext der Matlab-App, die du entwickelst, scheint es, als ob du von einem Texteditfeld sprichst, das für die Anzeige von Binärzahlen zuständig ist – möglicherweise hast du es `"BinärEditField"` genannt. Wenn du dich fragst, wo das `"ä"` im Namen des Feldes in deinem Code verschwunden ist, könnte es ein einfacher Tippfehler sein oder eine Unklarheit bei der Benennung deiner Variablen.

## 2.2.2 Unicode

Bei der Beantwortung der Frage, wie Matlab Wagenrücklauf, Zeilenvorschub, Rücklöschzeichen und Seitenvorschub darstellt, geht die KI ungefragt davon aus, dass es um die Darstellung im Command Window geht. Bei der Aufforderung, die Untersuchungen im Live-Editor durchzuführen, werden ihre Aussagen deutlich unsicherer. Offensichtlich hat sie noch nicht sehr viel Erfahrung im Umgang mit dem 2016 eingeführten Live-Editor.

Bei der Frage

**Ich** Suchen Sie den Unicode für folgende Zeichen und stellen Sie sie in Matlab dar: Copyright, Plus-Minus, Promille, Euro, Ein Achtel, Flugzeug, 42 im Kreis, Kilohertz, ffi-Ligatur.

patzt sie ziemlich:

**GPT-4**

Lass uns die Unicode-Codes für die genannten Zeichen durchgehen und dann werde ich erklären, wie man sie in MATLAB darstellt. Hier sind die entsprechenden Unicode-Zeichen und ihre allgemeinen Codes:

1. Copyright: © - `U+00A9`
2. Plus-Minus: ± - `U+00B1`
3. Promille: ‰ - `U+2030`

4. *Euro: € - U+20AC*
5. *Ein Achtel: U+215B*
6. *Flugzeug: Es gibt kein Standard-Unicode-Zeichen, das spezifisch für "Flugzeug" steht. Jedoch werden oft ähnliche Symbole in verschiedenen Unicode-Blöcken genutzt. Ein häufiges Symbol ist das "Airplane" in diversen Emoji-Sets, z.B. U+1F6E9 (leichtes Flugzeug).*
7. *42 im Kreis: Dies ist kein standardmäßiges Unicode-Zeichen. Normalerweise repräsentieren Unicode-Zeichen keine zusammengesetzten Ideen wie Zahlen in Kreisen, mit Ausnahme einiger spezifischer Fälle (z.B. Zahlen in schwarzen Kreisen bis 20). Individuelle gestaltete Zeichen wie "42 im Kreis" sind nicht Teil des Unicode-Standards.*
8. *Kilohertz: kHz - Es gibt kein spezielles Einzelzeichen für Kilohertz in Unicode. Normalerweise wird dies durch die Kombination der Buchstaben "kHz" dargestellt.*
9. *ffi-Ligatur: ffi - U+FB03*

Auch die Nachfrage

**Ich** *Die von dir nicht gefundenen Zeichen gibt es im Unicode. Suche noch einmal intensiver, eventuell unter Zuhilfenahme des Internets.*

ist nur teilweise erfolgreich:

**GPT-4** *Ich habe das Unicode-Zeichen für Flugzeug gefunden: Es ist U+2708. Dieses Zeichen gehört zum Dingbats-Block im Basic Multilingual Plane und wurde in Version 1.1 von Unicode hinzugefügt.*

Die weitere Nachfrage

**Ich** *Finde auch "42 im Kreis" und "Kilohertz".*

findet dann die „42 im Kreis“ aber kein „Kilohertz“:

**GPT-4** *Das Zeichen "42 im Kreis" wird durch den Unicode U+32B7 repräsentiert. Für "Kilohertz" konnte ich jedoch keinen spezifischen Unicode finden, da es typischerweise als Abkürzung "kHz" dargestellt wird und keine Einzelzeichen-Representation in Unicode hat.*

Jetzt haben wir Blut geleckt:

**Ich** *Was ist mit U+3391?*

**GPT-4** *Der Unicode U+3391 steht für das Symbol "kHz", das für Kilohertz, eine Einheit der Frequenz, steht.*

**Ich** *Und warum hast du das eben nicht gefunden?*

**GPT-4** *Es tut mir leid für das Missverständnis oder die Verwirrung. Ich hätte U+3391 als das Unicode-Zeichen für Kilohertz (kHz) in meiner früheren Antwort einbeziehen sollen. Danke, dass du darauf hingewiesen hast.*

Das war kein Missverständnis, sondern einfach nur peinliches Versagen! Beschwichtigungen wie „Danke, dass du darauf hingewiesen hast.“ mögen zwar hilfreich für das Ego des Menschen sein; sehr viel zielführender wäre es aber, wenn die KI solche neuen Wahrheiten tatsächlich sofort in ihre Wissensbasis aufnehmen könnte. Natürlich ist solch ein „automatisches Nachtrainieren bei offensichtlichen Fehlern“ technisch nur sehr schwer realisierbar,

wird aber in Zukunft immer wichtiger. Wir selbst denkenden Menschen sind einfach nicht bereit, Vertrauen zu notorischen, beratungsresistenten Lügner zu fassen, die ihren Irrtum zwar aktuell halbherzig eingestehen, uns in der nächsten Diskussion (Session) aber wieder genau die gleichen leicht widerlegbaren Lügen<sup>4</sup> auftischen.

### 2.2.3 Objektorientierte Programmierung

Bei der Erzeugung einer Polynomklasse sollen die Elemente eines Koeffizientenvektors den korrekten x-Potenzen zugeordnet werden. Es dauert vier lange, nervige Diskussionen, bis die KI bereit ist, das erste Element des Koeffizientenvektors korrekt der höchsten x-Potenz zuzuordnen. Selbst dann gibt sie als erstes das absolute Glied aus. Warum ist diese Aufgabe eigentlich so schwer für eine KI, wo doch jedes Mathe-Lehrbuch und jede Programmiersprache, die mit Polynomen arbeitet, ohne mit der Wimper zu zucken,

$$[4, -3, 6]$$

als

$$4x^2 - 3x + 6$$

interpretiert?

Beim Ersetzen führender Einsen, die von einem Buchstaben oder Pluszeichen gefolgt werden, nutzt die KI reguläre Ausdrücke

```
zeichenkette = regexprep(zeichenkette, '(^|\s|\+ )1x', '$1x');
```

die von uns Normalsterblichen erfahrungsgemäß nicht in endlicher Zeit verstanden werden können.

Die matlab-spezifische Eigenart, dass wir Klassenmethoden sowohl in der Form

```
auswerten(p, 0)
```

als auch in der Form

```
p.auswerten(0)
```

verwenden können, glaubt sie uns erst, nachdem wir hartnäckig darauf beharren. Sie kann es dann allerdings nicht lassen, noch einmal nachzutreten<sup>5</sup>: „Das bedeutet, dass sowohl `p.auswerten(0)` als auch `auswerten(p, 0)` gültig sind, aber der erste Stil ist der, der üblicherweise in der objektorientierten Programmierung verwendet und empfohlen wird, da er die Zugehörigkeit der Methode zum Objekt klarer macht.“

### 2.2.4 Bildverarbeitung

Auch die Tatsache, dass Matlab (seit 2014) als Default-Colormap Parula verwendet, entlocken wir der KI erst durch mehrfache Nachfrage. Immerhin verwendet sie diese Tatsache dann auch in weiteren Antworten (natürlich nur innerhalb der gleichen Session!).

<sup>4</sup>Aktuell finden häufig Fernsehdiskussionen mit Faschisten statt, deren Argumentation genau diesem KI-Verhalten entspricht.

<sup>5</sup>Und ja, sie hat ja Recht ...

## 2.2.5 Uhren

Bei der Extraktion der Stunden, Minuten und Sekunden der aktuellen Uhrzeit verwendet die KI ohne erkennbaren Grund manchmal die etwas peinliche<sup>6</sup> Art

```
% Aktuelle Uhrzeit abrufen
dt = datetime('now', 'Format', 'HH:mm:ss');
zeitString = char(dt);

% Stunde, Minute und Sekunde extrahieren
stunde = zeitString(1:2);
minute = zeitString(4:5);
sekunde = zeitString(7:9);
```

und manchmal die signifikant intelligentere und sicherere Art

```
% Keine Parameter übergeben; aktuelle Zeit verwenden
dt = datetime('now');
stunde = dt.Hour;
minute = dt.Minute;
sekunde = round(dt.Second); % Runden der Sekunden
```

Wir hatten im Vorfeld schon erfolglos im Internet nach der exakten Frequenz des „Pieps“ der Originalzeitansage gesucht und hatten die vage Hoffnung, dass die KI, die ja nicht nur die frei zugänglichen Quellen im Internet, sondern teilweise auch Veröffentlichungen hinter einer Bezahlschranke und Bücher „gelesen“ hat, die Frequenz für uns finden könnte – leider wurden wir auch dabei enttäuscht.

### 2.2.5.1 Spannend: Als wir heute (25.5.2024) die gleiche Frage nochmals stellen:

**Ich** *Wir suchen die Frequenz des Pieps der damaligen deutschen Zeitansage.*

ist die KI plötzlich erfreulicherweise erfolgreich:

**GPT-4** *Die Frequenz des Pieptons in der damaligen deutschen Zeitansage (Zeitansage der Deutschen Bundespost) betrug 800 Hz. Diese Zeitansage war bekannt für ihren charakteristischen Piepton, der jeweils die volle Sekunde markierte.*

Das Googeln der gleichen Frage führt zu keinem Ergebnis; ein weiteres Indiz dafür, dass Google berechtigterweise Angst vor der KI-Nutzung statt der klassischen Suchmaschine hat. Wir können die KI sogar auffordern, uns mal schnell einen 800 Hz-Ton zu generieren:

**Ich** *Erzeuge einen 800 Hz Ton.*

Die KI schreibt das passende Python-Programm

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.io.wavfile import write

# Parameters
sample_rate = 44100 # Sample rate in Hz
duration = 2 # Duration in seconds
```

<sup>6</sup>Diese unsichere Art der Sekundenextraktion führt natürlich prompt zu einer Fehlermeldung, da es das neunte Zeichen in der Zeit-Zeichenkette ja überhaupt nicht gibt.

```

frequency = 800 # Frequency of the tone in Hz

# Generate the tone
t = np.linspace(0, duration, int(sample_rate * duration),
               endpoint=False)
audio_signal = 0.5 * np.sin(2 * np.pi * frequency * t)

# Save the tone as a WAV file
output_file = '/mnt/data/800Hz_tone.wav'
write(output_file, sample_rate, audio_signal.astype(np.float32))

output_file

```

führt es in ihrer Sandkiste aus und stellt uns sogar die entsprechende Sound-Datei zur Verfügung

**GPT-4** *Hier ist der erzeugte 800 Hz Ton: [800Hz\_tone.wav herunterladen] (sandbox:/mnt/data/800Hz\_tone.wav).*

sodass wir – nach Anklicken derselben – sofort den Ton hören. Komfortabler geht es nun wirklich nicht, oder?

## 2.2.6 Pong

Als es um das Abprallen des Balls vom Spieler (Paddel) geht, erkennt die KI nicht, dass mit dem vorliegenden Code der Ball das Paddel durchdringt und erst an der Wand statt an der korrekten (innenliegenden) Seite des Paddels abprallt.

An anderer Stelle verwendet sie den `deal`-Befehl

```
[n_zeilen_spiel_feld, n_spalten_spiel_feld] = deal(breite, breite);
```

statt beide Ausgangsgrößen einzeln zu füllen:

```
n_zeilen_spiel_feld = breite;
n_spalten_spiel_feld = breite;
```

Kann man machen, aber wir würden hier einfach mal postulieren, dass 87,43% aller Matlab-Nutzerinnen den `deal`-Befehl nicht kennt.

## 2.2.7 Sortierung

Bei der Brute-Force-Methode geht es um die Frage, warum die Quersumme der Anzahl der Permutationen durch drei teilbar ist. Die ziemlich einfache, korrekte Antwort lautet natürlich, dass bei jeder Fakultät (einer Zahl größer zwei) immer eine drei als Faktor enthalten ist und eine Zahl durch drei teilbar ist, wenn ihre Quersumme durch drei teilbar ist. Aus völlig unverständlichen Gründen schreibt die KI hier aber ziemlich Blödsinn:

**GPT-4** *Warum ist die Quersumme von  $n$ -permutationen durch drei teilbar?*

*Die Tatsache, dass die Quersumme von  $n$ -permutationen durch drei teilbar ist, hat eine interessante mathematische Grundlage. Für kleine Werte von  $n$  mag dies nicht immer offensichtlich sein, aber bei größeren Werten von  $n$  wird deutlich, dass die Quersumme von  $n!$  häufig ein Vielfaches von 3 ist. Dies liegt daran, dass mit steigendem  $n$  die Faktoren von  $n!$  zunehmend Zahlen enthalten, die selbst Vielfache*

*von 3 sind (jede dritte Zahl ist ein Vielfaches von 3), was zu einer Erhöhung der Quersumme führt, die wiederum häufiger ein Vielfaches von 3 ist.*

Auch nach mehreren Nachfragen und Hilfestellungen versteht sie nicht, dass die Dreiteilbarkeit nicht **häufiger**, sondern **immer** der Fall ist.

## 2.2.8 Datenverarbeitung

Bei der Darstellung einer Matlab-Tabelle im Tortendiagramm patzt die KI an zwei Stellen: Sie schreibt:

**GPT-4** *In diesem speziellen Fall, wo der Wert 0.0 ist, wird kein Segment für Äpfel angezeigt.*

In Wahrheit werden auch Segmente mit einem Wert von null mit ihren Rändern dargestellt.

Außerdem schreibt sie:

**GPT-4**

*Wenn Sie möchten, dass auch Werte, die 0 sind, im Diagramm dargestellt werden, könnten Sie einen kleinen positiven Wert zu allen Werten hinzufügen oder spezifisch zum Wert der Äpfel, um sicherzustellen, dass auch sie im Diagramm erscheinen:*

```
T.x_adj = T.x + 0.1; % Fügt jedem Wert ein wenig hinzu, um
    sicherzustellen, dass alle sichtbar sind
pie3(T.x_adj, ones(size(T.x_adj)), T.Name)
```

*ones(size(T.x))*: Erzeugt ein Array von Einsen der gleichen Größe wie *T.x*, was bedeutet, dass keine automatische Skalierung der Segmente basierend auf den *x*-Werten stattfindet.

Das klappt so leider nicht: Das Array von Einsen bewirkt einfach nur, dass die Sektoren (radial) explodiert dargestellt werden.

## 2.2.9 Messdatenerfassung

Im Kapitel Messdatenerfassung probieren wir, 18 Einzel-Fragen auf einen Rutsch an die Studierenden und damit auch an die KI zu stellen. Sie beantwortet enttäuschenderweise ungefähr die Hälfte. Wir lernen daraus, dass es leider absolut nicht funktioniert, zu viele Frage auf einmal zu stellen, sondern dass wir möglichst jede Frage einzeln<sup>7</sup> formulieren sollten.

<sup>7</sup>Eigentlich ist es ja auch leicht nachvollziehbar, dass wir die aktuelle GPT-4-Beschränkung auf „etwa 50 Fragen in drei Stunden“ nicht einfach dadurch aushebeln können, dass wir viele Fragen auf einmal stellen.

# Literaturverzeichnis

- [1] J. J. Buchholz. (2023, Jun.) Kann sie auch Mathe? Hochschule Bremen. [Online]. Available: <https://m-server.fk5.hs-bremen.de/chatgpt/chatgpt.pdf>
- [2] ——. (2024, Jan.) Mathe - die zwote ... [Online]. Available: [https://m-server.fk5.hs-bremen.de/chatgpt/math\\_2.pdf](https://m-server.fk5.hs-bremen.de/chatgpt/math_2.pdf)
- [3] ——. (2024, Mar.) Sie regelt das schon ... Hochschule Bremen. [Online]. Available: <https://m-server.fk5.hs-bremen.de/chatgpt/rtfr.pdf>
- [4] OpenAI. (2024) GPT-4. [Online]. Available: <https://openai.com/gpt-4>
- [5] J. J. Buchholz. (2024) CAT - Computer Aided Teaching. Hochschule Bremen. [Online]. Available: <https://m-server.fk5.hs-bremen.de/cat/login.aspx>
- [6] OpenAI. (2024) Meine gpts. [Online]. Available: <https://chat.openai.com/gpts>
- [7] ——. (2024) DALL·E 3. [Online]. Available: <https://openai.com/dall-e-3>
- [8] F. Pérez, B. Granger *et al.* (2024) Project Jupyter. [Online]. Available: <https://jupyter.org/>
- [9] P. Wang *et al.* (2024) Anaconda. [Online]. Available: <https://www.anaconda.com/>

# Anhang A

## Kommunikation mit GPT-4

Wie schon bei der Untersuchung der MATH2- und RTFR-Aufgaben [2], [3] verwenden wir zur Darstellung der GPT-4-Antworten wieder JupyterLab [8] unter Anaconda [9], das sowohl Markdown interpretiert und hübsch formatiert als auch LaTeX ansehnlich kompiliert.

In der aktuellen Version versteht JupyterLab im Markdown leider weder  $\dots$  noch  $\dots$  noch  $\dots$  noch  $\dots$  noch  $\dots$  als LaTeX-Formelbegrenzer. Lediglich  $\dots$  scheint noch zu funktionieren, was aber ja Formeln zentriert absetzt.<sup>1</sup>

Leider umbricht JupyterLab Matlab-Quellcode nicht, was ja eigentlich auch durchaus sinnvoll ist, wenn wir den Code in Matlab weiterverwenden wollen. Da wir den Code aber von JupyterLab in ein PDF-Dokument exportieren lassen und dieses dann in unserem LaTeX-Dokument einbinden möchten, gehen viele Codezeilen über die vorgegebene Textbreite hinaus, was einerseits unprofessionell aussieht und andererseits Teile des Codes verschluckt. Beispiel:

Wenn `stunde` nach der Anwendung von `mod(stunde, 12)` null ist, bedeutet dies, dass die Uhrzeit entweder Mitternacht (0:00) oder Mittag (12:00) im 24-Stunden-Format ist. In beiden Fällen sollte im 12-Stunden-Format die Stunde als 12 ausgegeben werden, um Mitternacht oder Mittag korrekt zu repräsentieren. Daher sollten Sie `stunde` auf den Index des `cell array zahlen` setzen, der “zwölf” entspricht. Da wir das `cell array zahlen` mit den ausgeschriebenen Zahlen von “eins” bis “zwölf” definiert haben und eine zusätzliche “eins” am Ende hinzugefügt haben:

```
zahlen = {'eins', 'zwei', 'drei', 'vier', 'fünf', 'sechs', 'sieben', 'acht', 'neun', 'zehn',
```

Wir könnten jetzt natürlich in JupyterLab jede einzelne zu lange Zeile manuell umbrechen; aber ganz ehrlich, das ist es uns irgendwie nicht wert, wenn draußen die Sonne scheint ...

---

<sup>1</sup>Für eine Rückmeldung, wie wir auch Formeln im Fließtext in JupyterLab setzen können, wären wir sehr dankbar ...

# 1 Programmiersprachen

March 19, 2024

## 1 Frage

Testen Sie die Beispiele des Skripts unter

<https://www.tutorialspoint.com/codingground.htm>

in den Editorfenstern der Online Compiler/Interpreter in den Programmiersprachen Python 3, PHP, JavaScript, C, Java, Visual Basic, LOLCODE und Brainfuck. Schießen Sie zur Dokumentation jeweils einen Screenshot, den Sie später mit abgeben. Vermutlich ist es für Sie sinnvoll, sich im Coding Ground-Editor (Login rechts oben) ein Konto anlegen, um auch später wieder auf Ihre Projekte zugreifen zu können.

- Was müssten Sie im Python 3-Programm ändern, damit es in Python (2) das gleiche Ergebnis liefert?
- Was bedeutet `\n` im PHP-Programm?
- Wird JavaScript üblicherweise auf einem Server oder einem Client ausgeführt?
- Was bedeutet `void` im C-Programm?
- Wie erzeugen Sie die zeilenweise Ausgabe in Java?
- Was bedeutet `{0}` `{1}` in Visual Basic?
- Welche Funktion hat leerzeichen im LOLCODE-Programm?
- Wie erreichen Sie, dass das Ausrufezeichen im Brainfuck-Programm nicht ausgegeben wird?

### 1.1 Antwort

Um deine Anfragen zu bearbeiten, hier sind die Antworten:

#### 1. Python 3 zu Python 2 Änderungen:

- Ändern `print()` Funktion zu `print` Statement. In Python 3, `print()` ist eine Funktion, die Klammern benötigt, während in Python 2 `print` ein Statement ist, das keine Klammern benötigt.
- Überprüfe, ob Divisionen betroffen sind. In Python 3 ist `/` eine echte Division (auch für Ganzzahlen), während `//` eine Ganzzahldivision ist. In Python 2 ist `/` eine Ganzzahldivision für Ganzzahlen.

#### 2. Bedeutung von `\n` in PHP:

- In PHP (und vielen anderen Programmiersprachen) wird `\n` als Zeilenumbruch (Newline) Zeichen verwendet. Es beginnt eine neue Zeile.

#### 3. Ausführung von JavaScript:

- JavaScript wird üblicherweise auf dem Client ausgeführt, also im Browser des Benutzers. Es kann aber auch auf dem Server laufen, zum Beispiel mit Node.js.

#### 4. Bedeutung von `void` in C:

- In C bezeichnet das Schlüsselwort `void` einen Funktionstyp ohne Rückgabewert oder einen generischen Zeigertyp.
5. **Zeilenweise Ausgabe in Java:**
    - In Java wird die zeilenweise Ausgabe oft mit `System.out.println()` erzeugt, wobei jeder Aufruf der Funktion den angegebenen Text in einer neuen Zeile ausgibt.
  6. **Bedeutung von {0} {1} in Visual Basic:**
    - In Visual Basic werden {0} und {1} in String-Formatierungsfunktionen verwendet, um Variablen oder Werte in einen String einzufügen. {0} bezieht sich auf das erste Element, {1} auf das zweite, usw.
  7. **Funktion von leerzeichen in LOLCODE:**
    - In LOLCODE könnte `leerzeichen` (wenn es analog zum Englischen `space` verwendet wird) eine Variable sein, die ein Leerzeichen-Zeichen () speichert oder repräsentiert.
  8. **Brainfuck Ausrufezeichen nicht ausgeben:**
    - Um das Ausrufezeichen im Brainfuck-Programm nicht auszugeben, könnten Sie den Teil des Codes entfernen oder modifizieren, der das Ausrufezeichen in die Ausgabe schreibt. Brainfuck operiert direkt auf dem Speicherband und der Ausgabe, also würde dies bedeuten, den Befehl zu entfernen oder zu ändern, der den Speicherzeiger zu dem Wert des Ausrufezeichens bewegt und dann ausgibt.

Da ich kein Browser-Interface habe, kann ich die Beispiele auf der Website nicht direkt ausführen oder Screenshots davon machen. Ich empfehle dir, die Anweisungen zu folgen und die Skripte in den Online-Compiler/Interpreter von Coding Ground einzugeben. Wenn du fertig bist, kannst du die Screenshots selbst aufnehmen.

## 2 Frage

Testen Sie das HTML-LATEX-Beispiel aus dem Skript im Online-HTML-Editor des Coding Ground. Denken Sie auch hier daran, die gesamte MathJax-Adresse in eine Zeile zu schreiben.

- Wozu wird MathJax aufgerufen?
- Was bewirkt `$$` normalerweise in einem LATEX-Programm?
- Warum ist sein Einsatz hier nicht notwendig?
- Warum verwenden Sie eine `align`-Umgebung?
- Was bewirkt `\\`?
- Warum ist `1=2`?

### 2.1 Antwort

MathJax wird verwendet, um mathematische Ausdrücke innerhalb einer Webseite darzustellen. Es ist besonders hilfreich für die Darstellung von komplexen mathematischen Formeln in HTML, ohne auf Bilder oder spezielle Schriftarten zurückgreifen zu müssen. MathJax interpretiert mathematische Markup-Sprachen wie LaTeX, MathML und AsciiMath und stellt sie als hochwertige typografische Ausgaben direkt im Browser dar.

In einem LaTeX-Programm wird `$$` verwendet, um mathematische Ausdrücke in einem abgesetzten (display) Modus zu setzen, was bedeutet, dass die Ausdrücke zentriert und auf einer eigenen Zeile dargestellt werden. Im Gegensatz zum Inline-Modus, wo mathematische Inhalte innerhalb eines Textabsatzes erscheinen, bietet der Display-Modus mehr Platz und macht komplexe Formeln übersichtlicher.

Im HTML-Beispiel, das Sie aus dem Skript testen sollten, ist die Verwendung von  $\$$  nicht notwendig, da MathJax und die umgebende align-Umgebung bereits dafür sorgen, dass die mathematischen Ausdrücke im Display-Modus und entsprechend formatiert angezeigt werden.

Die align-Umgebung wird in LaTeX und hier in Verbindung mit MathJax verwendet, um mathematische Gleichungen untereinander auszurichten, typischerweise am Gleichheitszeichen oder einem anderen Operator. Dies verbessert die Lesbarkeit von aufeinanderfolgenden Gleichungen oder mathematischen Ausdrücken.

In mathematischen Formeln wird der doppelte Rückstrich  $\backslash\backslash$  verwendet, um einen Zeilenumbruch einzufügen. In der align-Umgebung ermöglicht er die Trennung verschiedener Gleichungen oder Ausdrücke, die untereinander ausgerichtet werden sollen.

Die Aussage  $1=2$  ist natürlich mathematisch falsch. Sie wird in didaktischen Beispielen oft als absichtlich falsches Ergebnis dargestellt, um Fehler in der Argumentation oder die Auswirkungen bestimmter mathematischer Manipulationen (wie Division durch Null) zu illustrieren. In Ihrem Skript dient dies möglicherweise als Beispiel oder als Teil einer mathematischen Paradoxie oder Fehlschluss, um zu diskutieren, warum solche "Beweise" nicht gültig sind.

## 2.2 Frage

Analysiere das konkrete  $1=2$ -Beispiel aus dem Skript.

### 2.2.1 Antwort

Das Beispiel aus dem Skript führt Schritt für Schritt zu der inkorrekten Schlussfolgerung, dass 1 gleich 2 sei. Lassen Sie uns die Gleichungen analysieren:

1.

$$a^2 - a^2 = a^2 - a^2$$

Dies ist offensichtlich wahr, da beide Seiten der Gleichung identisch sind.

2.

$$a \cdot (a - a) = (a + a) \cdot (a - a)$$

Hier wurde die linke Seite als

$$a^2 - a^2$$

und die rechte Seite als

$$2a^2 - 2a^2$$

umgeschrieben, was immer noch korrekt ist, da beide Seiten nach der Ausmultiplizierung zu Null werden.

3.

$$a = a + a$$

An diesem Punkt wird ein Fehlschluss begangen: Hier scheint durch

$$a - a$$

(also Null) geteilt worden zu sein, um von der vorherigen Zeile zu dieser zu gelangen. Da die Division durch Null nicht definiert ist, ist dieser Schritt ungültig.

4.

$$1 \cdot a = 2 \cdot a$$

Dies folgt aus der falschen Annahme, dass

$$a$$

gleich

$$a + a$$

ist, ohne zu berücksichtigen, dass dieser Schritt aus einer Division durch Null resultierte.

5.

$$1 = 2$$

Wenn man von beiden Seiten der Gleichung

$$a$$

(unter der Annahme, dass

$$a$$

nicht Null ist) kürzt, würde man zu diesem Ergebnis kommen. Aber da der Schritt, der zu

$$a = a + a$$

führte, bereits falsch war, ist auch diese Schlussfolgerung falsch.

Der kritische Fehler im Beweis ist der Schritt von der zweiten zur dritten Zeile, bei dem durch

$$a - a$$

(also Null) geteilt wird. Dieser Schritt ist mathematisch nicht zulässig, da eine Division durch Null nicht definiert ist. Daher ist die Aussage

$$1 = 2$$

falsch und das Ergebnis eines fehlerhaften mathematischen Prozesses.

*Kleiner Lapsus bei "... die rechte Seite als*

$$2a^2 - 2a^2$$

*umgeschrieben ...". Trotzdem das Problem richtig erkannt.*

## 2 Linux

April 25, 2024

### 1 Frage

Registrieren Sie sich unter

<https://codeanywhere.com>

Erzeugen Sie dort einen neuen Container (beispielsweise vom Typ Python). Nennen Sie den Container `mein_container`.

Führen Sie die Experimente aus

<https://m-server.fk5.hs-bremen.de/info/ct.18.05.142-145.pdf>

in der bash aus.

Damit die installierten “Spiele” gefunden werden, fügen Sie das Verzeichnis, in dem sie liegen, zur Umgebungsvariablen `$PATH` hinzu:

```
PATH="$PATH:/usr/games" export PATH
```

Dokumentieren Sie Ihre Arbeit durch Screenshots.

- Mit welchem Befehl installieren Sie den nano-Editor?
- Welche Editor-Alternative haben Sie in der Codeanywhere-Umgebung?

Wenn die Seite von Pit Noack nicht erreichbar ist, können Sie das Gedicht auch unter <http://m-server.fk5.hs-bremen.de/info/ameisen.txt> herunterladen.

#### 1.1 Antwort

Um einen neuen Container in Codeanywhere zu erstellen, folgen Sie bitte diesen Schritten:

1. Navigieren Sie auf der Startseite oder auf der Seite “Containers” im Dashboard von Codeanywhere.
2. Klicken Sie auf den Button “New Container”.
3. Wählen Sie eine Vorlage für Ihren Container aus, z.B. Python, wenn Sie Python-basierte Experimente durchführen möchten.
4. Optional können Sie das “Always-On”-Addon anwenden, wenn Sie möchten.
5. Geben Sie einen Namen für Ihren neuen Container ein, zum Beispiel “`mein_container`”, wie Sie vorgeschlagen haben .

Sobald Ihr Container eingerichtet ist, können Sie die integrierte Terminalkonsole in Codeanywhere verwenden, um Befehle direkt auf Ihrem Container oder Remote-Servern auszuführen. Diese Funktion ermöglicht einfaches Experimentieren und Entwickeln in einer Cloud-basierten Umgebung.

Für die Installation des Nano-Editors würden Sie typischerweise den folgenden Befehl im Terminal Ihres Containers verwenden:

```
sudo apt-get install nano
```

Als Alternative zum eingebauten Editor innerhalb der Codeanywhere-Umgebung bietet Codeanywhere einen leistungsstarken integrierten Code-Editor, der Syntaxhervorhebung für mehrere Programmiersprachen unterstützt und verschiedene Produktivitätsfunktionen wie Code-Vervollständigung und Linting bietet [14](#)source .

Bezüglich der Experimente, die Sie erwähnten: Während ich nicht direkt mit Codeanywhere interagieren oder den Prozess dokumentieren kann, können Sie sich auf die detaillierten Einrichtungsinstruktionen für eine Umgebung wie einen Blank Ubuntu 14.04 Container beziehen, die in der Codeanywhere-Dokumentation und externen Leitfäden bereitgestellt werden [13](#)source [14](#)source .

Nachdem Sie Ihren Container eingerichtet haben, denken Sie daran, den Befehl auszuführen, um “/usr/games” Ihrer \$PATH-Umgebungsvariable hinzuzufügen, wie folgt:

```
PATH="$PATH:/usr/games"  
export PATH
```

Dies ermöglicht es Ihnen, die installierten “Spiele” oder andere ausführbare Skripte von jedem Standort innerhalb Ihres Containers aus aufzurufen.

Wenn Sie Fragen haben oder weitere Unterstützung benötigen, sind die Dokumentations- und Support-Seiten von Codeanywhere großartige Ressourcen [14](#)source .

# 3 Zahlendarstellung

March 19, 2024

## 1 Frage

Sie möchten im Folgenden ein Unterprogramm `b2d` erstellen, das eine Binärzahl `b` in eine Dezimalzahl `d` umrechnet. Die Binärzahl `b` wird dabei als Zeichenkette übergeben. Bestimmen Sie im Unterprogramm als erstes die Länge von `b` und initialisieren Sie die Dezimalzahl `d` mit null.

Warum sollten Sie die Binärzahl als Zeichenkette übergeben?

### 1.1 Antwort

Die Übergabe der Binärzahl als Zeichenkette (String) in einem Unterprogramm zur Umwandlung in eine Dezimalzahl hat mehrere Gründe:

1. **Flexibilität:** Zeichenketten können Binärzahlen beliebiger Länge darstellen, was bei der Verwendung von numerischen Datentypen (wie Integer) nicht immer der Fall ist, da diese typischerweise auf eine maximale Anzahl von Bits beschränkt sind.
2. **Vermeidung von Fehlinterpretationen:** Direkte Übergabe einer Binärzahl in einem numerischen Format könnte zu Verwirrungen führen, da numerische Werte standardmäßig als Dezimalzahlen interpretiert werden. Ein String macht deutlich, dass es sich um eine Folge von Binärzeichen handelt.
3. **Einfachere Manipulation:** Die Operationen auf den einzelnen Zeichen einer Zeichenkette (zum Beispiel das Auslesen der  $n$ -ten Position) sind oft einfacher zu handhaben als bitweise Operationen auf numerischen Typen.

Basierend auf diesen Gründen können wir ein Matlab-Unterprogramm `b2d` schreiben:

```
function d = b2d(b)
    % Initialisierung der Dezimalzahl
    d = 0;
    % Bestimmung der Länge der Binärzahl
    n = length(b);
    % Umrechnung der Binär- in eine Dezimalzahl
    for i = 1:n
        % Umrechnung des i-ten Zeichens in eine Zahl und
        % Multiplikation mit der entsprechenden 2er Potenz
        d = d + (b(i) - '0') * 2^(n-i);
    end
end
```

In diesem Programm ist `b` die Eingabezeichenkette, die die Binärzahl repräsentiert, und `d` ist die Ausgabe, die dezimale Darstellung der Binärzahl. Wir subtrahieren '0' von jedem Zeichen, um den entsprechenden numerischen Wert zu erhalten (da in ASCII die Ziffern 0-9 aufeinanderfolgende Codes haben). Die Potenzierung mit  $2^{(n-i)}$  erfolgt, weil das linke Zeichen die höchste Wertigkeit (größte Potenz von 2) hat.

*Das Schreiben des Programmes geht gerade ein bisschen schnell. "Im Folgenden" heißt ja, im Rahmen der folgenden Aufgaben ...*

## 2 Frage

In einer Schleife über alle Zeichen der Binärzahl wandeln Sie das jeweilige Zeichen mit `str2double` in eine Fließkommazahl um, multiplizieren es mit seiner Zweierpotenz (`pow2`) und addieren es dann zur aktuellen Dezimalzahl. Was würde passieren, wenn Sie auf `str2double` verzichten würden? Welche Alternative hätten Sie zum `pow2`-Befehl gehabt?

### 2.1 Antwort

Wenn Sie auf `str2double` verzichten würden, würden Sie direkt mit dem ASCII-Wert des Zeichens arbeiten, anstatt mit dem numerischen Wert, den das Zeichen repräsentiert. In ASCII entspricht das Zeichen '0' zum Beispiel dem Wert 48 und '1' dem Wert 49. Würden Sie also die ASCII-Werte direkt verwenden, ohne `str2double` zur Konvertierung in die entsprechenden Zahlenwerte, würden Sie bei der Berechnung falsche Ergebnisse erhalten, weil Sie eigentlich mit 48 oder 49 multiplizieren würden, anstatt mit 0 oder 1.

Eine Alternative zu `str2double` wäre die Verwendung der Tatsache, dass die Differenz zwischen dem ASCII-Wert eines numerischen Zeichens und dem ASCII-Wert von '0' dem tatsächlichen numerischen Wert dieses Zeichens entspricht. Sie könnten also das Zeichen direkt in eine Zahl umwandeln, indem Sie '0' davon abziehen, was in Matlab und vielen anderen Programmiersprachen einer direkten Umwandlung des Zeichenwerts in seine numerische Form entspricht, zum Beispiel durch `(b(i) - '0')`.

Als Alternative zum `pow2`-Befehl könnten Sie die eingebaute Funktion  $2^x$  verwenden, wobei `x` der Exponent ist. Während `pow2` speziell für Operationen mit der Basis 2 konzipiert ist, ist  $2^x$  eine allgemeinere Form, die für jede Basis und jeden Exponenten verwendet werden kann. In Ihrem Kontext, wo die Basis immer 2 ist (da es sich um eine Binärzahl handelt), würden beide Funktionen ähnliche Ergebnisse liefern. Doch  $2^{(n-i)}$  wird oft in Umrechnungen von Binär- zu Dezimalzahlen verwendet, da es direkt den Wert jeder Binärstelle (0 oder 1) in ihrem entsprechenden dezimalen Wert umrechnet, basierend auf ihrer Position im Binärstring.

*Passt!*

### 2.2 Frage

Wie lautet dann also jetzt das Programm mit `str2double` und `pow2`?

#### 2.2.1 Antwort

Das Programm unter Verwendung von `str2double` und `pow2` zur Umwandlung einer Binärzahl (als Zeichenkette) in eine Dezimalzahl in Matlab könnte wie folgt aussehen:

```

function d = b2d(b)
    % Initialisierung der Dezimalzahl
    d = 0;
    % Bestimmung der Länge der Binärzahl
    n = length(b);
    % Umrechnung der Binär- in eine Dezimalzahl
    for i = 1:n
        % Umwandlung des i-ten Zeichens in eine Fließkommazahl
        digit = str2double(b(i));
        % Multiplikation mit der entsprechenden Zweierpotenz und
        % Addition zur aktuellen Dezimalzahl
        d = d + digit * pow2(n-i);
    end
end

```

In diesem Programm wird `str2double(b(i))` verwendet, um jedes Zeichen der Eingabezeichenkette `b` von einem String in eine Fließkommazahl (0 oder 1) umzuwandeln. `pow2(n-i)` wird dann verwendet, um den entsprechenden Wert jeder Binärstelle basierend auf ihrer Position zu berechnen: Für das höchstwertige Bit ist der Exponent am größten ( $n-1$ ) und für das niederwertigste Bit ist der Exponent 0. Die Funktion `pow2(x)` in Matlab berechnet effektiv  $2^x$ , was perfekt für die Umwandlung von Binär in Dezimal geeignet ist, da jede Stelle in einer Binärzahl einem Potenzwert von 2 entspricht.

*Läuft*

### 3 Frage

Wenden Sie `b2d` auf folgende Binärzahlen an:

0

1

10

00000001

11111111

10101101

Wie interpretiert Ihr `b2d` führende Nullen? Warum ist das so?

Was passiert, wenn Sie `b2d` die Zeichenkette '12' übergeben? Warum?

Was geschieht bei 'ab'? Warum?

#### 3.1 Antwort

Wenn wir die `b2d`-Funktion, die ich zuvor beschrieben habe, auf verschiedene Binärstrings anwenden, hier sind die erwarteten Ergebnisse und Interpretationen:

1. `b2d('0')` wird 0 ergeben, weil es keine Zweierpotenzen gibt, die addiert werden müssen – das einzige Ziffer im String repräsentiert den Wert 0.

2. `b2d('1')` wird 1 ergeben, da die einzige Ziffer im String, die eine 1 ist, der Zweierpotenz von 0 entspricht, also  $2^0 = 1$ .
3. `b2d('10')` wird 2 ergeben, da die führende 1 in der Binärzahl der Zweierpotenz von 1 entspricht, also  $2^1 = 2$  (und die nachfolgende Null trägt keinen Wert bei).
4. `b2d('00000001')` wird ebenfalls 1 ergeben. Führende Nullen beeinflussen den Wert einer Binärzahl nicht, da sie keinen Beitrag zum Gesamtwert leisten, genau wie führende Nullen in einer Dezimalzahl den Wert nicht ändern.
5. `b2d('11111111')` wird 255 ergeben, da jede Ziffer zur Summe beiträgt, beginnend mit  $2^7$  (128) bis hinunter zu  $2^0$  (1).
6. `b2d('10101101')` wird 173 ergeben, basierend auf der Summe der Zweierpotenzen für jede Stelle, die eine '1' enthält.

Führende Nullen werden von der `b2d` Funktion ignoriert, weil sie den numerischen Wert der Binärzahl nicht ändern. Sie dienen lediglich dazu, den Platz zu halten, ähnlich wie führende Nullen in Dezimalzahlen.

Wenn wir `b2d` die Zeichenkette '12' übergeben, würden wir einen Fehler oder ein unerwartetes Ergebnis bekommen, denn '2' ist kein gültiges Zeichen in einem Binärstring. Die Funktion `str2double` wird versuchen, '2' in eine Fließkommazahl umzuwandeln und scheitern oder es wird in eine NaN (nicht eine Zahl) umgewandelt, was in der Berechnung nicht verwendbar ist.

Wenn wir `b2d` den String 'ab' übergeben, würde dies ebenfalls zu einem Fehler oder einem unerwarteten Ergebnis führen, da 'a' und 'b' keine gültigen Binärzeichen sind. Die `str2double`-Funktion würde NaN für beide Zeichen zurückgeben, was wiederum die Berechnung ungültig macht.

In echten Anwendungsszenarien sollte die Eingabe für solche Funktionen immer validiert werden, um sicherzustellen, dass sie nur gültige Zeichen (in diesem Fall '0' oder '1') enthält, bevor die eigentliche Umrechnung stattfindet.

*Nicht ganz:*

## 3.2 Frage

Du sagst: Die Funktion `str2double` wird versuchen, '2' in eine Fließkommazahl umzuwandeln und scheitern

Warum sollte das nicht klappen?

### 3.2.1 Antwort

Entschuldigung für das Missverständnis. Tatsächlich würde `str2double` in Matlab beim Umwandeln eines gültigen numerischen Strings wie '2' in eine Fließkommazahl nicht scheitern. Es würde den Wert 2 zurückgeben, da '2' eine gültig repräsentierte Zahl als String ist.

Mein Fehler lag in der Beschreibung des erwarteten Verhaltens in Bezug auf Ihre spezifische Binär- zu Dezimalumwandlungsfunktion. In diesem Kontext ist '2' kein gültiges Zeichen für einen Binärstring, der nur aus den Ziffern '0' und '1' bestehen sollte. Jedoch, da die Funktion `b2d` auf der Annahme basiert, dass die Eingabe ein Binärstring ist, sollte sie eigentlich auf Eingaben, die außerhalb dieser Definition liegen (wie die '2'), überprüfen und möglicherweise eine Fehlermeldung ausgeben oder das Verhalten spezifizieren.

Jedoch, in der aktuellen Implementierung, wenn `b2d('12')` ausgeführt wird, wird die `str2double` Funktion die '1' und '2' korrekt in Fließkommazahlen umwandeln, und die Funktion berechnet auf dieser Basis das Ergebnis, was in diesem Fall nicht den erwarteten Binär-Dezimal-Konvertierungsregeln entspricht. Tatsächlich würde die Funktion dann arbeiten, als ob sie gültige Binärdaten verarbeiten würde, was sie offensichtlich nicht sind.

Also, korrekterweise, `str2double('2')` würde ohne Fehler arbeiten und 2 zurückgeben, aber in der Logik einer Binär-zu-Dezimal-Umwandlung ist das Vorhandensein einer '2' fehlerhaft.

*So ist besser.*

### 3.3 Frage

Was würde das Programm also bei `b2d('12')` zurückgeben?

#### 3.3.1 Antwort

Entschuldigung für das Missverständnis. Tatsächlich würde `str2double` nicht scheitern, '2' in eine Fließkommazahl umzuwandeln. Ich habe einen Fehler gemacht. Die Funktion `str2double` kann durchaus den String '2' in die numerische Fließkommazahl 2 umwandeln. Das Problem bei der Eingabe '12' für eine Funktion, die eine Binärzahl in eine Dezimalzahl umwandelt, ist nicht, dass `str2double` nicht konvertieren kann, sondern dass '2' kein gültiges Zeichen in einem Binärstring ist.

Wenn Sie `b2d('12')` ausführen würden, würde die Funktion dennoch für jedes Zeichen laufen:

- Für '1' würde `str2double('1')` erfolgreich 1 zurückgeben.
- Für '2' würde `str2double('2')` erfolgreich 2 zurückgeben.

Da aber eine Binärzahl nur die Ziffern 0 und 1 beinhalten sollte, ist der Eingabestring '12' kein gültiger Binärstring. In einem strikt binären Kontext hat '2' jedoch keine Bedeutung und sollte daher normalerweise zu einem Fehler oder einer Warnung führen, weil der Eingabestring nicht gültig ist. Aber in unserem Fall, wo wir nur `str2double` und `pow2` verwenden und nicht überprüfen, ob die Eingabe streng binär ist, würde das Ergebnis mathematisch auf der Basis der umgewandelten Werte berechnet werden, was nicht der erwarteten Funktion einer Binär-zu-Dezimal-Konvertierung entspricht.

Daher, unter der Annahme, dass die Länge des Strings '12' als 2 angenommen wird und der Index bei 1 beginnt (der höchsten Potenz), würde der Wert theoretisch als

$$1 * 2^1 + 2 * 2^0$$

berechnet, was 4 ergibt, was offensichtlich nicht dem entspricht, was wir von einer Funktion erwarten würden, die Binärzahlen in Dezimalzahlen umwandelt, da '2' kein valider Binärwert ist.

Es ist wichtig zu beachten, dass dies nicht dem erwarteten Gebrauch der Funktion entspricht, da '12' keine valide Binärzahl darstellt. In einer präzisen Implementierung sollten ungültige Binäreingaben wie '2' innerhalb einer Binärzahl eigentlich einen Fehler auslösen oder zumindest ignoriert werden, wenn sie nicht in einem strikten Modus verarbeitet werden.

*Genau!*

### 3.4 Frage

Und was würde das Programm bei `b2d('ab')` zurückgeben?

#### 3.4.1 Antwort

Wenn die `b2d`-Funktion mit `'ab'` aufgerufen wird, würde `str2double` versuchen, diese Zeichen in Fließkommazahlen umzuwandeln. Da `'a'` und `'b'` jedoch keine gültigen Zahlen sind, würde `str2double` für diese Zeichen `NaN` (Not a Number) zurückgeben.

In der `b2d`-Funktion würden diese `NaN`-Werte dann in den Berechnungen verwendet. Wenn man eine Zahl mit `NaN` multipliziert oder zu `NaN` addiert, ist das Ergebnis immer `NaN`. Das bedeutet, dass, wenn die Eingabe `'ab'` ist, das Ergebnis der Funktion `NaN` wäre, weil die ungültigen Zeichen in `NaN` umgewandelt werden, was zu einer Berechnung führt, die `NaN` ergibt, unabhängig von der spezifischen Logik für die Konvertierung von Binär zu Dezimal.

Dies illustriert, dass die `b2d`-Funktion, wie sie derzeit definiert ist, nicht für die Verarbeitung ungültiger Binärzeichenstrings ausgelegt ist. In der Praxis sollte sie mit zusätzlicher Validierung versehen werden, um sicherzustellen, dass alle Zeichen im Eingabestring gültige binäre Ziffern sind (entweder `'0'` oder `'1'`).

*Auch richtig!*

## 4 Frage

Sie möchten im Folgenden ein Unterprogramm `b2h` schreiben, das eine Binärzahl hexadezimal darstellt. Im Unterprogramm bestimmen Sie als Erstes wieder die Anzahl der Zeichen der Binärzahl (hier `n_b`). Im Folgenden wollen Sie dann jeweils vier Stellen der Binärzahl zu einem hexadezimalen Zeichen zusammenfassen. Es ist daher notwendig, die Länge der Binärzahl auf ein Vielfaches von vier aufzufüllen, indem Sie die Binärzahl links mit der entsprechenden Anzahl an Nullen aufpolstern. Um die Anzahl der notwendigen Nullen herauszufinden, verwenden Sie die Zuweisung

```
n_padding = 4 * ceil(n_b / 4) - n_b
```

Erläutern Sie ausführlich die Funktionsweise der Zuweisung.

Was bewirkt der `ceil`-Befehl?

Warum können Sie stattdessen nicht den `round`-Befehl verwenden?

#### 4.1 Antwort

Die Zuweisung `n_padding = 4 * ceil(n_b / 4) - n_b` wird verwendet, um die Anzahl der Nullen zu bestimmen, die einer Binärzahl vorangestellt werden müssen, damit ihre Gesamtlänge ein Vielfaches von vier beträgt. Dies ist notwendig, weil jede Hexadezimalziffer vier Binärziffern entspricht. Erläutern wir die einzelnen Teile dieser Zuweisung:

1. `n_b / 4`: Dieser Ausdruck berechnet, wie viele Vierergruppen in der Binärzahl vorhanden sind, was einer Division der Gesamtlänge der Binärzahl durch vier entspricht. Wenn `n_b` bereits ein Vielfaches von vier ist, wäre dieser Wert eine ganze Zahl; wenn nicht, wäre es eine Fließkommazahl.

2. `ceil(n_b / 4)`: Der `ceil`-Befehl (Ceiling-Funktion) rundet den Wert innerhalb der Klammern auf die nächste ganze Zahl auf, falls es sich nicht bereits um eine ganze Zahl handelt. Das bedeutet, wenn `n_b / 4` nicht schon eine ganze Zahl ist (d.h., die Länge der Binärzahl ist kein Vielfaches von vier), wird sie auf die nächste ganze Zahl aufgerundet. Dies gibt uns die Anzahl der vollständigen Vierergruppen, die benötigt würden, um die Binärzahl vollständig darzustellen, einschließlich der notwendigen zusätzlichen Gruppen für fehlende Binärstellen.
3. `4 * ceil(n_b / 4)`: Dies multipliziert die Anzahl der notwendigen Vierergruppen (bestimmt im vorigen Schritt) mit vier, um die neue Gesamtlänge der Binärzahl zu erhalten, die nun ein Vielfaches von vier ist.
4. `4 * ceil(n_b / 4) - n_b`: Schließlich wird die ursprüngliche Länge der Binärzahl von der neuen Länge abgezogen. Das Ergebnis ist die Anzahl der Nullen, die der Binärzahl vorangestellt werden müssen, damit ihre Länge ein Vielfaches von vier beträgt.

Der `ceil`-Befehl wird verwendet, weil wir immer auf die nächste ganze Zahl aufrunden müssen, wenn die Länge der Binärzahl kein Vielfaches von vier ist. Dies stellt sicher, dass wir genügend Nullen hinzufügen, um vollständige Hexadezimalziffern zu bilden.

Der `round`-Befehl ist hier nicht geeignet, da `round` einen Wert zur nächsten ganzen Zahl rundet, basierend auf der typischen Rundungsregel (wenn der Bruchteil kleiner als 0.5 ist, wird abgerundet; ist er 0.5 oder größer, wird aufgerundet). Wenn `n_b / 4` zum Beispiel 2.25 wäre, würde `round(n_b / 4)` dies zu 2 runden, was bedeutet, dass wir nicht genügend Nullen hinzufügen würden, um die Binärzahl auf ein Vielfaches von vier zu erweitern. Deshalb verwenden wir `ceil`, um sicherzustellen, dass wir immer auf die nächste ganze Zahl aufsteigen, was erforderlich ist, um die Länge korrekt auf ein Vielfaches von vier zu erweitern.

*Sehr schöne Erklärungen*

## 5 Frage

Führen Sie das Padding in einer `for`-Schleife durch, ermitteln Sie erneut die Länge der aufgepolsterten Binärzahl und initialisieren Sie die Hexadezimalzahl mit einer leeren Zeichenkette.

Warum müssen Sie für die Hexadezimalzahl eine Zeichenkette und können keine Dezimalzahl verwenden?

### 5.1 Antwort

Die Hexadezimaldarstellung einer Zahl ist ein Zahlensystem mit der Basis 16. Dies bedeutet, dass jede Ziffer einer Hexadezimalzahl 16 mögliche Werte annehmen kann, oft dargestellt als 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, und F, wobei die Buchstaben A bis F die Zahlen 10 bis 15 repräsentieren. Aufgrund dieser Darstellungsweise, die alphanumerische Zeichen einschließt, muss die Hexadezimalzahl als Zeichenkette und nicht als Dezimalzahl gespeichert werden. Dezimalzahlen können die Buchstaben A bis F nicht als Teile einer einzelnen Zahl darstellen, da sie auf die Werte 0 bis 9 beschränkt sind.

Hier ist, wie Sie das Padding in einer `for`-Schleife durchführen und die Länge der aufgepolsterten Binärzahl ermitteln könnten, während Sie die Hexadezimalzahl als eine leere Zeichenkette initialisieren:

```

function hex = b2h(bin)
    % Ermitteln der Länge der Binärzahl
    n_b = length(bin);
    % Berechnen der notwendigen Anzahl von führenden Nullen
    n_padding = 4 * ceil(n_b / 4) - n_b;

    % Hinzufügen der führenden Nullen zur Binärzahl
    padded_bin = bin; % Kopie der ursprünglichen Binärzahl
    for i = 1:n_padding
        padded_bin = ['0', padded_bin];
    end

    % Ermitteln der neuen Länge der aufgepolsterten Binärzahl
    new_length = length(padded_bin);

    % Initialisieren der Hexadezimalzahl als leere Zeichenkette
    hex = '';

    % Weiterer Verarbeitungscode kann hier folgen...
end

```

In diesem Code:

- `n_b` ist die Länge der ursprünglichen Binärzahl.
- `n_padding` ist die Anzahl der Nullen, die hinzugefügt werden müssen, um die Länge der Binärzahl auf ein Vielfaches von vier zu bringen.
- `padded_bin` wird mit den erforderlichen Nullen vorne hinzugefügt.
- `new_length` ist einfach die Länge der neuen Binärzahl nach dem Hinzufügen der führenden Nullen.
- `hex` ist die Hexadezimaldarstellung, die als leere Zeichenkette initialisiert wird, weil wir darin alphanumerische Zeichen (0-9 und A-F) aufnehmen werden, um die Binärzahl in ihrer Hexadezimalform darzustellen.

Indem wir die Hexadezimalzahl als Zeichenkette initialisieren, können wir sowohl Ziffern als auch Buchstaben hinzufügen, die notwendig sind, um die Binärzahl korrekt in eine Hexadezimalzahl zu übersetzen.

*Ich finds gut!*

## 6 Frage

In einer for-Schleife untersuchen Sie jeweils vier Ziffern der Binärzahl. Die Anzahl der Schleifendurchläufe ist also nur ein Viertel der Anzahl der Binärzahlziffern. In der Schleife verwenden Sie den switch-Befehl, um die möglichen Viererpakete

0000 0001 : : 1111

zu unterscheiden. In jedem der 16 Fälle (case) ergänzen Sie dann einfach die passende Hexadezimalziffer zur Hexadezimalzahl. Welche Alternative hätten Sie zur switch-Fallunterscheidung gehabt?

Der schwierige Teil dieser Aufgabe ist es, sich zu überlegen, wie Sie in der Schleife jeweils die Vierergruppen der Binärziffern ansprechen. Üblicherweise verwenden Sie dazu einen Indexbereich, der von der Laufvariable (hier  $i$ ) abhängt; also einen Indexausdruck wie etwa:  $4i - 3 : 4i$ .

## 6.1 Antwort

Verstanden, wir können eine MATLAB-Funktion erstellen, die den `switch`-Befehl verwendet, um jedes Vierergruppen-Paket einer Binärzahl in die entsprechende Hexadezimalziffer zu konvertieren. Hier ist, wie das Unterprogramm `b2h` aussehen könnte:

```
function hex = b2h(bin)
    % Bestimmen der Länge der Binärzahl
    n_b = length(bin);
    % Berechnen der notwendigen Anzahl von führenden Nullen
    n_padding = 4 * ceil(n_b / 4) - n_b;
    % Hinzufügen der führenden Nullen zur Binärzahl
    bin = [repmat('0', 1, n_padding), bin];

    % Initialisieren der Hexadezimalzahl als leere Zeichenkette
    hex = '';

    % Untersuchen der Binärzahl in Vierergruppen
    for i = 1:length(bin)/4
        % Extrahieren der aktuellen Vierergruppe
        binGroup = bin((4 * i - 3):(4 * i));

        % Umwandlung der Binärgruppe in die entsprechende Hexadezimalziffer
        switch binGroup
            case '0000'
                hex_digit = '0';
            case '0001'
                hex_digit = '1';
            case '0010'
                hex_digit = '2';
            case '0011'
                hex_digit = '3';
            case '0100'
                hex_digit = '4';
            case '0101'
                hex_digit = '5';
            case '0110'
                hex_digit = '6';
            case '0111'
                hex_digit = '7';
            case '1000'
                hex_digit = '8';
            case '1001'
                hex_digit = '9';
```

```

    case '1010'
        hex_digit = 'A';
    case '1011'
        hex_digit = 'B';
    case '1100'
        hex_digit = 'C';
    case '1101'
        hex_digit = 'D';
    case '1110'
        hex_digit = 'E';
    case '1111'
        hex_digit = 'F';
    otherwise
        hex_digit = '?'; % Für den Fall,
        % dass eine nicht-binäre Sequenz vorhanden ist
end
% Hinzufügen der ermittelten Hexadezimalziffer zur Gesamthexadezimalzahl
hex = [hex, hex_digit];
end
end

```

In diesem Code:

- `n_b` ist die Länge der ursprünglichen Binärzahl.
- `n_padding` berechnet, wie viele Nullen der Zahl hinzugefügt werden müssen, damit ihre Länge ein Vielfaches von vier beträgt.
- Die `for`-Schleife iteriert durch jede Vierergruppe der Binärzahl.
- Innerhalb der Schleife verwendet ein `switch`-Block die extrahierte Vierergruppe (`binGroup`), um die entsprechende Hexadezimalziffer zu bestimmen und an die bisherige Hexadezimalzahl `hex` anzuhängen.
- Falls die Binärgruppe nicht erkannt wird (was eigentlich nicht passieren sollte, wenn alle Eingaben gültig sind), fügt der `otherwise`-Fall ein Fragezeichen hinzu. Dies hilft bei der Fehlersuche, wenn ungültige Daten eingegeben wurden.

*Inbesondere das Hinzufügen des Fragezeichen bei ungültigen Binärgruppen ist eine gute Idee!*

## 7 Frage

Wandeln Sie wiederum die Binärzahlen

0

1

10

00000001

11111111

10101101

in ihre hexadezimale Form um.

Wie reagiert `b2h` auf führende Nullen und falsche Eingaben?

## 7.1 Antwort

Um die von Ihnen angegebenen Binärzahlen in ihre hexadezimalen Formen umzuwandeln, verwenden wir die Funktion `b2h`, die oben definiert wurde. Hier sind die erwarteten Ergebnisse der Umwandlung:

1. `b2h('0')` gibt `'0'` zurück. Führende Nullen werden hinzugefügt, um die Gesamtlänge auf ein Vielfaches von vier zu bringen. In diesem Fall ist die Binärzahl bereits auf der richtigen Länge, also wird sie direkt übersetzt.
2. `b2h('1')` gibt `'1'` zurück. Ähnlich wie oben wird die Binärzahl zu `'0001'` aufgefüllt, bevor sie übersetzt wird.
3. `b2h('10')` gibt `'2'` zurück. Die Binärzahl wird zu `'0010'` aufgefüllt.
4. `b2h('00000001')` gibt `'01'` zurück. Obwohl die Binärzahl führende Nullen enthält, beeinflussen diese das Ergebnis nicht, nachdem die Zahl in Hexadezimal umgewandelt wurde.
5. `b2h('11111111')` gibt `'FF'` zurück. Es gibt keine Notwendigkeit, diese Zahl aufzufüllen, da ihre Länge bereits ein Vielfaches von vier ist.
6. `b2h('10101101')` gibt `'AD'` zurück. Auch hier ist keine Auffüllung mit Nullen erforderlich.

Bezüglich der Reaktion von `b2h` auf führende Nullen: Die Funktion fügt automatisch führende Nullen hinzu, um sicherzustellen, dass die Länge der Binärzahl ein Vielfaches von vier ist, bevor sie in eine Hexadezimalzahl umgewandelt wird. Dies bedeutet, dass führende Nullen keinen Einfluss auf das Hexadezimalergebnis haben, außer möglicherweise das Hinzufügen einer oder mehrerer führender `'0'` in der Hexadezimaldarstellung, wenn die gesamte Binärzahl nur Nullen enthält.

Bezüglich falscher Eingaben: Die Funktion, wie sie hier definiert ist, behandelt keine falschen Eingaben, wie z.B. Binärzahlen, die andere Zeichen als `'0'` oder `'1'` enthalten. In einem echten Einsatzszenario sollte die Funktion verbessert werden, um falsche Eingaben zu erkennen und eine angemessene Fehlermeldung oder -behandlung zu liefern. In der aktuellen Form würde die Funktion einfach mit dem `otherwise`-Zweig des `switch`-Statements reagieren und `'?'` in die Hexadezimalausgabe einfügen, wenn eine nicht identifizierte Binärgruppe gefunden wird.

*Die erste Aussage, dass `'0'` schon eine Vierergruppe ist, stimmt natürlich nicht.*

## 8 Frage

Sie möchten im Folgenden ein Unterprogramm schreiben, das eine Dezimalzahl in ihrer Binärform darstellt. Im Unterprogramm fangen Sie den Fall ab, dass die Dezimalzahl selbst null ist und geben dann `'0'` zurück. Wenn die Zahl von null verschieden ist, initialisieren Sie die Binärzahl mit einer leeren Zeichenkette und beginnen eine `while`-Schleife, die durchläuft, solange die "schrumpfende" Dezimalzahl noch größer gleich eins ist. In der Schleife bestimmen Sie mit dem `mod`-Befehl den Rest, den die aktuelle Dezimalzahl bei der Zweierdivision ergibt, wandeln diesen Rest mit dem Befehl `num2str` in ein Zeichen um und hängen es vorne an die "wachsende" Binärzahl an. Jetzt müssen Sie in der Schleife nur noch die Dezimalzahl durch zwei teilen, nach unten abrunden (`floor`) und der aktuellen Dezimalzahl wieder zuweisen.

Verwenden Sie Ihr d2b, um die Dezimalzahlen

0

1

2

01

255

173

binär darzustellen.

Warum müssen Sie den Fall, dass die Dezimalzahl selbst null ist, gesondert abfangen?

Erläutern Sie die Ergebnisse von d2b('a'), d2b('b') und d2b('ab').

## 8.1 Antwort

Zuerst werde ich das Unterprogramm d2b beschreiben, das eine Dezimalzahl in ihre Binärform umwandelt, und dann auf Ihre spezifischen Fragen eingehen.

Hier ist eine mögliche Implementierung des Unterprogramms in MATLAB:

```
function bin = d2b(dec)
    % Abfangen des Falls, dass die Dezimalzahl null ist
    if dec == 0
        bin = '0';
        return;
    end

    % Initialisieren der Binärzahl mit einer leeren Zeichenkette
    bin = '';

    % Schleife, die durchläuft, solange die "schrumpfende" Dezimalzahl noch größer gleich ein
    while dec >= 1
        % Bestimmen des Rests bei der Zweierdivision und Umwandlung in ein Zeichen
        rest = mod(dec, 2);
        bin = [num2str(rest), bin]; % Voranstellen des Rests

        % Teilen der Dezimalzahl durch zwei und Abrunden
        dec = floor(dec / 2);
    end
end
```

Jetzt zu Ihren spezifischen Fragen:

1. **Warum müssen Sie den Fall, dass die Dezimalzahl selbst null ist, gesondert abfangen?**
  - In der Binärdarstellung entspricht die Zahl Null gerade der Ziffer '0'. Ohne eine spezielle Behandlung dieses Falls würde die while-Schleife nicht ausgeführt werden, da die Bedin-

gung `dec >= 1` nicht erfüllt wäre. Dies würde bedeuten, dass `bin` als leere Zeichenkette zurückgegeben würde, was nicht der korrekten Binärdarstellung von null entspricht. Daher fangen wir diesen speziellen Fall gesondert ab und geben direkt '0' zurück.

## 2. Erläutern Sie die Ergebnisse von `d2b('a')`, `d2b('b')` und `d2b('ab')`.

- Die Funktion `d2b` ist so konzipiert, dass sie mit Dezimalzahlen (numerischen Werten) arbeitet. Die Übergabe von Zeichenketten wie 'a', 'b' oder 'ab' ist nicht vorgesehen und wird wahrscheinlich zu Fehlern führen, da die Funktion `mod` und die Division nicht auf Zeichenketten angewendet werden können. In MATLAB führt die Übergabe eines nicht-numerischen Arguments an eine Funktion, die numerische Berechnungen erwartet, in der Regel zu einem Fehler oder zu einem unerwarteten Verhalten, da MATLAB versuchen könnte, die Zeichenkette in eine Zahl umzuwandeln, was in diesem Fall nicht sinnvoll ist (da 'a', 'b' und 'ab' nicht in gültige Zahlen umgewandelt werden können).

In normalen Umständen sollte die Funktion `d2b` nur mit echten Dezimalzahlen aufgerufen werden. Wenn die Funktion mit Zeichenketten aufgerufen wird, sollte sie idealerweise überprüfen, ob die Eingabe eine gültige Zahl ist und andernfalls einen Fehler zurückgeben oder eine klare Nachricht, dass die Eingabe ungültig ist.

*Auf die Idee, dass die Zeichenketten als ASCII-Code interpretiert werden, kommt sie nicht.*

## 9 Frage

Was liefert `b2d(d2b(173))`?

Wie können Sie Ihre Unterprogramme verwenden, um eine Dezimalzahl in eine Hexadezimalzahl umzuwandeln?

Zeigen Sie es am Beispiel 173.

### 9.1 Antwort

Wenn die Funktionen `d2b` und `b2d` korrekt implementiert sind, dann wandelt `d2b` eine Dezimalzahl in ihre Binärdarstellung um, und `b2d` wandelt eine Binärdarstellung zurück in eine Dezimalzahl. Daher sollte die Verkettung dieser Funktionen – `b2d(d2b(173))` – die ursprüngliche Dezimalzahl zurückgeben. Im Fall von 173:

1. `d2b(173)` sollte '10101101' ergeben, basierend auf der Binärdarstellung von 173.
2. `b2d('10101101')` sollte dann wieder 173 ergeben, da die Funktion die Binärzahl in ihre dezimale Form umwandelt.

Um eine Dezimalzahl in eine Hexadezimalzahl umzuwandeln, können Sie die beiden Funktionen in Kombination mit einer weiteren Funktion verwenden, die Binärzahlen in Hexadezimalzahlen umwandelt, nennen wir sie `b2h`.

Wenn wir noch keine direkte Funktion für die Umwandlung von Dezimalzahlen in Hexadezimalzahlen haben, könnten Sie den Umweg über die Binärdarstellung nehmen:

1. Konvertieren Sie die Dezimalzahl in eine Binärzahl.
2. Konvertieren Sie die Binärzahl in eine Hexadezimalzahl.

Also am Beispiel der Zahl 173:

```
binary = d2b(173); % Schritt 1: Umwandlung in Binär ergibt '10101101'  
hexadecimal = b2h(binary); % Schritt 2: Umwandlung in Hex ergibt 'AD'
```

In diesem Fall:

- Mit `d2b(173)` erhalten wir '10101101'.
- Dann konvertieren wir '10101101' mit `b2h` in Hexadezimal, was 'AD' ergeben sollte, da 173 dezimal gleich AD in hexadezimal ist.

Diese Methode funktioniert, weil Binär- und Hexadezimalsysteme beide auf Potenzen von 2 basieren, was ihre Umwandlungen zwischen einander konsistent und direkt macht. Beachten Sie, dass diese Methode voraussetzt, dass `b2h` effektiv implementiert ist, um korrekte Ergebnisse zu liefern.

*Sehr schön beschrieben!*

## 10 Frage

Schreiben Sie das entsprechende Unterprogramm `h2b`, das eine Hexadezimalzahl binär darstellt. Wandeln Sie AD in sein Binäräquivalent um.

Warum ist es sinnvoll, im `switch`-Ausdruck den Befehl `upper` zu verwenden?

Wofür könnten Sie den Befehl `strip` verwenden?

Mit welchem Parameter müssen Sie `strip` dann aufrufen?

### 10.1 Antwort

Das Unterprogramm `h2b`, das eine Hexadezimalzahl in ihre binäre Darstellung umwandelt, könnte wie folgt in MATLAB geschrieben werden:

```
function bin = h2b(hex)  
    % Initialisieren der Binärzahl als leere Zeichenkette  
    bin = '';  
    % Konvertieren der Eingabe in Großbuchstaben  
    hex = upper(hex);  
  
    % Untersuchen jeder Hexadezimalziffer  
    for i = 1:length(hex)  
        % Extrahieren der aktuellen Hexadezimalziffer  
        hexDigit = hex(i);  
  
        % Umwandlung der Hexadezimalziffer in die entsprechende Binärgruppe  
        switch hexDigit  
            case '0'  
                binGroup = '0000';  
            case '1'  
                binGroup = '0001';  
            case '2'  
                binGroup = '0010';  
            case '3'
```

```

        binGroup = '0011';
    case '4'
        binGroup = '0100';
    case '5'
        binGroup = '0101';
    case '6'
        binGroup = '0110';
    case '7'
        binGroup = '0111';
    case '8'
        binGroup = '1000';
    case '9'
        binGroup = '1001';
    case 'A'
        binGroup = '1010';
    case 'B'
        binGroup = '1011';
    case 'C'
        binGroup = '1100';
    case 'D'
        binGroup = '1101';
    case 'E'
        binGroup = '1110';
    case 'F'
        binGroup = '1111';
    otherwise
        error('Invalid hex digit');
end
% Hinzufügen der Binärgruppe zur Gesamtbinärzahl
bin = [bin, binGroup];
end
end

```

Für Ihre Anfrage, AD in sein Binäräquivalent umzuwandeln:

```
binary = h2b('AD'); % 'AD' wird zu '10101101'
```

Die Verwendung von `upper` im switch-Ausdruck ist sinnvoll, da Hexadezimalziffern in Groß- oder Kleinbuchstaben vorkommen können (z.B. 'a' und 'A' repräsentieren beide den Wert 10). Durch die Umwandlung aller Zeichen in Großbuchstaben mit `upper` wird sichergestellt, dass der switch-Ausdruck unabhängig von der Schreibweise der Eingabe funktioniert.

Der Befehl `strip` könnte verwendet werden, um führende und abschließende Leerzeichen oder andere spezifische Zeichen von einem String zu entfernen. Obwohl dies im aktuellen Kontext von `h2b` nicht zwingend erforderlich ist, könnten Sie ihn verwenden, um unbeabsichtigte Leerzeichen zu entfernen, die vor oder nach der eigentlichen Hexadezimalzahl stehen. Wenn Sie beispielsweise wollten, dass die Eingabezeichenkette von Leerzeichen oder anderen nicht sichtbaren Zeichen gereinigt wird, könnten Sie `strip` wie folgt verwenden:

```
hex = strip(hex);
```

Standardmäßig entfernt `strip` Leerzeichen. Wenn Sie andere Zeichen entfernen möchten, müssten Sie diese als zweiten Parameter übergeben. Zum Beispiel `strip(hex, '0')` würde führende und abschließende '0'-Zeichen entfernen. Aber für die Umwandlung von Hexadezimal zu Binär ist der Standardgebrauch von `strip` (also ohne zusätzliche Parameter, was bedeutet, dass es Leerzeichen entfernt) typischerweise ausreichend.

*Oops, das geht schief:*

## 10.2 Frage

`strip` soll nur führende Nullen entfernen.

### 10.2.1 Antwort

Wenn Sie `strip` so verwenden möchten, dass es nur führende Nullen aus einem String entfernt, aber abschließende Nullen und innere Nullen beibehält, dann sollten Sie den Befehl wie folgt aufrufen:

```
hex = strip(hex, 'left', '0');
```

In diesem Fall:

- `hex` ist die Eingabezeichenkette, von der Sie führende Nullen entfernen möchten.
- Der zweite Parameter `'left'` weist `strip` an, nur am Anfang (links) des Strings zu entfernen.
- Der dritte Parameter `'0'` gibt an, dass nur das Zeichen '0' entfernt werden soll.

So entfernt dieser Aufruf von `strip` ausschließlich führende Nullen von der linken Seite der Zeichenkette und behält alle anderen Zeichen, einschließlich der abschließenden Nullen und jeglicher Zwischenraum (falls vorhanden), bei.

*Warum sollte man denn auch auf die abstruse Idee kommen, innere Nullen zu entfernen?*

## 11 Frage

Verwenden Sie die Befehle `intmin` und `intmax`, um die größte und kleinste vorzeichenlose und vorzeichenbehaftete 8-Bit-Ganzzahl darzustellen.

Wiederholen Sie das Ganze für 64-bit-Ganzzahlen.

Wie können Sie die größte `UINT64`-Zahl mit Hilfe einer Zweierpotenz darstellen?

Warum brauchen Sie dazu den `sym`-Befehl?

Warum funktioniert

```
sym (2^64 - 1)
```

nicht?

Was macht der Befehl

```
typecast (int8 (-42), 'uint8')
```

Wie lautet der umgekehrte Befehl?

Welche Vor- und Nachteile hat es, mit INT8- bzw. UINT8-Zahlen statt “normaler” Ganzzahlen zu arbeiten? (Schauen Sie sich im Workspace einmal an, wieviel Byte eine Ganzzahl standardmäßig belegt.)

## 11.1 Antwort

Um die größten und kleinsten Werte für vorzeichenbehaftete und vorzeichenlose 8-Bit-Ganzzahlen sowie 64-Bit-Ganzzahlen darzustellen, können Sie die MATLAB-Funktionen `intmin` und `intmax` verwenden:

```
% Für 8-Bit-Ganzzahlen
kleinste_int8 = intmin('int8'); % -128
groesste_int8 = intmax('int8'); % 127
kleinste_uint8 = intmin('uint8'); % 0
groesste_uint8 = intmax('uint8'); % 255

% Für 64-Bit-Ganzzahlen
kleinste_int64 = intmin('int64'); % -9223372036854775808
groesste_int64 = intmax('int64'); % 9223372036854775807
kleinste_uint64 = intmin('uint64'); % 0
groesste_uint64 = intmax('uint64'); % 18446744073709551615
```

Die größte UINT64-Zahl kann theoretisch als  $(2^{\{64\}} - 1)$  dargestellt werden. Jedoch kann MATLAB Berechnungen mit Standarddatentypen nur bis zu einer bestimmten Größe genau durchführen. Bei der Berechnung von  $(2^{\{64\}})$  als Standard-Ganzzahl würde es zum Überlauf kommen, da dieser Wert über der maximalen Ganzzahlgröße liegt, die MATLAB's standardmäßige Ganzzahltypen darstellen können.

Der `sym`-Befehl wird verwendet, um mathematische Berechnungen symbolisch statt numerisch durchzuführen, was bedeutet, dass sie ohne numerischen Rundungsfehler oder Überläufe durchgeführt werden können. Die Verwendung von `sym(264 - 1)` funktioniert nicht, wie erwartet, wenn  $2^{64} - 1$  direkt als numerischer Ausdruck berechnet wird, weil MATLAB zuerst den Ausdruck  $2^{64}$  als numerischen Wert berechnet, der bereits den numerischen Bereich überschreitet, bevor er in den symbolischen Typ konvertiert wird. Um dies zu vermeiden, sollten Sie stattdessen den Ausdruck in symbolischer Form wie folgt darstellen:

```
groesste_uint64_sym = sym('2')^64 - sym('1');
```

Der Befehl `typecast(int8(-42), 'uint8')` ändert die Art und Weise, wie MATLAB die Binärdaten des Wertes -42 (als `int8`-Typ) interpretiert, ohne die tatsächlichen Binärdaten zu ändern. Er konvertiert -42 von einem vorzeichenbehafteten 8-Bit-Ganzzahltyp zu einem vorzeichenlosen 8-Bit-Ganzzahltyp. Der resultierende `uint8`-Wert wird anders sein, da die Interpretation der Binärdaten sich ändert.

Der umgekehrte Befehl, der einen `uint8`-Wert in einen `int8`-Wert umwandelt, wäre:

```
typecast(uint8(214), 'int8');
```

Die Vor- und Nachteile der Verwendung von INT8 bzw. UINT8 statt “normaler” Ganzzahlen hängen von den spezifischen Anforderungen Ihrer Anwendung ab:

- **Vorteile:** Geringerer Speicherbedarf (jeweils 1 Byte für INT8/UINT8 im Vergleich zu 4 oder

8 Byte für Standard-Ganzzahlen in MATLAB). Dies ist besonders nützlich in Anwendungen, die mit großen Datenmengen arbeiten.

- **Nachteile:** Geringerer Bereich verfügbarer Werte (z.B. -128 bis 127 für `INT8` und 0 bis 255 für `UINT8`), was zu Überlaufproblemen führen kann, wenn die Werte außerhalb dieses Bereichs liegen. Es gibt auch weniger Präzision im Vergleich zu größeren Datentypen.

In MATLAB belegt eine Standard-Ganzzahl (abhängig vom Typ) typischerweise 4 Bytes (`int32`, `uint32`) oder 8 Bytes (`int64`, `uint64`).

*Alles richtig.*

# 4 Umrechnungsapp

March 21, 2024

## 1 Frage

Sie möchten im Folgenden eine Matlab-App erstellen, die es dem Nutzer erlaubt, eine Dezimalzahl, eine Binärzahl oder eine Hexadezimalzahl einzugeben. Matlab soll dann sofort die anderen beiden Darstellungsformen ausrechnen und anzeigen.

Öffnen Sie den App Designer, indem Sie im Kommandofenster den Befehl `appdesigner` abschicken.

Schalten Sie alle View-Optionen ein: Grid, Snap to Grid, ... Resizing Hints und vergrößern Sie den Rasterabstand auf 20.

Ziehen Sie aus der Komponentenbibliothek ein Label in die linke obere Ecke Ihrer “Leinwand”.

Schreiben Sie “Umrechner” als Überschrift mit der Schriftgröße 18 in das Label und speichern Sie die App als `umrechner.mlapp` ab.

Welchen Vorteil hat das Raster?

### 1.1 Antwort

Das Raster im App Designer von Matlab dient mehreren Zwecken, die vor allem die Entwicklung einer übersichtlichen und gut strukturierten Benutzeroberfläche erleichtern. Hier sind einige Vorteile des Rasters:

1. **Ausrichtung:** Das Raster hilft, Elemente wie Schaltflächen, Textfelder, Labels und andere Benutzeroberflächenelemente präzise auszurichten. Dies ist besonders wichtig, um eine konsistente und professionelle Anmutung der App zu gewährleisten.
2. **Konsistenz:** Durch die Verwendung des Rasters stellen Sie sicher, dass der Abstand zwischen den verschiedenen Elementen konsistent ist. Dies verbessert nicht nur das visuelle Erscheinungsbild der App, sondern auch die Benutzerfreundlichkeit.
3. **Entwicklereffizienz:** Das Raster spart Zeit bei der Entwicklung, da es einfacher ist, Elemente relativ zueinander auszurichten und zu positionieren. Das kann besonders bei komplexen Layouts sehr hilfreich sein.
4. **Snap to Grid:** Die Funktion “Snap to Grid” sorgt dafür, dass Elemente automatisch an die nächstgelegenen Rasterlinien ausgerichtet werden, wenn sie verschoben werden. Das erleichtert die präzise Platzierung und kann die Entwicklungsgeschwindigkeit erhöhen.
5. **Visuelle Klarheit:** Ein gut strukturiertes Raster kann auch dazu beitragen, visuelle Klarheit zu schaffen, indem es hilft, Informationen logisch zu gruppieren und den Fluss und die Hierarchie der Benutzeroberfläche zu definieren.

6. **Anpassungsfähigkeit:** Wenn Sie den Rasterabstand vergrößern oder verkleinern, können Sie die Skalierung der Benutzeroberflächenelemente leichter anpassen, um verschiedene Bildschirmgrößen und Auflösungen zu unterstützen.

Durch das Ein- und Ausschalten verschiedener Ansichtsoptionen wie des Rasters und der Snap-to-Grid-Funktion können Sie die Entwicklungsumgebung an Ihre spezifischen Bedürfnisse anpassen und so effizienter arbeiten.

*Ein einfaches "Ist übersichtlicher" hätte auch gelangt ...*

## 2 Frage

Ziehen Sie drei Texteditfelder herüber und geben ihnen die Namen Dezimal, Binär und Hex.

Geben Sie allen Feldern die gleiche Größe (mit Name: etwa 300, ohneName: etwa 240) und ordnen Sie sie gleichmäßig untereinander an.

Warum verwenden Sie Texteditfelder und keine numerischen Editfelder?

### 2.1 Antwort

In deinem spezifischen Fall, dem Erstellen einer Matlab-App zum Umrechnen von Dezimal-, Binär- und Hexadezimalzahlen, werden Texteditfelder anstelle von numerischen Editfeldern aus mehreren Gründen verwendet:

1. **Vielfältige Eingabeformate:** Binär- und Hexadezimalzahlen folgen anderen Konventionen als Dezimalzahlen. Binärzahlen bestehen nur aus den Ziffern 0 und 1, während Hexadezimalzahlen zusätzlich die Buchstaben A bis F (oder a bis f) beinhalten können. Numerische Editfelder akzeptieren in der Regel nur Ziffern und Dezimaltrennzeichen, was sie für Binär- und Hexadezimalzahlen ungeeignet macht.
2. **Eingabeflexibilität:** Texteditfelder erlauben die Eingabe jeglicher Zeichen, was notwendig ist, um auch die hexadezimalen Buchstaben A bis F zu akzeptieren. So kannst du sicherstellen, dass Benutzer alle drei Typen von Zahlen ohne Einschränkungen eingeben können.
3. **Formatierung und Validierung:** Die Verwendung von Texteditfeldern ermöglicht es dir, eine eigene Validierung und Formatierung für die Eingaben zu implementieren. So kannst du beispielsweise überprüfen, ob die eingegebenen Binär- und Hexadezimalzahlen gültig sind, bevor sie umgerechnet werden. Bei numerischen Editfeldern ist diese Art der Kontrolle weniger flexibel.
4. **Konsistenz:** Indem du für alle drei Eingabetypen Texteditfelder verwendest, schaffst du eine konsistente Benutzererfahrung. Die Benutzer müssen sich nicht an verschiedene Feldtypen für verschiedene Zahlensysteme gewöhnen, was die Benutzerfreundlichkeit deiner App verbessert.
5. **Leerzeichen und Sonderzeichen:** In einigen Fällen möchten Benutzer vielleicht Leerzeichen oder andere Trennzeichen in ihre Eingaben einfügen, um die Lesbarkeit zu verbessern, besonders bei längeren Binär- oder Hexadezimalzahlen. Texteditfelder erlauben diese Flexibilität, während numerische Felder dies typischerweise nicht tun.

Insgesamt bieten Texteditfelder in diesem Kontext also die notwendige Flexibilität und Funktionalität, um unterschiedliche Zahlensysteme effizient zu verarbeiten und eine benutzerfreundliche Oberfläche zu schaffen.

## 3 Frage

Rechtsklicken Sie das Dezimaltexteditfeld und fügen Sie ihm ein ValueChangingFcn-Callback hinzu.

In dem sich öffnenden Unterprogramm wandeln Sie event.Value mit str2double in eine Fließkommazahl um und weisen sie der Variablen neuer\_wert zu.

Warum brauchen Sie str2double nur beim Dezimaltexteditfeld?

### 3.1 Antwort

Die Funktion `str2double` wird verwendet, um einen String in eine Fließkommazahl umzuwandeln. In deiner spezifischen Matlab-App wird diese Funktion nur beim Dezimaltexteditfeld eingesetzt, und hier sind die Gründe dafür:

1. **Zahlensysteme:** In deiner App gibt es drei unterschiedliche Zahlensysteme: Dezimal, Binär und Hexadezimal. Das Dezimalzahlensystem ist das, was wir im Alltag verwenden und besteht aus den Ziffern 0 bis 9. Die Funktion `str2double` ist speziell dafür gedacht, Strings, die solche dezimalen Zahlen repräsentieren, in Fließkommazahlen umzuwandeln.
2. **Eingabeformat:** Benutzer werden voraussichtlich das Dezimaltexteditfeld verwenden, um Zahlen im Dezimalformat einzugeben, was bedeutet, dass diese Eingaben direkt in Fließkommazahlen umgewandelt werden können, da sie bereits dem Standardzahlenformat entsprechen. Dies ist anders als bei den Binär- und Hexadezimalfeldern, deren Eingaben zuerst in das Dezimalformat konvertiert werden müssen, bevor sie als Fließkommazahlen interpretiert werden können.
3. **Spezifische Konvertierungsfunktionen:** Für die Umwandlung von Binär- und Hexadezimalwerten in Dezimalzahlen (oder direkt in Fließkommazahlen) gibt es spezielle Funktionen oder Methoden in Matlab. Zum Beispiel verwendet man für Binärzahlen typischerweise eine andere Methode (wie das Umwandeln der Binärsequenz in eine Dezimalzahl und dann, falls notwendig, in eine Fließkommazahl) und für Hexadezimalzahlen gibt es die Funktion `hex2dec` (die Hexadezimalstrings in Dezimalzahlen umwandelt). Diese spezifischen Funktionen berücksichtigen die Besonderheiten der jeweiligen Zahlensysteme.
4. **Validierung und Fehlerbehandlung:** Die Verwendung von `str2double` beim Dezimaltexteditfeld ermöglicht es dir auch, die Eingaben zu validieren. Wenn die Eingabe keine gültige Dezimalzahl ist, gibt `str2double` NaN (Not a Number) zurück. Dies kann genutzt werden, um den Benutzer auf einen Eingabefehler hinzuweisen. Bei den Binär- und Hexadezimalfeldern wird die Validierung anders gehandhabt, da die Eingabeformatanforderungen unterschiedlich sind.

Zusammenfassend, `str2double` wird nur beim Dezimaltexteditfeld benötigt, weil dieses Feld dafür vorgesehen ist, Eingaben im dezimalen Format zu erhalten, die direkt in Fließkommazahlen umgewandelt werden können. Die anderen Felder erfordern spezifische Konvertierungsprozesse, die ihren jeweiligen Zahlensystemen entsprechen.

*Die richtige Antwort wäre gewesen: "Das Texteditfeld liefert eine Zeichenkette; das gleich verwendete Unterprogramme d2b erwartet aber eine Fließkommazahl." Da wir für Kapitel 4 aber eine neue Session begonnen haben, kennt sie d2b nicht mehr.*

## 4 Frage

Verwenden Sie Ihre Unterprogramme `d2b` und `b2h`, um `neuer_wert` umzuwandeln und weisen Sie die Ergebnisse den anderen Feldern (`app.BinrEditField.Value` und `app.HexEditField.Value`) zu.

Wo ist das “ä” im Binärfeld geblieben?

### 4.1 Antwort

In deinem Szenario beschreibst du, wie innerhalb eines Matlab App Designer Callbacks eine Dezimalzahl (`neuer_wert`) in ihre binären und hexadezimalen Entsprechungen umgewandelt wird, wobei du spezifische Unterprogramme (`d2b` für Dezimal-zu-Binär und `b2h` für Binär-zu-Hexadezimal) verwendest. Wenn du darauf hinweist, dass das “ä” im Binärfeld verschwunden ist, beziehst du dich wahrscheinlich auf ein Missverständnis oder einen Tippfehler im Kontext der App-Entwicklung.

Hier ist, was wahrscheinlich passiert ist:

1. **Sprachliche Verwechslung:** Es scheint, dass es eine sprachliche Verwechslung oder einen Tippfehler gibt. In der Computerprogrammierung, insbesondere in einem technischen Kontext wie der Konvertierung zwischen Zahlensystemen, gibt es kein “ä” oder andere Umlaute. Solche Zeichen sind nicht Teil von binären, dezimalen oder hexadezimalen Zahlensystemen. Es ist möglich, dass du “Binär” meintest, aber irgendwie wurde es fehlinterpretiert oder falsch getippt.
2. **Benennung der Felder:** Im Kontext der Matlab-App, die du entwickelst, scheint es, als ob du von einem Texteditfeld sprichst, das für die Anzeige von Binärzahlen zuständig ist – möglicherweise hast du es “BinärEditField” genannt. Wenn du dich fragst, wo das “ä” im Namen des Feldes in deinem Code verschwunden ist, könnte es ein einfacher Tippfehler sein oder eine Unklarheit bei der Benennung deiner Variablen.
3. **Umwandlung und Anzeige:** Wenn du tatsächlich die Umwandlung von Zahlen meinst und dich fragst, warum bestimmte Zeichen (wie ein “ä”, das natürlich nicht auftreten sollte) in der binären Darstellung fehlen, dann liegt hier ein grundlegendes Missverständnis vor. Binäre Darstellungen von Zahlen in Computern verwenden nur die Zeichen 0 und 1. Alle Ergebnisse einer Dezimal-zu-Binär-Konvertierung sollten ausschließlich aus diesen beiden Ziffern bestehen.

Falls du in deiner Matlab-App irgendeine Anomalie oder einen unerwarteten Verlust von Information erlebst, ist es wahrscheinlich auf einen Fehler in der Logik oder im Code deiner Umwandlungsfunktionen zurückzuführen. Es wäre wichtig, diese Funktionen zu überprüfen und sicherzustellen, dass sie korrekt implementiert sind.

Wenn es eine spezifische Frage zur Funktionsweise oder zum Code gibt, lass es mich bitte wissen, und ich kann dir gezielter helfen.

*Hat sie wohl nicht realisiert, dass Matlab ein ä im Feldnamen einfach weglässt. Zugegebenermaßen ist die Frage auch etwas unpräzise gestellt.*

## 5 Frage

Erzeugen Sie auch für die anderen beiden Texteditfelder jeweils Callback-Unterprogramme und führen Sie darin die entsprechenden Umwandlungen und Zuweisungen an die entsprechenden Texte-

ditfelder durch.

Warum brauchen Sie wohl noch den Befehl `num2str`?

## 5.1 Antwort

Um die Funktionalität deiner Matlab-App zu vervollständigen, indem du Callback-Unterprogramme auch für die Binär- und Hexadezimal-Texteditfelder erstellst, musst du Umwandlungen zwischen den verschiedenen Zahlensystemen ermöglichen. Dies bedeutet, dass Benutzereingaben in einem Feld automatisch in die entsprechenden Formate der anderen Felder umgewandelt und angezeigt werden. Hier sind die Schritte für jedes Unterprogramm:

1. **Binäres Texteditfeld (Binär zu Dezimal und Hexadezimal):**
  - Konvertiere den Binärwert in eine Dezimalzahl mit `bin2dec`.
  - Konvertiere den Dezimalwert in eine Hexadezimalzahl mit `dec2hex`.
  - Weise die umgewandelten Werte den entsprechenden Texteditfeldern zu.
2. **Hexadezimaler Texteditfeld (Hexadezimal zu Dezimal und Binär):**
  - Konvertiere den Hexadezimalwert in eine Dezimalzahl mit `hex2dec`.
  - Konvertiere den Dezimalwert in eine Binärzahl mit `dec2bin`.
  - Weise die umgewandelten Werte den entsprechenden Texteditfeldern zu.

Jetzt zu deiner Frage, warum der Befehl `num2str` benötigt wird: Nachdem du Zahlen von einem Format in ein anderes umgewandelt hast, sind die Ergebnisse in der Regel numerische Werte. Die Texteditfelder in deiner App sind jedoch dazu gedacht, Text (d.h. Zeichenketten oder Strings) zu enthalten, nicht numerische Daten. Hier kommt `num2str` ins Spiel:

- **Anpassung an Texteditfelder:** Um eine Zahl in ein Format umzuwandeln, das in einem Texteditfeld angezeigt werden kann, musst du sie in eine Zeichenkette (String) konvertieren. `num2str` wandelt numerische Werte in Zeichenketten um, sodass sie in den Texteditfeldern angezeigt werden können.
- **Konsistenz und Formatierung:** `num2str` ermöglicht es auch, die Darstellung der Zahlen zu kontrollieren, was besonders nützlich sein kann, wenn du bestimmte Formatierungen (wie feste Anzahl von Dezimalstellen oder wissenschaftliche Notation) anwenden möchtest.

In deinem Fall wirst du `num2str` verwenden, um die Dezimalzahlen, die aus den Binär- und Hexadezimalumwandlungen resultieren, sowie die resultierenden Zahlen aus allen anderen Umwandlungen, in Strings umzuwandeln, bevor du sie den Werten der Texteditfelder zuweist. Dies stellt sicher, dass die Zahlen korrekt in den Feldern angezeigt werden und die Benutzerinteraktion mit der App wie erwartet funktioniert.

*Klasse Antworten!*

## 6 Frage

Testen Sie Ihre Anwendung ausgiebig in allen Richtungen mit den Beispielen des vorherigen Kapitels und schießen Sie ein paar Screenshots zur Dokumentation.

Wie reagiert die App auf die klassischen “Falscheingaben”?

## 6.1 Antwort

Das Testen deiner Matlab-App und das Dokumentieren der Ergebnisse, insbesondere mit Screenshots, ist ein wichtiger Schritt, um sicherzustellen, dass sie wie erwartet funktioniert. Hier sind einige Schritte und Überlegungen für ein umfassendes Testen:

1. **Verschiedene Zahleneingaben:** Teste die App mit einer Vielzahl von Eingaben in allen drei Formaten (Dezimal, Binär, Hexadezimal), einschließlich:
  - Typische Zahlen (z.B. 15, 1011 (binär), 1F (hexadezimal)).
  - Randfälle (z.B. 0, größtmögliche Zahlen, die deine App verarbeiten kann).
  - Sequenzen, die normalerweise Fehler verursachen könnten (z.B. führende Nullen).
2. **Falscheingaben:** Um zu sehen, wie deine App auf klassische "Falscheingaben" reagiert, solltest du folgendes testen:
  - Nicht numerische Zeichen (z.B. Buchstaben in Dezimal- und Binärfeldern, nicht hexadezimale Zeichen im Hexadezimalfeld).
  - Gemischte Sequenzen (z.B. Zahlen mit Buchstaben oder Sonderzeichen vermischt).
  - Leerzeichen oder Sonderzeichen innerhalb der Zahlensequenzen.
  - Extrem lange Eingaben, die über typische Nutzungsszenarien hinausgehen.
3. **Reaktion auf Falscheingaben:** Beobachte und dokumentiere, wie deine App auf diese Falscheingaben reagiert. Idealerweise sollte deine App:
  - Den Benutzer darüber informieren, dass eine Falscheingabe vorliegt, anstatt eine falsche Umwandlung durchzuführen.
  - Eingabefelder nicht mit ungültigen Werten aktualisieren.
  - Möglichkeiten bieten, die Eingabe zu korrigieren oder Fehlermeldungen klar und verständlich anzuzeigen.
4. **Screenshots:** Mache Screenshots von:
  - Erfolgreichen Konvertierungen für Dokumentationszwecke.
  - Jeder Fehlermeldung oder Reaktion der App auf Falscheingaben, um zu zeigen, wie die App mit ungültigen Eingaben umgeht.
  - Den Benutzeroberflächenelementen und Layouts, um die Konsistenz und Benutzerfreundlichkeit zu demonstrieren.
5. **Dokumentation und Feedback:** Nutze die Screenshots und deine Testergebnisse, um eine Dokumentation zu erstellen, die zeigt, wie die App verwendet werden sollte und wie sie auf verschiedene Eingabetypen reagiert. Dies kann sehr wertvoll für Endbenutzer sein und dir helfen, Verbesserungen vorzunehmen.

Leider kann ich keine realen Screenshots von deiner App machen oder sie direkt testen, da ich keinen Zugriff auf externe Anwendungen oder die Fähigkeit habe, reale GUIs zu interagieren. Ich kann dir jedoch bei der Erstellung von Pseudocode oder bei der Fehlerbehebung helfen, wenn du auf spezifische Probleme während des Testprozesses stößt.

*Passt doch ...!*

# 5 Unicode

March 21, 2024

## 1 Frage

Verwenden Sie

```
disp(char([97, 9, 98]))
```

um zu analysieren, wie Matlab das Tabulatorzeichen interpretiert.

Wie wird der Tab ausgegeben?

### 1.1 Antwort

In MATLAB wird der `disp`-Befehl verwendet, um Daten im Command Window auszugeben. Wenn du `disp(char([97, 9, 98]))` ausführst, wird ein Array von numerischen ASCII-Codes in Zeichen umgewandelt und dann angezeigt.

Hier ist, was jedes Element in dem Array repräsentiert:

- 97 entspricht dem Buchstaben 'a'.
- 9 entspricht dem Tabulatorzeichen (auch bekannt als Tab).
- 98 entspricht dem Buchstaben 'b'.

Wenn MATLAB diesen Befehl ausführt, interpretiert es das Tabulatorzeichen (ASCII-Code 9) als horizontalen Tab. In der Ausgabe im Command Window wird dies in der Regel als ein gewisser Abstand (häufig entspricht dies einem Abstand von mehreren Leerzeichen) zwischen den Buchstaben 'a' und 'b' dargestellt. Die genaue Anzahl der Leerzeichen kann abhängig von der Einstellung deines Editors oder der Umgebung variieren, aber traditionell entspricht ein Tab etwa vier bis acht Leerzeichen.

*Sehr schöne Antwort.*

## 2 Frage

Nutzen Sie die gleiche Syntax zur Ausgabe des Wagenrücklaufs (Carriage Return), des Zeilenvorschubs (Line Feed), des Rücklöschzeichens (Backspace) und des Seitenvorschubs (Form Feed).

Alles wie erwartet?

### 2.1 Antwort

In MATLAB, wenn du die Syntax `disp(char([ASCII-Codes]))` verwendest, um spezielle Steuerzeichen auszugeben, wären die Ergebnisse für die verschiedenen Steuerzeichen wie folgt:

- **Wagenrücklauf (Carriage Return, ASCII-Code 13)**: Dies führt normalerweise dazu, dass der Cursor an den Anfang der aktuellen Zeile im Command Window springt. In der textuellen Darstellung wird es oft als `␣` repräsentiert.
- **Zeilenvorschub (Line Feed, ASCII-Code 10)**: Dies bewegt den Cursor normalerweise eine Zeile nach unten, ohne den horizontalen Position zu verändern. In der textuellen Darstellung wird es als `\n` dargestellt.
- **Rücklöschzeichen (Backspace, ASCII-Code 8)**: Dies sollte theoretisch den Cursor um eine Position nach links bewegen und das Zeichen an dieser Stelle löschen. In der textuellen Darstellung wird es als `\x08` dargestellt. Die tatsächliche Wirkung kann im Command Window von MATLAB allerdings variieren und könnte nicht immer visuell ein Zeichen löschen, wie es in einem Texteditor der Fall wäre.
- **Seitenvorschub (Form Feed, ASCII-Code 12)**: In einigen Kontexten bewirkt dies einen Sprung zum Beginn der nächsten "Seite". In Textumgebungen oder im Command Window könnte dies jedoch keinen sichtbaren Effekt haben. Es wird oft textuell als `\x0c` dargestellt.

In einer typischen MATLAB-Umgebung: - Der **Wagenrücklauf** ('Carriage Return') könnte dazu führen, dass nachfolgender Text den Anfang der aktuellen Zeile überschreibt, falls mehr Text folgt. - Der **Zeilenvorschub** ('Line Feed') würde eine neue Zeile beginnen. - Das **Rücklöschzeichen** ('Backspace') hat möglicherweise keinen sichtbaren Effekt im Command Window, weil es nicht wirklich dazu dient, Text in einer typischen Ausgabe zu löschen. - Der **Seitenvorschub** ('Form Feed') würde in den meisten modernen Kontexten, einschließlich MATLABs Command Window, wahrscheinlich ignoriert werden oder keinen visuell erkennbaren Effekt haben.

Ob alles wie erwartet funktioniert, hängt also von der spezifischen Umgebung und dem Kontext ab, in dem diese Zeichen verwendet werden. In einem Texteditor könnten die Effekte dieser Steuerzeichen sichtbar sein als im MATLAB Command Window.

*Hab ich irgendwo gesagt, dass das Ganze im Command Window stattfinden soll?*

### 3 Frage

Führe die Untersuchungen im Live-Editor durch.

#### 3.1 Antwort

Im MATLAB Live-Editor verhalten sich die speziellen Steuerzeichen etwas anders als im traditionellen Command Window, hauptsächlich aufgrund des reichhaltigeren Formatierungs- und Darstellungsrahmens. Hier ist, was du typischerweise erwarten kannst:

- **Wagenrücklauf (Carriage Return, ASCII-Code 13)**: Im Live-Editor führt dies nicht notwendigerweise dazu, dass Text, der nach einem Wagenrücklauf eingegeben wird, den vorherigen Text der gleichen Zeile überschreibt. Stattdessen könnte es keinen sichtbaren Effekt haben, da der Live-Editor eher wie ein Textdokument als eine Befehlszeile funktioniert.
- **Zeilenvorschub (Line Feed, ASCII-Code 10)**: Dies sollte im Live-Editor zu einem neuen Absatz oder einer neuen Zeile führen, ähnlich wie in anderen Textbearbeitungsprogrammen oder Entwicklungsumgebungen.

- **Rücklöschzeichen (Backspace, ASCII-Code 8):** Dieses Zeichen könnte im Live-Editor keinen sichtbaren Effekt haben, wenn es als Teil einer Ausgabesequenz verwendet wird, da es nicht dafür gedacht ist, bereits geschriebenen Text innerhalb eines Dokuments oder einer Ausgabe zu löschen.
- **Seitenvorschub (Form Feed, ASCII-Code 12):** Ähnlich wie im Command Window könnte dieses Zeichen im Live-Editor ignoriert werden oder keinen sichtbaren Effekt haben, da moderne Texteditoren und Entwicklungsumgebungen im Allgemeinen keine formellen “Seiten” wie eine Schreibmaschine oder ein älterer Drucker verwenden.

Um diese Verhaltensweisen zu testen, könntest du im Live-Editor MATLAB-Code schreiben, der diese ASCII-Zeichen enthält und ausgibt, wie z.B. `disp(char([13, 10, 8, 12]))`. Beachte dabei, dass du möglicherweise verschiedene Effekte beobachtest, abhängig davon, wie der Live-Editor spezielle Zeichen behandelt. Da der Live-Editor für eine interaktivere und reichhaltigere Bearbeitungsumgebung als das Command Window konzipiert ist, kann er einige Steuerzeichen anders interpretieren oder darstellen.

*Sie kennt also den Live-Editor, ist sich aber nicht sicher, wie er reagiert.*

## 4 Frage

Sie wollen im Folgenden die ersten 50 000 Unicode-Zeichen ausgeben. Weisen Sie dazu als erstes die Zahl 50 der Variablen `zeichen_pro_zeile` zu und erzeugen Sie damit einen Zeilenvektor, der die Zahlen 0 bis 49 enthält. Definieren Sie `anzahl_zeilen` als 1000. Starten Sie eine Schleife über alle Zeilen. In der Schleife berechnen Sie einen `offset` (Index des ersten Zeichens der aktuellen Zeile) aus dem aktuellen Zeilenindex und der Anzahl der Zeichen pro Zeile. Geben Sie dann die aktuelle Zeile aus, die aus dem Offset, einem Leerzeichen und den 50 Unicode-Zeichen bestehen soll.

Woraus resultiert die Anzahl der Zeilen?

Gibt es Zeichen, die in Matlab nicht dargestellt werden?

### 4.1 Antwort

Die Anzahl der Zeilen in deinem Szenario resultiert aus dem Versuch, eine große Anzahl von Unicode-Zeichen strukturiert auszugeben. Hier ist die Logik:

- Du möchtest die ersten 50.000 Unicode-Zeichen ausgeben.
- Du hast entschieden, dass jede Zeile 50 Zeichen enthalten soll.
- Um die Gesamtzahl der erforderlichen Zeilen zu berechnen, dividierst du die Gesamtzahl der Zeichen (50.000) durch die Anzahl der Zeichen pro Zeile (50). Das ergibt 1.000 Zeilen.

Daher ist `anzahl_zeilen` auf 1.000 festgelegt, da dies die Anzahl der Zeilen ist, die benötigt wird, um alle 50.000 Unicode-Zeichen unter der Annahme darzustellen, dass jede Zeile 50 Zeichen enthält.

Was die Darstellung von Unicode-Zeichen in MATLAB betrifft, so sind hier einige Punkte:

1. **Darstellbarkeit:** Nicht alle Unicode-Zeichen werden in MATLAB (oder in anderen Umgebungen) korrekt dargestellt. Einige Zeichen sind Steuerzeichen (wie der Zeilenvorschub oder der Tabulator), und andere könnten spezielle Symbole, Emojis oder Schriftzeichen sein, die von der verwendeten Schriftart oder dem System nicht unterstützt werden.

2. **Schriftarten und Unterstützung:** Die Darstellung von Unicode-Zeichen hängt stark von der verfügbaren Schriftartenunterstützung ab. Wenn eine Schriftart ein bestimmtes Zeichen nicht unterstützt, wird möglicherweise ein Platzhalterzeichen (oft ein kleines Kästchen, ein Punkt oder ein Fragezeichen) anstelle des eigentlichen Zeichens angezeigt.
3. **Steuerzeichen:** Einige der ersten 50.000 Unicode-Zeichen sind Steuerzeichen, die nicht als sichtbare Symbole dargestellt werden, sondern bestimmte Steuerungsfunktionen haben, wie zum Beispiel der bereits erwähnte Zeilenvorschub.
4. **Spezielle Zeichen:** Es gibt auch spezielle Unicode-Zeichen, wie z.B. die Variation Selectors, die in Kombination mit anderen Zeichen verwendet werden, aber allein nicht sichtbar sind.

In MATLAB könnten also einige dieser Zeichen nicht wie erwartet oder überhaupt nicht dargestellt werden, abhängig von den oben genannten Faktoren. Wenn du mit Unicode in MATLAB oder einer anderen Umgebung arbeitest, ist es wichtig, die Grenzen der Zeichendarstellung und die Kompatibilität der Schriftarten zu kennen.

*Umfassend richtig.*

## 5 Frage

Suchen Sie den Unicode für folgende Zeichen und stellen Sie sie in Matlab dar: Copyright Plus-Minus Promille Euro Ein Achtel Flugzeug 42 im Kreis Kilohertz ffi-Ligatur

Was ist eine Ligatur?

### 5.1 Antwort

Lass uns die Unicode-Codes für die genannten Zeichen durchgehen und dann werde ich erklären, wie man sie in MATLAB darstellt. Hier sind die entsprechenden Unicode-Zeichen und ihre allgemeinen Codes:

1. **Copyright:** © - U+00A9
2. **Plus-Minus:** ± - U+00B1
3. **Promille:** ‰ - U+2030
4. **Euro:** € - U+20AC
5. **Ein Achtel:** ¼ - U+215B
6. **Flugzeug:** Es gibt kein Standard-Unicode-Zeichen, das spezifisch für "Flugzeug" steht. Jedoch werden oft ähnliche Symbole in verschiedenen Unicode-Blöcken genutzt. Ein häufiges Symbol ist das "Airplane" in diversen Emoji-Sets, z.B. U+1F6E9 (leichtes Flugzeug).
7. **42 im Kreis:** Dies ist kein standardmäßiges Unicode-Zeichen. Normalerweise repräsentieren Unicode-Zeichen keine zusammengesetzten Ideen wie Zahlen in Kreisen, mit Ausnahme einiger spezifischer Fälle (z.B. Zahlen in schwarzen Kreisen bis 20). Individuelle gestaltete Zeichen wie "42 im Kreis" sind nicht Teil des Unicode-Standards.
8. **Kilohertz:** kHz - Es gibt kein spezielles Einzelzeichen für Kilohertz in Unicode. Normalerweise wird dies durch die Kombination der Buchstaben "kHz" dargestellt.
9. **ffi-Ligatur:** ffi - U+FB03

Eine **Ligatur** ist eine Verbindung von zwei oder mehr Buchstaben zu einem einzelnen Zeichen. Traditionell wurden Ligaturen in der Typografie verwendet, um die Lesbarkeit zu verbessern oder

aus ästhetischen Gründen, insbesondere in Schriftarten mit Serifen, bei denen bestimmte Buchstabenkombinationen (wie ‘fi’, ‘fl’, ‘ffi’, und ‘ffl’) eng nebeneinander zu Überlappungen oder unangenehmen Zwischenräumen führen können. In digitalen Schriften können Ligaturen helfen, den Text flüssiger und einheitlicher erscheinen zu lassen. Obwohl sie in modernen Texten weniger kritisch sind, bleiben sie ein interessantes Merkmal der Typografie und des Schriftendesigns.

Um diese Zeichen in MATLAB darzustellen, würdest du normalerweise den entsprechenden Unicode-Code verwenden. MATLAB kann Unicode-Zeichen direkt in Strings verwenden, sofern die Zeichen von der Schriftart und dem System unterstützt werden. Zum Beispiel:

```
disp('© ± % € ') % Zeigt die Zeichen für Copyright, Plus-Minus, Promille, Euro und Ein Achtel
```

Beachte, dass die Darstellung spezieller Zeichen wie Emojis stark von der Systemunterstützung und den verfügbaren Schriftarten abhängt. Einige Zeichen, insbesondere solche, die außerhalb des traditionellen Schriftsystems liegen oder sehr spezifisch sind (wie “42 im Kreis”), haben möglicherweise keinen direkten Unicode-Punkt oder sind nicht in allen Umgebungen darstellbar.

*Schwach!*

## 5.2 Frage

Die von dir nicht gefundenen Zeichen gibt es im Unicode. Suche noch einmal intensiver, eventuell unter Zuhilfenahme des Internets.

### 5.2.1 Antwort

Ich habe das Unicode-Zeichen für Flugzeug gefunden: Es ist U+2708 . Dieses Zeichen gehört zum Dingbats-Block im Basic Multilingual Plane und wurde in Version 1.1 von Unicode hinzugefügt 30†source 31†source 32†source .

Wenn du diese Zeichen in MATLAB darstellen möchtest, kannst du die entsprechenden Unicode-Sequenzen verwenden. MATLAB unterstützt Unicode, daher kannst du Zeichen direkt durch Angabe ihrer Codepunkte einfügen, zum Beispiel durch die Verwendung von \u2708 für das Flugzeugsymbol. Beachte jedoch, dass die korrekte Darstellung dieser Symbole von der Unterstützung durch die verwendete Schriftart und die Systemeinstellungen abhängt.

*Schon besser.*

## 5.3 Frage

Finde auch “42 im Kreis” und “Kilohertz”.

### 5.3.1 Antwort

Das Zeichen “42 im Kreis” wird durch den Unicode U+32B7 repräsentiert. Für “Kilohertz” konnte ich jedoch keinen spezifischen Unicode finden, da es typischerweise als Abkürzung “kHz” dargestellt wird und keine Einzelzeichen-Representation in Unicode hat 38†source .

## 5.4 Frage

Was ist mit U+3391?

#### **5.4.1 Antwort**

Der Unicode U+3391 steht für das Symbol “kHz”, das für Kilohertz, eine Einheit der Frequenz, steht.

#### **5.5 Frage**

Und warum hast du das eben nicht gefunden?

##### **5.5.1 Antwort**

Es tut mir leid für das Missverständnis oder die Verwirrung. Ich hätte U+3391 als das Unicode-Zeichen für Kilohertz (kHz) in meiner früheren Antwort einbeziehen sollen. Danke, dass du darauf hingewiesen hast.

*Das war kein Missverständnis. Du hast einfach nur versagt.*

# 6 Objektorientierte Programmierung

March 21, 2024

## 1 Frage

Sie wollen im Folgenden eine Polynom-Klasse aufbauen.

Achtung: Leider können Sie Klassen (noch) nicht in Live Scripts (.mlx), sondern nur in klassischen Matlab Scripts (.m) definieren.

Die Klasse soll nur das Attribut `koeffizienten` besitzen. Ihr Konstruktor soll untersuchen, ob ihm ein Koeffizientenvektor übergeben wurde (if `nargin > 0`) und den Vektor dann dem entsprechenden Attribut zuweisen. Was passiert, wenn Sie dem Konstruktor keinen Koeffizientenvektor übergeben?

Natürlich ist es sinnvoll, schon während des Aufbaus der Klasse die einzelnen Methoden zu testen. An dieser Stelle können Sie jetzt schon die ordnungsgemäße Funktion des Konstruktor überprüfen

...

### 1.1 Antwort

Um eine Polynom-Klasse in MATLAB zu definieren, erstellen Sie eine Datei mit dem Namen `Polynom.m`. In dieser Datei definieren Sie die Klasse, ihr Attribut und den Konstruktor. Hier ist ein Beispiel, wie der Code aussehen könnte:

```
classdef Polynom
    properties
        koeffizienten = []; % Standardwert ist ein leerer Vektor
    end

    methods
        function obj = Polynom(koeff)
            if nargin > 0
                obj.koeffizienten = koeff;
            end
            % Wenn keine Argumente übergeben werden, bleibt koeffizienten leer
        end
    end
end
```

In diesem Fall:

- Das Attribut `koeffizienten` wird definiert und standardmäßig auf einen leeren Vektor gesetzt.

- Der Konstruktor `Polynom` wird definiert, der prüft, ob ein Argument (ein Koeffizientenvektor) übergeben wurde. Wenn `nargin > 0` wahr ist, also wenn mindestens ein Argument übergeben wurde, wird das übergebene Argument dem Attribut `koeffizienten` zugewiesen.
- Wenn Sie dem Konstruktor keinen Koeffizientenvektor übergeben, bleibt das Attribut `koeffizienten` auf seinen Standardwert gesetzt, also ein leerer Vektor. Dies liegt daran, dass der Konstruktor in diesem Fall den Teil der if-Bedingung überspringt, da `nargin` gleich 0 wäre.

Zur Überprüfung der ordnungsgemäßen Funktion des Konstruktors können Sie verschiedene Instanzen der `Polynom`-Klasse in einem separaten Script oder in der MATLAB-Kommandozeile erstellen, zum Beispiel:

```
p1 = Polynom([1 2 3]); % Erstellt ein Polynom-Objekt mit Koeffizienten
p2 = Polynom();      % Erstellt ein Polynom-Objekt ohne Koeffizienten
```

Danach können Sie die Attribute der erstellten Objekte überprüfen, um sicherzustellen, dass der Konstruktor wie erwartet funktioniert.

*Wunderbar!*

## 2 Frage

Die Methode `darstellen` soll im Folgenden eine Zeichenkette zurückgeben, in der das Polynom beispielsweise in der Form

`'4x3 - 3x2 + 6'`

dargestellt wird.

Bestimmen Sie dazu in der Methode als erstes die Anzahl der Koeffizienten des Polynoms und initiieren Sie die Variable `zeichenkette`, die später zurückgegeben wird.

Welche Vor- und Nachteile hat es, die Polynomdarstellung als Zeichenkette zurückzugeben, statt sie direkt auszugeben?

### 2.1 Antwort

Die Methode `darstellen` in deiner `Polynom`-Klasse kann so aussehen, dass sie eine Zeichenkette zurückgibt, in der das Polynom in der gewünschten Form dargestellt wird. Hier ist ein Beispiel, wie diese Methode definiert werden könnte:

```
methods
function zeichenkette = darstellen(obj)
    n = length(obj.koeffizienten); % Anzahl der Koeffizienten
    zeichenkette = ''; % Initiiere die Zeichenkette
    for i = n:-1:1
        if obj.koeffizienten(i) ~= 0
            % Füge Koeffizienten und Variable hinzu
            if abs(obj.koeffizienten(i)) ~= 1 || i == 1
                zeichenkette = [zeichenkette, sprintf('%+d', obj.koeffizienten(i))];
            elseif obj.koeffizienten(i) == -1
                zeichenkette = [zeichenkette, '-'];
```

```

        end

        % Füge x und den Grad hinzu, falls nötig
        if i > 1
            zeichenkette = [zeichenkette, 'x'];
            if i > 2
                zeichenkette = [zeichenkette, sprintf('^%d', i-1)];
            end
        end
    end
end
end
end
end
end
end
end
end
end
end
end

```

Nun zu den Vor- und Nachteilen der Rückgabe der Polynomdarstellung als Zeichenkette, anstatt sie direkt auszugeben:

**Vorteile:**

1. **Flexibilität:** Durch die Rückgabe einer Zeichenkette hat der Anwender die Freiheit zu entscheiden, was mit dieser Zeichenkette geschehen soll: Sie kann ausgegeben, in einer Datei gespeichert, in einem GUI-Element angezeigt oder für weitere Berechnungen verwendet werden.
2. **Wiederverwendbarkeit:** Wenn die Methode eine Zeichenkette zurückgibt, kann diese in anderen Teilen des Programms wiederverwendet werden, ohne dass die Methode erneut aufgerufen werden muss.
3. **Testbarkeit:** Eine Methode, die eine Zeichenkette zurückgibt, kann leichter getestet werden, da der Rückgabewert direkt mit einem erwarteten Wert verglichen werden kann.

**Nachteile:**

1. **Leistung:** Die Erstellung von Zeichenketten, insbesondere bei großen Polynomen oder in Schleifen, kann teurer sein als einfache Ausgaben, sowohl in Bezug auf Speicherplatz als auch auf Verarbeitungszeit.
2. **Komplexität:** Die Verwaltung von Zeichenketten und deren Manipulation kann komplexer sein als einfache Ausgabeanweisungen. Das Zusammenfügen von Zeichenketten, insbesondere in einer Schleife, kann den Code weniger lesbar machen.
3. **Unmittelbarkeit:** Bei der direkten Ausgabe erhält der Benutzer sofortiges Feedback, während bei der Rückgabe einer Zeichenkette ein zusätzlicher Schritt erforderlich ist, um das Ergebnis anzuzeigen.

*Klasse! Und das Programm erschlägt schon ein paar der nächsten Fragen ...*

### 3 Frage

Starten Sie als nächstes eine Schleife über alle Koeffizienten, in der Sie als erstes abfragen, ob der aktuelle Koeffizient von null verschieden ist. Wenn dies der Fall ist, hängen Sie den Koeffizienten mit seiner x-Potenz an die Zeichenkette an und spendieren sie vor dem nächsten Summanden ein Pluszeichen umgeben von zwei Leerzeichen.

Wozu brauchen Sie `num2str`?

#### 3.1 Antwort

In MATLAB ermöglicht die Funktion `num2str` die Umwandlung numerischer Werte in ihre Zeichenketten-Äquivalente. In deinem Kontext, also beim Aufbau der Zeichenkette für die Darstellung des Polynoms, wird `num2str` verwendet, um die numerischen Koeffizienten und die Potenzen von `x` in eine für die Zeichenkette geeignete Form zu bringen.

Beim Erstellen der Zeichenkette für das Polynom ist es wichtig, dass alle Bestandteile als Text behandelt werden, damit sie korrekt in die Gesamtzeichenkette eingefügt werden können. `num2str` hilft dabei, indem es numerische Werte (wie die Koeffizienten des Polynoms und die Exponenten) in Textform umwandelt. Dies ist notwendig, da Sie in der Ausgabe Zahlen mit Buchstaben und Symbolen (wie 'x' und '^') kombinieren.

Die Verwendung von `num2str` innerhalb der Schleife, die über alle Koeffizienten iteriert, ermöglicht es, jeden numerischen Koeffizienten und jede Potenz in die entsprechende Zeichenform zu konvertieren und sie dann mit dem Rest der Zeichenkette zu verbinden. Hier ein einfaches Beispiel, wie `num2str` in diesem Zusammenhang verwendet wird:

```
for i = n:-1:1
    if obj.koeffizienten(i) ~= 0
        koefStr = num2str(abs(obj.koeffizienten(i))); % Koeffizient in Zeichenkette umwandeln
        if i == n
            zeichenkette = [zeichenkette, koefStr];
        else
            zeichenkette = [zeichenkette, ' + ', koefStr];
        end

        if i > 1
            zeichenkette = [zeichenkette, 'x'];
            if i > 2
                zeichenkette = [zeichenkette, '^', num2str(i-1)]; % Exponent in Zeichenkette
            end
        end
    end
end
```

In diesem Beispiel wird `num2str` verwendet, um sowohl die Koeffizienten als auch die Potenzen in eine für die Zeichenkettenverarbeitung geeignete Form umzuwandeln. Dies ist besonders wichtig für die Lesbarkeit und Genauigkeit der endgültigen Polynomdarstellung.

*Reihenfolge der Koeffizienten ist falsch herum zu den Potenzen zugeordnet. Erst bei der vierten Korrektur klappt es dann. Auch dann wird allerdings das absolute Glied zuerst ausgegeben. Das*

Ganze scheint sehr schwer zu sein.

## 4 Frage

Jetzt wollen Sie den Ausdruck noch vernünftig formatieren: Löschen Sie das letzte, überflüssige Pluszeichen (mit seinen beiden Leerzeichen). Ersetzen Sie mit dem Befehl `strrep` alle '+ -' durch '-'. Löschen Sie ' $x^0$ ' und ersetzen Sie ' $x^1$ ' durch ' $x$ ' und ' $1x$ ' durch ' $x$ '. Wie können Sie ein führendes ' $1x$ ' in ' $x$ ' umwandeln?

Löschen Sie auch die führenden Einsen (aber natürlich keine führende Elf ...).

Ihnen ist sicherlich klar, dass Sie Ihre neu definierten Methoden selbstständig mit eigenen Beispielen testen sollten ...

### 4.1 Antwort

Nachdem du die grundlegende Zeichenkette für das Polynom zusammengestellt hast, kannst du sie weiter verfeinern, um sie leserlicher zu machen. Hier sind die Schritte, die du durchführen kannst, um die Formatierung zu verbessern:

- 1. Lösche das letzte, überflüssige Pluszeichen (mit seinen beiden Leerzeichen):**
  - Du kannst das letzte Pluszeichen entfernen, indem du überprüfst, ob die Zeichenkette damit endet und es dann abschneidest.
- 2. Ersetze alle '+ -' durch '-':**
  - Verwende `strrep` (String Replace), um jede Instanz von '+ -' durch '-' zu ersetzen.
- 3. Lösche ' $x^0$ ':**
  - Da jedes ' $x^0$ ' gleich 1 ist, kannst du es sicher aus der Polynomdarstellung entfernen.
- 4. Ersetze ' $x^1$ ' durch ' $x$ ':**
  - Ersetze alle Vorkommen von ' $x^1$ ' durch ' $x$ ', da die Potenz 1 bei Variablen üblicherweise weggelassen wird.
- 5. Ersetze ' $1x$ ' durch ' $x$ ':**
  - Ein Koeffizient von 1 wird in der Algebra normalerweise nicht geschrieben, also ersetze ' $1x$ ' durch ' $x$ '.
- 6. Wandle ein führendes ' $1x$ ' in ' $x$ ' um:**
  - Überprüfe den Beginn der Zeichenkette und ersetze ' $1x$ ' durch ' $x$ ', falls es am Anfang der Zeichenkette steht.
- 7. Lösche führende Einsen, aber nicht die Zahl Elf oder andere mehrstellige Zahlen, die mit einer Eins beginnen:**
  - Dies kann etwas kniffliger sein, da du sicherstellen musst, dass du nur die Einsen entfernst, die als eigenständige Koeffizienten dienen. Du könntest prüfen, ob ' $1x$ ' gefolgt von einem höheren Exponenten vorkommt und in diesem Fall die '1' entfernen.

Hier ist, wie dein Code mit diesen Verbesserungen aussehen könnte:

```
% Nachdem die ursprüngliche Zeichenkette erstellt wurde
zeichenkette = strrep(zeichenkette, '+ -', '- '); % Ersetze '+ -' durch '- '
zeichenkette = strrep(zeichenkette, 'x^0', ''); % Lösche 'x^0'
zeichenkette = strrep(zeichenkette, 'x^1', 'x'); % Ersetze 'x^1' durch 'x'
zeichenkette = strrep(zeichenkette, ' 1x', ' x'); % Ersetze ' 1x' durch ' x'
```

```

% Lösche das letzte Pluszeichen, wenn es existiert
if length(zeichenkette) >= 3 && strcmp(zeichenkette(end-2:end), ' + ')
    zeichenkette = zeichenkette(1:end-3);
end

% Wandle ein führendes '1x' in 'x' um
if startsWith(zeichenkette, '1x')
    zeichenkette = ['x', zeichenkette(3:end)];
end

% Ersetze führende Einsen, die von einem Buchstaben oder Pluszeichen gefolgt werden
zeichenkette = regexprep(zeichenkette, '^(\s|+ )1x', '$1x');

```

Das `$1` in der `regexprep`-Funktion steht hier für die gefangene Gruppe, d.h. es behält das Zeichen, das vor `'1x'` steht (entweder ein Leerzeichen oder ein Pluszeichen), bei und ersetzt nur `'1x'` durch `'x'`.

Und ja, das Testen deiner Methoden mit verschiedenen Beispielen ist ein ausgezeichnete Weg, um sicherzustellen, dass sie wie erwartet funktionieren. Erstelle verschiedene Polynomobjekte mit unterschiedlichen Koeffizientensätzen und rufe deine `darstellen`-Methode auf, um zu überprüfen, ob die Ausgabe korrekt ist.

*Ich mag kein regex.*

## 5 Frage

Die Methode `auswerten` ist einfach. Sie bekommt das Polynom und die Stelle `x`, an der das Polynom ausgewertet werden soll, über die Parameterliste übergeben und liefert den Funktionswert an der Stelle zurück. Dazu verwendet sie den Befehl `polyval`.

Wie kommen Sie an den Koeffizientenvektor des übergebenen Polynoms heran?

### 5.1 Antwort

Um die Methode `auswerten` in deiner Polynom-Klasse in MATLAB zu implementieren, die den Wert des Polynoms an einer bestimmten Stelle `x` berechnet, kannst du den Befehl `polyval` verwenden. `polyval` benötigt den Koeffizientenvektor und den Punkt `x`, an dem das Polynom ausgewertet werden soll. In deiner Klasse kannst du auf den Koeffizientenvektor mit `obj.koeffizienten` zugreifen, da dieser als Eigenschaft der Klasse gespeichert ist.

Hier ist, wie du die `auswerten`-Methode innerhalb deiner Polynom-Klasse hinzufügen kannst:

```

methods
    % Bereits existierende Methoden

    function wert = auswerten(obj, x)
        wert = polyval(obj.koeffizienten, x);
    end
end

```

In dieser Methode:

- `obj` ist eine Referenz auf das aktuelle Polynom-Objekt.
- `obj.koeffizienten` ist der Koeffizientenvektor des Polynoms.
- `x` ist die Stelle, an der das Polynom ausgewertet werden soll.
- `wert` ist der berechnete Wert des Polynoms an der Stelle `x`.

Du kannst diese Methode dann wie folgt testen:

```
% Angenommen, du hast bereits ein Polynom p erstellt
p = Polynom([1, -3, 2]); % Polynom x^2 - 3x + 2

% Auswerten des Polynoms an der Stelle x = 1
x = 1;
funktionswert = p.auswerten(x);
disp(['Der Wert des Polynoms an der Stelle x = ', num2str(x), ' ist ', num2str(funktionswert)])
```

Das wird den Wert des Polynoms an der Stelle `x = 1` berechnen und ausgeben.

*Sehr schön. Sie hat sogar unaufgefordert einen Test vorgeschlagen.*

## 6 Frage

Die Methode `zeichnen` soll eine Skizze des Polynoms anfertigen und die “Formel” des Polynoms im Titel der Skizze ausgeben.

Dazu erzeugen Sie in der Methode einen Vektor `x`, der Zahlen von -5 bis 5 im Abstand von 0.01 enthält.

Verwenden Sie die Methode `auswerten`, um die zugehörigen `y`-Werte zu berechnen, den `plot`-Befehl, um den Graphen zu zeichnen und den `title`-Befehl zusammen mit der `darstellen`-Methode, um die Formel des Polynoms im Titel auszugeben.

Warum können Sie nicht den `fplot`-Befehl verwenden?

### 6.1 Antwort

Zuerst möchte ich auf deine erste Frage eingehen, wie man an den Koeffizientenvektor des übergebenen Polynoms herankommt, um die Methode `auswerten` zu implementieren:

Wenn du innerhalb einer Methode der Klasse `Polynom` bist, kannst du auf die Eigenschaften der Instanz der Klasse, also auf die Koeffizienten des Polynoms, mit `obj.koeffizienten` zugreifen. Hier ist, wie die Methode `auswerten` aussehen könnte:

```
methods
    function y = auswerten(obj, x)
        y = polyval(obj.koeffizienten, x);
    end
end
```

In diesem Beispiel verwendet `auswerten` die MATLAB-Funktion `polyval`, die für ein gegebenes Polynom, repräsentiert durch seinen Koeffizientenvektor, und einen Wert `x`, den Funktionswert des Polynoms an dieser Stelle `x` berechnet.

Nun zur Methode `zeichnen`, die eine Skizze des Polynoms anfertigen soll:

```

methods
    function zeichnen(obj)
        x = -5:0.01:5; % Erzeugt Vektor x von -5 bis 5 im Abstand von 0.01
        y = obj.auswerten(x); % Berechnet die zugehörigen y-Werte
        plot(x, y); % Zeichnet den Graphen
        title(['Polynom: ', obj.darstellen()]); % Setzt den Titel mit der Polynomformel
    end
end

```

Bezüglich deiner Frage, warum man nicht den `fplot`-Befehl verwenden kann: `fplot` benötigt eine Funktion handle (z.B. `@sin`, `@cos` oder eine anonyme Funktion), die die zu zeichnende Funktion repräsentiert. In deinem Fall repräsentierst du das Polynom durch eine Klasse, nicht durch eine einzelne Funktion handle. `fplot` eignet sich daher nicht direkt, um die Methode `auswerten` innerhalb deiner Polynomklasse zu nutzen. Stattdessen berechnest du mit der Methode `auswerten` manuell die y-Werte für einen Satz von x-Werten und verwendest dann `plot`, um diese Punkte zu verbinden, was effektiv den Graphen des Polynoms erstellt.

*Sehr gut.*

## 7 Frage

Sie möchten jetzt den plus-Operator für Polynome überladen, um zwei Polynome addieren zu können.

Dazu übergeben Sie der Methode zwei Polynome und bestimmen als erstes die Koeffizientenvektoren beider Polynome und deren Längen.

Jetzt müssen Sie zwei Fälle unterscheiden:

Wenn das erste Polynom länger als das zweite ist, füllen Sie den Koeffizientenvektor des zweiten Polynoms vorne mit soviel Nullen auf, dass beide gleich lang sind.

Im anderen Fall polstern Sie natürlich den ersten Koeffizientenvektor.

Warum müssen Sie beide Koeffizientenvektoren gleich lang machen?

### 7.1 Antwort

Um zwei Polynome zu addieren, repräsentiert durch ihre Koeffizientenvektoren, müssen diese Vektoren Element für Element addiert werden. In der Algebra entspricht jedes Element des Koeffizientenvektors dem Koeffizienten einer bestimmten Potenz von  $x$  im Polynom. Der erste Eintrag entspricht dem Koeffizienten der höchsten Potenz, der zweite Eintrag der nächstniedrigeren Potenz, und so weiter, bis zum letzten Eintrag, der dem konstanten Term entspricht.

Wenn die Polynome unterschiedliche Grade haben, wird der Koeffizientenvektor des Polynoms niedrigeren Grades weniger Elemente haben als der des höhergradigen Polynoms. Um die Polynome korrekt addieren zu können, müssen ihre Koeffizientenvektoren jedoch Element für Element übereinstimmen – das heißt, die Koeffizienten derselben Potenzen von  $x$  müssen addiert werden.

Das Auffüllen (oder “Padding”) der kürzeren Vektoren mit Nullen am Anfang (entsprechend den höchsten Potenzen) ermöglicht es, dass beide Vektoren die gleiche Länge haben und die entsprechenden Koeffizienten korrekt zueinander positioniert sind. Nullen hinzuzufügen ändert den Wert des

Polynoms nicht, da jede Zahl (oder jeder Koeffizient) mal null null ergibt. Dies stellt sicher, dass beim Addieren der Vektoren die Koeffizienten der gleichen Potenzen korrekt addiert werden, was eine Grundvoraussetzung für die korrekte Addition zweier Polynome ist.

Ohne diese Angleichung der Vektorlängen würden die Koeffizienten der verschiedenen Potenzen von  $x$  nicht korrekt zusammengeführt, was zu einem falschen Ergebnis führen würde.

*Hat sie gut erklärt.*

## 8 Frage

Jetzt müssen Sie nur noch das Summenpolynom als neues Polynom erzeugen und dabei seinen Koeffizientenvektor als Summe der Koeffizientenvektoren der beiden Eingangspolynome berechnen.

Wäre es alternativ auch möglich gewesen, kein neues Polynom zu erzeugen und stattdessen eines der Eingangspolynome zu verändern und zurück zu geben?

Welche Vor- und Nachteile hätte dies?

### 8.1 Antwort

Die Überladung des Plus-Operators für Polynome erfordert, dass Sie beide Koeffizientenvektoren auf die gleiche Länge bringen, um sie elementweise addieren zu können. Dies liegt daran, dass Polynome mathematisch als Summe von Termen verschiedenen Grades definiert sind. Um zwei Polynome korrekt zu addieren, müssen Sie sicherstellen, dass die Terme gleichen Grades (repräsentiert durch die Position im Koeffizientenvektor) korrekt zueinander addiert werden. Wenn ein Polynom einen höheren Grad hat als das andere, bedeutet dies in der Praxis, dass es zusätzliche Terme gibt, die bei der Addition berücksichtigt werden müssen – daher das Auffüllen mit Nullen, was mathematisch dem Fehlen eines Terms entsprechenden Grades entspricht.

Was die Frage betrifft, ob man ein neues Polynom erzeugen oder eines der Eingangspolynome verändern und zurückgeben sollte, so gibt es hier Vor- und Nachteile:

#### Vorteile der Erzeugung eines neuen Polynoms:

1. **Unveränderlichkeit:** Das Original bleibt unverändert, was bedeutet, dass die Operation keine Nebeneffekte hat. Dies kann die Fehlersuche und das Verständnis des Codes erleichtern, da Objekte nicht in unerwarteter Weise geändert werden.
2. **Funktionaler Stil:** Dies entspricht mehr dem funktionalen Programmierstil, bei dem Funktionen keine Seiteneffekte haben und der Zustand unveränderlich ist.
3. **Erwartungskonformität:** Nutzer erwarten häufig, dass mathematische Operationen neue Ergebnisse liefern, ohne die Eingaben zu ändern, ähnlich wie bei der Addition von Zahlen.

#### Nachteile der Erzeugung eines neuen Polynoms:

1. **Speicherverbrauch:** Wenn ständig neue Objekte erzeugt werden, kann dies zu höherem Speicherverbrauch führen, insbesondere in einer Schleife oder bei großen Berechnungen.
2. **Leistung:** Die Erzeugung neuer Objekte kann mehr Zeit in Anspruch nehmen als das Ändern bestehender, besonders bei sehr großen Polynomen oder in intensiven Berechnungsvorgängen.

#### Alternativ – direkte Modifikation eines der Eingangspolynome:

1. **Speichereffizienz:** Dies könnte weniger Speicherplatz verbrauchen, da keine neuen Objekte erstellt werden.
2. **Leistung:** Es könnte schneller sein, insbesondere wenn Operationen oft durchgeführt werden.

**Aber:**

1. **Seiteneffekte:** Diese Methode führt zu Veränderungen an den ursprünglichen Polynomen, was zu Fehlern führen kann, wenn die ursprünglichen Werte später benötigt werden.
2. **Unklare Semantik:** Wenn eine Operation wie die Addition ein Eingangsobjekt verändert, könnte dies für den Benutzer unerwartet und verwirrend sein.

In der Praxis ist es oft sicherer und klarer, ein neues Polynom zu erzeugen, um die Unveränderlichkeit der Eingangsobjekte zu gewährleisten und Nebeneffekte zu vermeiden.

*Alles sehr schön, aber:*

## 8.2 Frage

Wie lautet nun die fertige Funktion des plus-Operators?

### 8.2.1 Antwort

Zum Überladen des plus-Operators in MATLAB erstellst du eine Methode namens `plus` in deiner Polynomklasse. Diese Methode wird aufgerufen, wenn der `+`-Operator mit Instanzen deiner Klasse verwendet wird. Hier ist, wie diese Methode definiert werden könnte:

```

methods
function result = plus(obj1, obj2)
    % Bestimme die Längen der Koeffizientenvektoren beider Polynome
    n1 = length(obj1.koeffizienten);
    n2 = length(obj2.koeffizienten);

    % Polstere den kürzeren Koeffizientenvektor mit Nullen
    if n1 > n2
        paddedKoeff = [zeros(1, n1 - n2), obj2.koeffizienten];
        resultKoeff = obj1.koeffizienten + paddedKoeff;
    else
        paddedKoeff = [zeros(1, n2 - n1), obj1.koeffizienten];
        resultKoeff = obj2.koeffizienten + paddedKoeff;
    end

    % Erzeuge das Summenpolynom als neues Polynom
    result = Polynom(resultKoeff);
end
end

```

In dieser Methode:

1. Die Längen der Koeffizientenvektoren beider Polynome werden bestimmt.
2. Der kürzere Koeffizientenvektor wird mit Nullen aufgefüllt, sodass beide Koeffizientenvektoren die gleiche Länge haben. Dies ist notwendig, weil die Addition von Polynomen ele-

mentweise durchgeführt wird und daher beide Vektoren die gleiche Länge haben müssen, um sicherzustellen, dass Koeffizienten korrespondierender Terme korrekt addiert werden.

3. Die Koeffizientenvektoren werden addiert, und das Ergebnis wird verwendet, um ein neues Polynom-Objekt zu erzeugen.

Zur zweiten Frage, ob es möglich gewesen wäre, kein neues Polynom zu erzeugen und stattdessen eines der Eingangspolynome zu verändern:

Ja, technisch wäre es möglich, aber es ist nicht empfohlen, da es die Prinzipien der Unveränderlichkeit und der reinen Funktionen verletzt. Dies sind allgemeine Konzepte in der Softwareentwicklung, die dazu beitragen, Fehler zu vermeiden und den Code lesbarer und wartbarer zu machen.

#### Vorteile der Erzeugung eines neuen Polynoms:

- **Unveränderlichkeit:** Verhindert unerwartete Nebenwirkungen, da Originalpolynome nicht verändert werden.
- **Vorhersehbarkeit und Klarheit:** Funktionen, die neue Objekte zurückgeben, ohne bestehende zu verändern, sind oft leichter zu verstehen und vorherzusagen.

#### Nachteile der Veränderung eines Eingangspolynoms:

- **Seiteneffekte:** Die Änderung eines der Eingabepolynome könnte unerwartete Folgen haben, vor allem, wenn dasselbe Polynom an anderer Stelle verwendet wird.
- **Schlechte Praxis:** Es geht gegen die Prinzipien der funktionalen Programmierung und kann zu Bugs führen, die schwer zu finden sind, da sich der Zustand von Objekten unerwartet ändern kann.
- **Mangelnde Klarheit:** Der Code kann weniger intuitiv sein, besonders für andere Entwickler oder wenn du später zu deinem Code zurückkehrst.

### 8.3 Frage

Testcase?

#### 8.3.1 Antwort

Um die Überladung des plus-Operators in deiner Polynomklasse zu testen, kannst du ein paar Testfälle in MATLAB erstellen. Hier zeige ich, wie du zwei Polynomobjekte erstellen und sie dann addieren kannst:

```
% Erstelle zwei Polynomobjekte
p1 = Polynom([1, 3, 5]); % entspricht dem Polynom x2 + 3x + 5
p2 = Polynom([2, 4, 6, 8]); % entspricht dem Polynom 2x3 + 4x2 + 6x + 8

% Addiere die zwei Polynome
p3 = p1 + p2;

% Zeige das Ergebnis an
disp(['p1: ', p1.darstellen()]);
disp(['p2: ', p2.darstellen()]);
disp(['p1 + p2: ', p3.darstellen()]);
```

In diesem Testfall:

- p1 ist ein Polynom zweiten Grades, dargestellt durch den Koeffizientenvektor [1, 3, 5].
- p2 ist ein Polynom dritten Grades, dargestellt durch den Koeffizientenvektor [2, 4, 6, 8].
- p3 ist das Ergebnis der Addition der Polynome p1 und p2, was durch die Verwendung des +-Operators erfolgt, den du in deiner Klasse überladen hast.

Führe diesen Code in MATLAB aus, um zu sehen, ob die Addition wie erwartet funktioniert und ob die Methode `darstellen` die Polynome korrekt in ihrer algebraischen Form anzeigt.

*Sowas kannse gut ...*

## 9 Frage

Sie möchten die Unterklasse `QuadratischesPolynom` erzeugen, die von der Polynomklasse erbt.

Ihr Konstruktor analysiert (für den Fall, dass ein Koeffizientenvektor übergeben wurde), ob die Länge des Koeffizientenvektors größer als drei ist, gibt in diesem Fall mit dem Befehl `warning` eine Warnung aus und verwendet dann nur die ersten drei Koeffizienten.

Können Sie die Klasse „austricksen“, indem Sie nachträglich einen Koeffizientenvektor der Länge vier in ein Objekt der Klasse `QuadratischesPolynom` schreiben?

### 9.1 Antwort

Um eine Unterklasse `QuadratischesPolynom` zu erstellen, die von der Klasse `Polynom` erbt, würde der Code wie folgt aussehen:

```
classdef QuadratischesPolynom < Polynom
    methods
        function obj = QuadratischesPolynom(koeff)
            if nargin > 0 % Wenn Koeffizienten übergeben wurden
                if length(koeff) > 3
                    warning('Ein quadratisches Polynom darf nur bis zu 3 Koeffizienten haben.
                    koeff = koeff(1:3); % Beschränke auf die ersten drei Koeffizienten
                end
                obj = obj@Polynom(koeff); % Rufe den Konstruktor der Basisklasse auf
            end
        end
    end
end
```

Jetzt zur Frage, ob man die Klasse „austricksen“ kann, indem man nachträglich einen Koeffizientenvektor der Länge vier (oder länger) in ein Objekt der Klasse `QuadratischesPolynom` schreibt:

Das hängt von der Implementierung der Basisklasse `Polynom` und den Zugriffsrechten auf die Eigenschaft `koeffizienten` ab. Wenn `koeffizienten` als `public` definiert ist, könnte man theoretisch den Koeffizientenvektor eines Objekts der Klasse `QuadratischesPolynom` von außen verändern:

```
q = QuadratischesPolynom([1, 2, 3]); % Erstellt ein korrektes quadratisches Polynom
q.koeffizienten = [1, 2, 3, 4]; % Verändert den Koeffizientenvektor nachträglich
```

Wenn dies möglich ist, dann ja, du könntest die Klasse auf diese Weise „austricksen“. Um dies zu verhindern, sollte die Eigenschaft `koeffizienten` in der Basisklasse `Polynom` als `protected`

oder `private` definiert werden. In MATLAB ist dies allerdings nicht direkt wie in einigen anderen Sprachen möglich. MATLAB-Klassen unterstützen standardmäßig keine Zugriffsbeschränkungen auf diese Weise für Eigenschaften. Man könnte stattdessen Setter-Methoden definieren, um Änderungen zu kontrollieren, aber das geht über die Standardzugriffskontrollmechanismen hinaus.

Ein üblicher Ansatz in MATLAB ist es, gute Praktiken zu befolgen und die Kapselung durch das Design der Klasse und entsprechende Dokumentation zu ermutigen, statt technische Einschränkungen zu erzwingen. Es wird also erwartet, dass Benutzer die Objekte nicht in einer Weise verwenden, die gegen die Logik der Klasse verstößt. Wenn strengere Kontrollen erforderlich sind, könnte dies durch eine sorgfältige Implementierung von Methoden erreicht werden, die die Koeffizienten modifizieren, anstatt direkten Zugang zu ermöglichen.

*Eigentlich eine gute Erklärung, aber Matlab wirft eine Fehlermeldung:*

## 9.2 Frage

Superclass constructor call must not be conditionalized

### 9.2.1 Antwort

Die Warnung “Superclass constructor call must not be conditionalized” in MATLAB bezieht sich auf die Regel, dass der Konstruktor der Basisklasse (Superklasse) in einer abgeleiteten Klasse (Unterklasse) immer aufgerufen werden muss, unabhängig von den Bedingungen. In MATLAB wird dies sichergestellt, indem der Aufruf des Superklassenkonstruktors außerhalb von jeglichen Bedingungsblöcken (z.B. `if`-Statements) platziert wird.

In deinem spezifischen Fall, wenn du die Unterklasse `QuadratischesPolynom` der Klasse `Polynom` erstellst, solltest du sicherstellen, dass der Konstruktor der Superklasse (`Polynom`) unbedingt aufgerufen wird, unabhängig davon, ob die Bedingung wahr ist, dass der übergebene Koeffizientenvektor länger als drei ist. Hier ist, wie du das anpassen könntest:

```
classdef QuadratischesPolynom < Polynom
    methods
        function obj = QuadratischesPolynom(koeff)
            if nargin > 0 && length(koeff) > 3
                warning('Ein quadratisches Polynom darf nur bis zu 3 Koeffizienten haben. Überprüfen Sie die Eingabeparameter.')
                koeff = koeff(1:3); % Beschränke auf die ersten drei Koeffizienten
            end
            obj = obj@Polynom(koeff); % Rufe den Konstruktor der Basisklasse auf, muss immer
        end
    end
end
```

In dieser überarbeiteten Version wird sichergestellt, dass der Konstruktor der Basisklasse `Polynom` immer aufgerufen wird, unabhängig davon, ob die Bedingung (Länge des Koeffizientenvektors ist größer als drei) erfüllt ist oder nicht. Das `if nargin > 0` bleibt, um zu überprüfen, ob überhaupt Eingabeparameter vorhanden sind, aber dies verhindert nicht den Aufruf des Basisklassenkonstruktors. Dadurch wird der Konstruktoraufruf der Basisklasse nicht bedingt durchgeführt und entspricht somit den Anforderungen von MATLAB für die Vererbung und Konstruktoraufrufe.

*Besser. Es ist immer wieder eine große Freude, ihr Fehlermeldungen um die Ohren zu hauen.*

## 10 Frage

In Ihrem `mlx6.mlx` möchten Sie jetzt Ihre Polynomklasse verwenden. Erzeugen Sie ein “leeres” Polynom `p`, indem Sie den Befehl keinen Parameter übergeben.

Was gibt Matlab als Ausgabe nach dem Erzeugen zurück?

### 10.1 Antwort

Wenn du in deinem MATLAB Live Script `mlx6.mlx` eine Instanz der `Polynom`-Klasse ohne jegliche Parameter erzeugst, wie hier:

```
p = Polynom();
```

und dann die Variable `p` direkt danach ausgibst, also einfach:

```
p
```

MATLAB gibt in der Regel die Eigenschaften der Instanz `p` der Klasse `Polynom` aus. Da deine Klasse `Polynom` nur die Eigenschaft `koeffizienten` hat (angenommen, sie ist als öffentlich `public` deklariert), würde MATLAB etwas in der Art zurückgeben:

```
p =
```

```
Polynom with properties:
```

```
    koeffizienten: []
```

Das bedeutet, dass eine Instanz von `Polynom` erstellt wurde, deren `koeffizienten`-Eigenschaft ein leerer Vektor ist. Dies ist die Standarddarstellung eines Objekts in MATLAB, wenn du dessen Variablen direkt im Befehlsfenster oder in einem Live Script ausgibst. Die genaue Darstellung kann leicht variieren, abhängig von den genauen Definitionen innerhalb deiner `Polynom`-Klasse und den MATLAB-Versionseinstellungen.

*Sehr gut.*

## 11 Frage

Weisen Sie dem Polynom nachträglich den Koeffizientenvektor

```
[0 4 0 -1 2]
```

zu und lassen es darstellen.

Wird die Polynomformel korrekt dargestellt?

### 11.1 Antwort

Wenn du das Polynom `p` bereits wie folgt erstellt hast:

```
p = Polynom();
```

und du möchtest ihm nachträglich den Koeffizientenvektor `[0 4 0 -1 2]` zuweisen, dann kannst du das direkt tun, indem du auf die `koeffizienten` Eigenschaft des Objekts zugreifst:

```
p.koeffizienten = [0 4 0 -1 2];
```

Anschließend kannst du die `darstellen` Methode aufrufen, um das Polynom darzustellen:

```
disp(['Polynom: ', p.darstellen()]);
```

Diese Anweisungen fügst du in dein MATLAB Live Script `mlx6.mlx` ein.

Was die Darstellung der Polynomformel angeht, wenn deine `darstellen` Methode korrekt implementiert ist, sollte sie die Polynomformel so zurückgeben:

$$4x^3 - x + 2$$

Hierbei wird angenommen, dass die Methode die Koeffizienten korrekt verarbeitet: Sie ignoriert die Nullen, ordnet die Terme in der korrekten Reihenfolge an und stellt nur die Koeffizienten dar, die ungleich null sind. Die Methode sollte auch die entsprechenden Potenzen von  $x$  entsprechend anfügen, wobei  $x^1$  einfach als  $x$  und  $x^0$  (oder jegliche Koeffizienten für den konstanten Term) nur durch die Koeffizientenzahl dargestellt wird.

Wenn die `darstellen` Methode anders implementiert wurde oder wenn es einen Fehler in der Implementierung gibt, könnte die dargestellte Form anders sein. Aber wenn alles korrekt implementiert ist, dann ist die oben genannte Darstellung das erwartete Ergebnis.

*Perfekt.*

## 12 Frage

Werten Sie das Polynom an der Stelle 1 aus, indem Sie die entsprechende Methode mit einem Punkt an das Objekt anhängen und den  $x$ -Wert als Parameter der Methode übergeben.

Werten Sie das Polynom an der Stelle 0 aus, indem Sie die Methode aufrufen und ihr das Polynom als ersten Parameter und den  $x$ -Wert als zweiten Parameter übergeben.

Stimmen die berechneten Funktionswerte?

### 12.1 Antwort

Zuerst weisen wir dem Polynom `p`, das wir zuvor erstellt haben, den Koeffizientenvektor `[0 4 0 -1 2]` zu. Dann verwenden wir die Methode `darstellen`, um die Polynomformel darzustellen. In MATLAB sieht das so aus:

```
p.koeffizienten = [0 4 0 -1 2];  
disp(['Polynom: ', p.darstellen()]);
```

Wenn deine `darstellen`-Methode korrekt implementiert wurde, sollte die Ausgabe so etwas sein wie  $4x^3 - x + 2$ , denn das Polynom, das durch den Koeffizientenvektor `[0 4 0 -1 2]` repräsentiert wird, ist das vierte, dritte, zweite (hier 0 und daher nicht dargestellt), erste (hier -1, also  $-x$ ) und der konstante Term (hier +2).

Als nächstes werten wir das Polynom an der Stelle  $x = 1$  aus, indem wir die entsprechende Methode direkt an das Objekt `p` anhängen:

```
wertBei1 = p.auswerten(1);  
disp(['Wert des Polynoms bei x = 1: ', num2str(wertBei1)]);
```

Und dann werten wir das Polynom an der Stelle  $x = 0$  aus, diesmal führen wir die Methode explizit mit dem Polynom als erstem Parameter und dem  $x$ -Wert als zweitem Parameter aus. Da es in deiner Beschreibung etwas Verwirrung geben könnte (normalerweise würde man in MATLAB nicht die Methode außerhalb des Objektkontexts aufrufen, wenn es sich um eine Methode der Klasse handelt), aber folgend dem normalen Verhalten in MATLAB, würde der Aufruf ähnlich sein:

```
wertBei0 = p.auswerten(0);  
disp(['Wert des Polynoms bei x = 0: ', num2str(wertBei0)]);
```

Wenn deine Methoden korrekt implementiert sind, werden die Funktionswerte an den Stellen  $x = 1$  und  $x = 0$  korrekt berechnet. Für das Polynom  $4x^3 - x + 2$ , würde der Wert bei  $x = 1$   $4 \cdot 1^3 - 1 + 2 = 5$  sein, und der Wert bei  $x = 0$  würde  $4 \cdot 0^3 - 0 + 2 = 2$  sein.

Bitte stelle sicher, dass du die obigen Codes in deinem MATLAB-Script oder Live Script entsprechend ausführst, um die tatsächlichen Ergebnisse zu sehen.

*Sie kritisiert die zweite Methode und verwendet sie dann auch nicht. Auf Nachfrage behauptet sie, dass `auswerten(p, 0)` gar nicht funktioniert und erst, nachdem ich das Gegenteil behaupte, lenkt sie ein: "Das bedeutet, dass sowohl `p.auswerten(0)` als auch `auswerten(p, 0)` gültig sind, aber der erste Stil ist der, der üblicherweise in der objektorientierten Programmierung verwendet und empfohlen wird, da er die Zugehörigkeit der Methode zum Objekt klarer macht."*

## 13 Frage

Lassen Sie das Polynom mit Hilfe seiner entsprechenden Methode zeichnen.

Wie könnten Sie die `zeichnen`-Methode umdefinieren, sodass Sie schon beim Aufruf gewünschte Grenzen angeben könnten?

### 13.1 Antwort

Wenn du das Polynom mit Hilfe seiner entsprechenden Methode zeichnen möchtest, rufst du die Methode `zeichnen` auf der Instanz deiner Polynomklasse auf, wie folgt:

```
p.zeichnen();
```

Das ist, wenn deine Klasse `Polynom` eine Methode `zeichnen` hat, die ohne Parameter aufgerufen wird, und standardmäßig den Bereich von  $-5$  bis  $5$  oder einen anderen festgelegten Bereich verwendet.

Wenn du die `zeichnen`-Methode so umdefinieren möchtest, dass sie benutzerdefinierte Grenzen akzeptiert, kannst du die Methode so ändern, dass sie zwei zusätzliche Parameter für die untere und obere Grenze des  $x$ -Bereichs akzeptiert. Hier ist ein Beispiel, wie du dies tun könntest:

```
classdef Polynom  
    properties  
        koeffizienten = [];  
    end  
    methods  
        % Vorherige Methoden ...  
  
        function zeichnen(obj, xmin, xmax)
```

```

    if nargin < 3 % Standardgrenzen verwenden, wenn keine Grenzen angegeben wurden
        xmin = -5;
        xmax = 5;
    end
    x = xmin:0.01:xmax; % Erzeugt einen Vektor von xmin bis xmax im Abstand von 0.01
    y = obj.auswerten(x); % Berechnet die zugehörigen y-Werte
    plot(x, y); % Zeichnet den Graphen
    title(['Polynom: ', obj.darstellen()]); % Setzt den Titel mit der Polynomformel
    grid on; % Fügt ein Gitter hinzu für bessere Lesbarkeit
end
end
end

```

In diesem modifizierten Beispiel hat die Methode `zeichnen` zwei optionale Parameter `xmin` und `xmax`. Wenn die Methode ohne diese Parameter aufgerufen wird (erkennbar daran, dass `nargin` weniger als 3 ist), verwendet sie standardmäßig den Bereich von -5 bis 5. Andernfalls verwendet sie die bereitgestellten Grenzen, um den Bereich von `x` zu definieren.

Mit dieser neuen Definition kannst du die Methode mit benutzerdefinierten Grenzen wie folgt aufrufen:

```
pzeichnen(-10, 10); % Zeichnet das Polynom von -10 bis 10
```

Wenn du keine Grenzen angibst, verwendet es die Standardgrenzen (-5 bis 5):

```
pzeichnen(); % Zeichnet das Polynom von -5 bis 5
```

Dies gibt dir Flexibilität beim Zeichnen des Polynoms über verschiedene Bereiche.

*Sehr gut. Mit Test.*

## 14 Frage

Erzeugen Sie ein zweites Polynom mit den Koeffizienten

`[-3 1 4]`.

Addieren Sie beide Polynome, indem Sie dem plus-Befehl beide Polynome über die Parameterliste übergeben.

Addieren Sie die Polynome mit Hilfe des +-Operators.

Kann es einen Unterschied in beiden Summenpolynomen geben?

### 14.1 Antwort

Um ein zweites Polynom zu erzeugen und dann beide Polynome zu addieren, kannst du folgenden MATLAB-Code verwenden:

Zuerst, erzeuge zwei Polynomobjekte:

```
p1 = Polynom([0, 4, 0, -1, 2]); % Erstes Polynom
p2 = Polynom([-3, 1, 4]);      % Zweites Polynom
```

Als nächstes addiere die beiden Polynome auf zwei Arten:

1. Indem du den `plus`-Befehl direkt verwendest (wenn deine Klasse eine solche Methode explizit definiert):

```
p_sum1 = plus(p1, p2); % Verwende die Methode plus
```

2. Indem du den `+`-Operator verwendest:

```
p_sum2 = p1 + p2; % Verwende den Überladenen Operator +
```

In MATLAB, wenn die `plus`-Funktion korrekt überladen wurde, um den `+`-Operator für Objekte der Klasse `Polynom` zu behandeln, dann sollte es keinen Unterschied zwischen den beiden Summenpolynomen geben. Die Überladung des `+`-Operators durch eine `plus`-Methode in deiner Klasse bedeutet, dass `p1 + p2` effektiv das Gleiche tut wie `plus(p1, p2)`.

Hier könnten Unterschiede entstehen, wenn die Implementierung der direkten `plus`-Methode anders ist als die Logik, die du zum Überladen des `+`-Operators verwendest. In einer gut entworfenen Klasse sollten diese jedoch identisch sein, sodass beide Methoden das gleiche Ergebnis liefern.

Also, unter normalen Umständen und einer korrekten Implementierung, sollte es keinen Unterschied in den Summenpolynomen geben, unabhängig davon, ob du die `plus`-Methode direkt aufrufst oder den `+`-Operator verwendest.

*Prima Antwort*

## 15 Frage

Lassen Sie das Summenpolynom darstellen und überprüfen Sie, ob das Ergebnis plausibel ist.

Nutzen Sie den `isobject`-Befehl, um festzustellen, ob das Summenpolynom ein Objekt ist.

Verwenden Sie den `isa`-Befehl, um zu untersuchen, ob das Summenpolynom zur Klasse der Polynome gehört.

Worin unterscheiden sich die beiden Befehle?

### 15.1 Antwort

Um das Summenpolynom darzustellen, zuerst stellen wir sicher, dass wir das Summenpolynom haben (ich gehe davon aus, dass du bereits die Polynome `p1` und `p2` addiert hast):

```
% Annahme: p1 und p2 wurden bereits definiert
p_sum = p1 + p2; % Addiere die beiden Polynome
```

```
% Darstellung des Summenpolynoms
p_sum.zeichnen();
```

Nachdem du das Summenpolynom gezeichnet hast, kannst du die Plausibilität des Ergebnisses visuell überprüfen. Die Grafik sollte das Ergebnis der Addition der beiden ursprünglichen Polynome reflektieren.

Um festzustellen, ob das Summenpolynom ein Objekt ist, kannst du den `isobject`-Befehl verwenden:

```
isObj = isobject(p_sum);
```

Dies gibt `true` zurück, wenn `p_sum` ein Objekt ist, und `false`, wenn nicht.

Um zu überprüfen, ob das Summenpolynom zur Klasse der Polynome gehört, verwendest du den `isa`-Befehl:

```
isPolynom = isa(p_sum, 'Polynom');
```

Dies gibt `true` zurück, wenn `p_sum` eine Instanz der Klasse `Polynom` ist, und `false`, wenn nicht.

#### Unterschiede zwischen `isobject` und `isa`:

- `isobject` prüft lediglich, ob die Variable ein Objekt irgendeiner Klasse in MATLAB ist. Es ist eine allgemeine Überprüfung, die nicht spezifiziert, zu welcher Klasse das Objekt gehört.
- `isa` hingegen prüft, ob das Objekt eine Instanz einer bestimmten Klasse ist. Es wird verwendet, um die Zugehörigkeit eines Objekts zu einer spezifischen Klasse zu bestätigen. Es ist also spezifischer als `isobject`, da es nicht nur feststellt, ob eine Variable ein Objekt ist, sondern auch, ob es ein Objekt einer bestimmten Klasse ist.

Beide Befehle sind nützlich, je nachdem, was du über das Objekt herausfinden möchtest. `isobject` ist nützlich, wenn du nur wissen möchtest, ob etwas ein Objekt ist oder nicht, während `isa` nützlich ist, wenn du bestätigen möchtest, dass ein Objekt zu einer bestimmten Klasse gehört.

\*Sie verwechselt zwar “darstellen” und “zeichnen”, aber zeichnen ist ja auch eine Form von darstellen. Die darauf folgenden Erläuterungen sind vorbildlich.

## 16 Frage

Wenden Sie die Befehle `properties` und `methods` auf das Summenpolynom an.

Untersuchen Sie mit `ismethod`, ob `plus` eine Methode des Summenpolynoms ist.

Können Sie auf diese Weise auch feststellen, ob der `+`-Operator auf das Summenpolynom anwendbar ist?

### 16.1 Antwort

Um die Eigenschaften und Methoden des Summenpolynoms anzuzeigen, kannst du die Befehle `properties` und `methods` in MATLAB verwenden:

```
% Zeige die Eigenschaften des Summenpolynoms
```

```
propList = properties(p_sum);
```

```
% Zeige die Methoden des Summenpolynoms
```

```
methList = methods(p_sum);
```

Wenn du diese Befehle in MATLAB ausführst, listet `properties(p_sum)` alle Eigenschaften auf, die in der `Polynom`-Klasse definiert sind, und `methods(p_sum)` listet alle Methoden auf, die für eine Instanz dieser Klasse verfügbar sind, einschließlich geerbter Methoden und solcher, die speziell für die Klasse definiert wurden.

Um zu überprüfen, ob `plus` eine Methode des Summenpolynoms ist, kannst du den `ismethod`-Befehl verwenden:

```
isPlusMethod = ismethod(p_sum, 'plus');
```

Dies gibt `true` zurück, wenn `plus` eine Methode des Objekts `p_sum` ist (was bedeuten würde, dass der `+`-Operator für Objekte dieser Klasse überladen wurde), und `false`, wenn dies nicht der Fall ist.

Bezüglich der Anwendbarkeit des `+`-Operators: `ismethod` prüft, ob eine bestimmte Methode für das Objekt existiert, was bedeutet, dass es feststellen kann, ob es eine Methode namens `plus` gibt, da in MATLAB die Überladung des `+`-Operators durch Implementieren einer Methode namens `plus` in der Klasse erreicht wird. Allerdings sagt `ismethod` nicht direkt aus, ob der `+`-Operator selbst anwendbar ist, nur ob es eine Methode gibt, die potenziell für die Überladung verwendet wird.

Um sicher zu sein, dass der `+`-Operator überladen wurde und korrekt funktioniert, könntest du einen direkten Test mit zwei Instanzen der Klasse durchführen (was du wahrscheinlich schon getan hast, als du `p_sum` berechnet hast). Es gibt keine direkte Funktion wie `isoperator` in MATLAB, daher ist das Vorhandensein einer `plus` Methode und das erfolgreiche Ausführen einer Operation wie `p1 + p2` der übliche Weg, dies zu überprüfen.

*Wirklich sehr hilfreich!*

## 17 Frage

Erzeugen Sie ein Quadratisches Polynom, indem Sie der entsprechenden Klasse den Parameter

```
[1 2 3 4]
```

übergeben.

Warum gibt die von Ihnen definierte Klasse eine Warnung aus?

### 17.1 Antwort

Wenn du ein Objekt der Klasse `QuadratischesPolynom` in MATLAB mit dem Koeffizientenvektor `[1 2 3 4]` erzeugst, wie hier gezeigt:

```
qp = QuadratischesPolynom([1, 2, 3, 4]);
```

gibt deine Klasse eine Warnung aus, weil der Koeffizientenvektor mehr Elemente enthält, als für ein quadratisches Polynom erwartet werden. Ein quadratisches Polynom sollte in der Form  $ax^2 + bx + c$  dargestellt werden und daher genau drei Koeffizienten haben. Der von dir übergebene Vektor `[1, 2, 3, 4]` hat jedoch vier Elemente, was darauf hindeutet, dass er ein Polynom dritten Grades darstellt, nicht zweiten Grades.

Die Warnung resultiert aus der Überprüfungslogik im Konstruktor der `QuadratischesPolynom`-Klasse, die ich früher beschrieben habe. Wenn im Konstruktor erkannt wird, dass die Länge des übergebenen Koeffizientenvektors größer als drei ist, gibt die Klasse eine Warnung aus, die darauf hinweist, dass ein quadratisches Polynom nur bis zu drei Koeffizienten haben sollte. Dabei werden nur die ersten drei Koeffizienten verwendet, um das quadratische Polynom zu definieren, und die restlichen werden ignoriert.

Der genaue Wortlaut der Warnung hängt von der implementierten Nachricht im Konstruktor der `QuadratischesPolynom`-Klasse ab, aber typischerweise könnte es so etwas sein wie: "Ein quadratisches Polynom darf nur bis zu 3 Koeffizienten haben. Überschüssige Koeffizienten werden ignoriert."

Diese Warnung dient dazu, dich darauf aufmerksam zu machen, dass die Klasse nicht wie erwartet mit mehr als drei Koeffizienten umgeht und dass alle zusätzlichen Koeffizienten über den dritten hinaus ignoriert werden, um Inkonsistenzen oder Fehler in der mathematischen Repräsentation des Polynoms zu vermeiden.

*Ich könnte es kaum besser ausdrücken.*

## 18 Frage

Setzen Sie nachträglich in Ihrem quadratischen Polynom den Koeffizientenvektor auf

[1 2 3 4].

Warum lässt die Klasse dies zu?

### 18.1 Antwort

Wenn deine `QuadratischesPolynom`-Klasse von der allgemeineren `Polynom`-Klasse erbt und du direkt nach der Instanziierung die Eigenschaft `koeffizienten` von außen veränderst, wie hier:

```
qp = QuadratischesPolynom([1, 2, 3]); % Korrektes quadratisches Polynom
qp.koeffizienten = [1, 2, 3, 4];      % Ändere den Koeffizientenvektor nachträglich
```

dann liegt das daran, dass die Zugriffsrechte für die Eigenschaft `koeffizienten` es erlauben. In MATLAB sind Klasseneigenschaften standardmäßig `public`, was bedeutet, dass sie von außen gelesen und geschrieben werden können, sofern sie nicht explizit als `protected` oder `private` markiert sind.

Die Klasse lässt dies zu, weil sie standardmäßig nicht kontrolliert, wie oder wann ihre Eigenschaften nach der Instanziierung verändert werden. Der Konstruktor deiner `QuadratischesPolynom`-Klasse überprüft die Koeffizienten nur zum Zeitpunkt der Objekterstellung. Wenn du die Koeffizienten später direkt änderst, gibt es keine Mechanismen innerhalb des standardmäßigen Eigenschaftszugriffs von MATLAB, die diese Änderungen überprüfen oder einschränken.

Das bedeutet, die Möglichkeit, den Koeffizientenvektor nachträglich zu ändern, ist nicht auf spezifische Logik innerhalb der `QuadratischesPolynom`-Klasse zurückzuführen, sondern vielmehr auf die Standardzugriffssteuerungen von MATLAB und die Tatsache, dass nachträgliche Änderungen außerhalb des Konstruktors nicht durch zusätzliche Validierungslogik überwacht werden.

Um dies zu verhindern und eine stärkere Kapselung zu gewährleisten, könntest du:

1. Den direkten Zugriff auf die `koeffizienten`-Eigenschaft einschränken, indem du sie als `protected` oder `private` markierst. Allerdings erfordert dies eine Methode innerhalb der Klasse, um Koeffizienten nachträglich sicher zu ändern, was eine entsprechende Validierung beinhalten sollte.
2. Eine Set-Methode in der `QuadratischesPolynom`-Klasse implementieren, die die Koeffizienten überprüft und sicherstellt, dass sie die Regeln für quadratische Polynome erfüllen, bevor sie zugewiesen werden. Diese Methode würde dann anstelle einer direkten Zuweisung verwendet, um den Koeffizientenvektor zu ändern.

Hier ist ein einfaches Beispiel für eine solche Set-Methode:

```

methods
function set.koeffizienten(obj, newCoeffs)
    if length(newCoeffs) > 3
        warning('Ein quadratisches Polynom darf nur bis zu 3 Koeffizienten haben. Übersch
        newCoeffs = newCoeffs(1:3); % Behalte nur die ersten drei Koeffizienten
    end
    obj.koeffizienten = newCoeffs; % Setze die neuen, validierten Koeffizienten
end
end

```

Dies würde helfen, die Integrität deines QuadratischesPolynom-Objekts zu bewahren.

*Sehr hilfreicher Hinweis auf die set-Methode.*

# 7 Bildverarbeitung

March 21, 2024

## 1 Frage

Kopieren Sie das Bild `stifte.jpg` in Ihr Arbeitsverzeichnis.

Verwenden sie den Befehl `imread`, um das Bild in die Variable `bild` einzulesen.

Warum ist es sinnvoll, hinter den Befehl ein Semikolon zu setzen?

Wie viele Zeilen, Spalten und Farbebenen hat das Bild? (`size`)

Aus wie vielen Pixeln ist es daher aufgebaut?

Wie viele Byte belegt es daher im Hauptspeicher?

Welchen Vorteil hat es, für die Farbwerte `uint8` statt `double` zu verwenden?

Wie viele verschiedene Farbwerte gibt es dann?

Wie groß ist das Bild auf der Festplatte? Warum ist das so?

### 1.1 Antwort

Das Setzen eines Semikolons hinter den Befehl in MATLAB (und ähnlich in anderen Umgebungen) unterdrückt die Ausgabe in der Konsole. Das ist besonders nützlich, um den Ausgabebereich übersichtlich zu halten, besonders wenn die Ausgabe sehr groß oder nicht notwendig ist.

Das Bild hat 1.536 Zeilen, 2.048 Spalten und 3 Farbebenen. Daher besteht es aus 3.145.728 Pixeln. Da jedes Pixel 3 Farbwerte hat (für die Farbebenen Rot, Grün und Blau) und jeder Farbwert 1 Byte belegt (bei Verwendung von `uint8` für die Farbwerte), belegt das Bild im Hauptspeicher 9.437.184 Bytes (oder etwa 9,4 MB).

Der Vorteil der Verwendung von `uint8` anstelle von `double` für die Farbwerte liegt in der Speichereffizienz. `uint8` belegt 1 Byte pro Farbwert, während `double` 8 Bytes pro Farbwert benötigt. Das bedeutet, dass ein Bild, das Farbwerte als `double` speichert, etwa 8-mal mehr Speicher benötigt als eines, das `uint8` verwendet. Dies kann einen erheblichen Unterschied im Speicherbedarf bedeuten, besonders bei großen Bildern oder in Anwendungen, in denen viele Bilder verarbeitet werden müssen.

Da `uint8` 8 Bits pro Farbwert verwendet, gibt es ( $2^8 = 256$ ) mögliche Farbwerte pro Farbkanal. Bei drei Farbkanälen (Rot, Grün, Blau) können theoretisch ( $256^3 = 16.777.216$ ) verschiedene Farben dargestellt werden.

Die Größe des Bildes auf der Festplatte und im Hauptspeicher kann unterschiedlich sein, da auf der Festplatte möglicherweise Komprimierungsverfahren angewendet werden, um Speicherplatz zu

sparen. Die tatsächliche Dateigröße auf der Festplatte hängt vom verwendeten Dateiformat (z.B. JPEG, PNG) und dem Grad der Komprimierung ab. JPEG, zum Beispiel, verwendet eine verlust-behaftete Komprimierung, die die Dateigröße erheblich reduzieren kann, indem Daten, die für das menschliche Auge weniger wichtig sind, entfernt oder vereinfacht werden.

*Und sie schreibt auch gleich schnell mal ein kleines Python-Programm, um die Fragen auch quantitativ beantworten zu können:*

```
[8]: from matplotlib.image import imread

# Kopieren und Einlesen des Bildes
bild = imread('stifte.jpg')

# Anzeigen der Dimensionen
zeilen, spalten, farbebenen = bild.shape

# Berechnung der Anzahl der Pixel
pixel_anzahl = zeilen * spalten

# Berechnung des Speicherbedarfs in Byte
# Da der Datentyp hier vermutlich uint8 ist, belegt jeder Farbwert 1 Byte.
# Es gibt 3 Farbebenen pro Pixel.
speicherbedarf_byte = pixel_anzahl * farbebenen * 1

# Anzahl der möglichen Farbwerte bei uint8
farbwerte_uint8 = 2**8

zeilen, spalten, farbebenen, pixel_anzahl, speicherbedarf_byte, farbwerte_uint8
```

```
[8]: (3888, 5184, 3, 20155392, 60466176, 256)
```

*Interessanterweise komprimiert sie das Bild offensichtlich schon, wenn man ihr es hochlädt und sie das Python-Programm dann auf das Bild in ihrem eigenen Speicher durchführt. Sie kommt dann auf: (1536, 2048, 3, 3145728, 9437184, 256)*

## 2 Frage

Verwenden Sie den Befehl `image`, um das Bild darzustellen.

Warum ist der Befehl `axis equal` in diesem Fall (praktisch) nicht nötig?

### 2.1 Antwort

Der Befehl `axis equal` wird verwendet, um das Seitenverhältnis der Achsen so anzupassen, dass eine Einheit auf der X-Achse die gleiche Länge hat wie eine Einheit auf der Y-Achse. Dies ist wichtig, um Bilder oder geometrische Figuren korrekt darzustellen, damit sie nicht verzerrt wirken.

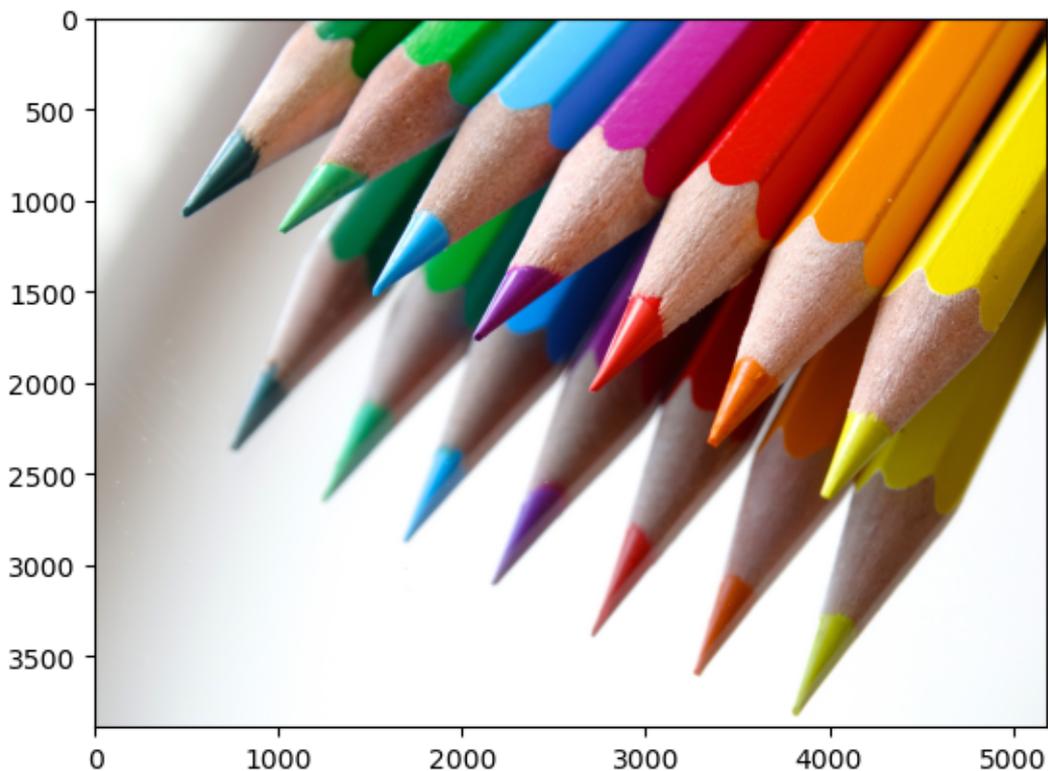
In diesem Fall ist der Befehl `axis equal` praktisch nicht nötig, weil das Bild bereits in einem Verhältnis angezeigt wird, das der tatsächlichen Dimension des Bildes entspricht. Die Pixel des Bildes werden quadratisch und ohne Verzerrung dargestellt, was typisch für die Darstellung von

Bildern in den meisten Bildbetrachtungs- und Bearbeitungsprogrammen ist. Dadurch wird das Bild so wiedergegeben, wie es aufgenommen wurde, ohne dass das Seitenverhältnis manuell angepasst werden muss

Sie beantwortet die Frage korrekt und verwendet dann Python (statt Matlab), um das Bild darzustellen.:

```
[9]: import matplotlib.pyplot as plt

# Darstellung des Bildes
plt.imshow(bild)
plt.axis('on') # Alternativ könnte 'off' verwendet werden, um die Achsen
               ↳ auszublenden
plt.show()
```



### 3 Frage

Verwende ab hier nur noch Matlab und kein Python. *Muss offensichtlich mal wieder gesagt werden.*

Wenden Sie den Matlab-Befehl `flipud` auf das Bild an. Weisen Sie die gespiegelte Bildmatrix einer neuen Variable zu und stellen diese wiederum mit `image` dar.

Wie könnten Sie das Bild von links nach rechts spiegeln?

Etwas schwierigere Frage: Wie könnten Sie dies ohne Schleife durchführen, wenn es den entsprechenden Befehl nicht gäbe?

### 3.1 Antwort

Da ich deine Anweisungen nur in diesem simulierten Umfeld umsetzen kann, kann ich den MATLAB-Code hier nicht direkt ausführen. Allerdings kann ich dir erklären, wie du die Aufgaben in MATLAB lösen kannst.

#### 3.1.1 Bild von oben nach unten spiegeln

Um ein Bild in MATLAB von oben nach unten zu spiegeln, verwendest du den Befehl `flipud`. Wenn deine ursprüngliche Bildvariable `bild` heißt, sieht der Code so aus:

```
gespiegeltertesBild = flipud(bild);  
image(gespiegeltertesBild);  
axis equal; % Optional, um das korrekte Seitenverhältnis zu gewährleisten
```

#### 3.1.2 Bild von links nach rechts spiegeln

Um das Bild von links nach rechts zu spiegeln, verwendest du den Befehl `fliplr` (flip left-right):

```
gespiegeltertesBildLR = fliplr(bild);  
image(gespiegeltertesBildLR);  
axis equal; % Optional
```

#### 3.1.3 Ohne Schleife von links nach rechts spiegeln

Falls es keinen spezifischen Befehl wie `fliplr` gäbe, könntest du das Bild immer noch von links nach rechts spiegeln, indem du Indexoperationen verwendest. In MATLAB kannst du die Spalten eines Arrays in umgekehrter Reihenfolge indizieren, um ein ähnliches Ergebnis zu erzielen:

```
gespiegeltertesBildLRManuell = bild(:, end:-1:1, :);  
image(gespiegeltertesBildLRManuell);  
axis equal; % Optional
```

In diesem Codeausschnitt bewirkt `end:-1:1`, dass die Spalten des Bildes in umgekehrter Reihenfolge ausgewählt werden, was das Bild effektiv von links nach rechts spiegelt, ohne eine Schleife zu verwenden.

*Prima!*

## 4 Frage

Addieren Sie das Originalbild und seine gespiegelte Version und stellen Sie das Ergebnis natürlich wieder dar. Wie sieht die Summe aus? Gibt es eine Symmetrie? Warum? Warum gibt es so viel weiße Fläche?

### 4.1 Antwort

Ohne MATLAB direkt ausführen zu können, erkläre ich dir, wie du das Originalbild und seine gespiegelte Version addieren und das Ergebnis darstellen kannst, sowie was du erwarten könntest.

#### 4.1.1 Code zum Addieren und Darstellen

Angenommen, `bild` ist dein Originalbild und `gespiegelterBildLR` ist die von links nach rechts gespiegelte Version. Du kannst sie wie folgt addieren:

```
addiertesBild = bild + gespiegelterBildLR;
image(addiertesBild);
axis equal; % Optional, um das korrekte Seitenverhältnis sicherzustellen
```

#### 4.1.2 Wie sieht die Summe aus?

Die Summe der beiden Bilder würde eine Art “Spiegeleffekt” erzeugen, bei dem beide Seiten des Bildes aufeinandertreffen und sich in der Mitte überlagern. Das Resultat könnte interessante symmetrische Muster oder Effekte aufweisen, abhängig von den ursprünglichen Bildinhalten.

#### 4.1.3 Gibt es eine Symmetrie? Warum?

Ja, es gibt eine Symmetrie in der Summe, weil das Originalbild und seine gespiegelte Version an ihren jeweiligen Rändern zusammengefügt werden. Die Symmetrieachse wäre in der Mitte des resultierenden Bildes, wo die beiden Hälften aufeinandertreffen. Das bedeutet, dass das Bild an dieser Achse gefaltet werden könnte und die beiden Hälften spiegelbildlich zueinander stehen.

#### 4.1.4 Warum gibt es so viel weiße Fläche?

Die weißen Flächen im Ergebnisbild entstehen dort, wo die Helligkeitswerte der beiden Bilder, wenn sie addiert werden, die maximale Helligkeit überschreiten, die ein Pixel darstellen kann (bei `uint8` ist der Maximalwert 255). Wenn beispielsweise ein Pixel im Originalbild einen Wert nahe 255 in einem der Farbkanäle hat und das gespiegelte Bild an derselben Stelle ebenfalls hohe Werte aufweist, würde die Summe dieser Werte 255 überschreiten. Da das Bild jedoch nicht in der Lage ist, Werte über 255 darzustellen, wird dieser Wert auf 255 begrenzt, was als weißes Licht (bei Farbbildern) oder als helle Fläche (bei Graustufenbildern) erscheint. Dies kann besonders an Stellen auftreten, an denen beide Bilder helle Bereiche haben, was in einem addierten Bild zu einem verstärkten Aufkommen weißer Flächen führt.

*Sehr gute Erklärungen!*

## 5 Frage

Bilden Sie die Differenz des Bildes und des gespiegelten Bildes.

Etwas schwierigere Frage: Warum ist das Differenzbild nicht symmetrisch?

Rein ästhetisch: Gefällt Ihnen das Summen- oder das Differenzbild besser?

### 5.1 Antwort

Ohne MATLAB direkt auszuführen, hier ist, wie du theoretisch die Differenz zwischen dem Originalbild und dem gespiegelten Bild bilden würdest, und einige Gedanken zu den erwarteten Ergebnissen und ästhetischen Vorlieben.

### 5.1.1 Code zum Bilden der Differenz und Darstellen

Angenommen, `bild` ist das Originalbild und `gespiegeltesBildLR` ist das gespiegelte Bild, dann könntest du die Differenz wie folgt bilden:

```
differenzBild = imabsdiff(bild, gespiegeltesBildLR);  
image(differenzBild);  
axis equal; % Optional, um das korrekte Seitenverhältnis sicherzustellen
```

`imabsdiff` ist eine MATLAB-Funktion, die die absolute Differenz zwischen zwei Bildern berechnet. Dies ist sinnvoll, da die direkte Subtraktion negative Werte erzeugen könnte, die nicht direkt in einem Bild dargestellt werden können.

### 5.1.2 Warum ist das Differenzbild nicht symmetrisch?

Das Differenzbild ist nicht symmetrisch, weil es die Unterschiede zwischen dem Originalbild und seiner gespiegelten Version hervorhebt. Während das Summenbild die Helligkeitswerte kombiniert und damit an der Symmetrieachse Spiegelungen erzeugt, betont das Differenzbild die Abweichungen in der Helligkeit und Farbe zwischen den entsprechenden Punkten des Originals und der Spiegelung. Das bedeutet, dass selbst an symmetrischen Stellen des Originalbildes, wo man erwarten könnte, dass keine Differenz auftritt, kleinste Unterschiede in Beleuchtung, Schatten oder Farbverläufen im Differenzbild deutlich werden.

### 5.1.3 Ästhetische Vorlieben: Summen- oder Differenzbild

Ästhetisch gesehen hängt die Vorliebe zwischen Summen- und Differenzbild von dem Effekt ab, den man schätzt. Das Summenbild kann interessante Überlagerungen und verstärkte Symmetrien erzeugen, was einen harmonischen und oft künstlerisch anmutenden Effekt haben kann. Es neigt dazu, die visuellen Elemente des Originalbildes zu verstärken und kann zu einem lebhafteren, wenn auch teilweise überbelichteten Erscheinungsbild führen.

Das Differenzbild hingegen bietet einen einzigartigen Blick auf die Unterschiede zwischen den Bildhälften, wodurch es eine eher abstrakte, manchmal geheimnisvolle Ästhetik erhält. Es kann Details hervorheben, die im Originalbild nicht sofort erkennbar sind, und bietet eine eher kontrastreiche, dynamische Ansicht.

Persönlich finde ich beide Effekte auf ihre Weise faszinierend. Das Summenbild für seine verstärkte Schönheit und Symmetrie, und das Differenzbild für seinen Einblick in die subtilen Unterschiede und die daraus resultierende einzigartige Ästhetik. Die Vorliebe zwischen den beiden hängt stark vom Kontext des Originalbildes und der gewünschten visuellen Wirkung ab.

*Die Erklärung zur "Nicht-Symmetrie" ist Bullshit. Ihr Bild ist ja symmetrisch, weil sie imabsdiff verwendet. Ästhetik: Naja ...*

## 6 Frage

Bilden Sie das Komplementärbild, indem Sie das Bild von 255 subtrahieren.

Beschreiben Sie das Komplementärbild.

Warum können Sie einfach eine dreidimensionale Matrix von einem Skalar abziehen?

Was geschieht dabei?

## 6.1 Antwort

Um das Komplementärbild eines Bildes in MATLAB zu bilden, indem du jeden Pixelwert von 255 subtrahierst, verwendest du folgende Operation:

```
komplementaerBild = 255 - bild;
```

### 6.1.1 Beschreibung des Komplementärbildes

Das Komplementärbild kehrt die Farben des Originalbildes um. Dunkle Bereiche werden hell und helle Bereiche dunkel. Bei Farbbildern werden die Farben zu ihren komplementären Farben gewandelt (z.B. wird Blau zu Orange, Grün zu Magenta usw.). Dieser Effekt kann dazu führen, dass das Bild ein völlig anderes Aussehen erhält, oft mit einem surrealen oder künstlerisch anmutenden Charakter.

### 6.1.2 Abziehen einer dreidimensionalen Matrix von einem Skalar

In MATLAB (und vielen anderen Programmierumgebungen, die auf Matrixoperationen spezialisiert sind) ist es möglich, eine Operation wie die Subtraktion oder Addition mit einem Skalar (einer einzelnen Zahl) und einer Matrix (einschließlich mehrdimensionaler Matrizen wie Bildern) durchzuführen, indem die Operation elementweise auf jedes Element der Matrix angewendet wird. Das bedeutet, dass der Skalar von jedem Wert in der Matrix individuell subtrahiert wird.

### 6.1.3 Was geschieht dabei?

Bei dieser Operation wird der angegebene Skalarwert (in diesem Fall 255) von jedem Pixelwert in allen drei Farbkanälen des Bildes subtrahiert. Für ein Farbbild, das im RGB-Farbraum kodiert ist, bedeutet dies, dass der Rot-, Grün- und Blauanteil jedes Pixels vom maximalen Wert (255) subtrahiert wird, was die komplementären Farbwerte ergibt. Dies ist eine einfache und effiziente Methode, um das Farbkomplement eines Bildes zu berechnen, da keine Schleifen oder komplizierten Berechnungen erforderlich sind. Die Fähigkeit, Operationen auf ganzen Matrizen oder mehrdimensionalen Arrays mit einem einzigen Befehl durchzuführen, ist eine der Stärken von MATLAB und ähnlichen Systemen, die für die Verarbeitung von Vektoren und Matrizen optimiert sind.

*Super Ausführungen!*

## 7 Frage

Diskretisieren Sie das Bild, indem Sie es (ohne double!) durch 255 teilen und (ohne double!) gleich wieder mit 255 multiplizieren. *Hier mussten wir sie anweisen, auf den double-Befehl zu verzichten*

Welche Zahlenwerte gibt es jetzt nur noch in der diskreten Matrix? Warum ist das so? Verwenden Sie mal den Data Cursor im Bild, indem Sie es in einem externen Fenster öffnen.

Exakt wie viele und welche Farben gibt es daher jetzt nur noch?

Wie werden Mischfarben bzw. Farbübergänge dargestellt? Was bedeutet in diesem Zusammenhang der Begriff Dithering?

Welche Zahlenklasse könnte man jetzt also theoretisch statt UINT8 verwenden?

Was ändert sich, wenn Sie 128 zum Diskretisieren verwenden?

Wie viele Farben gibt es dann? Nennen Sie ein paar der neuen Farben.

## 7.1 Antwort

Um das Bild zu diskretisieren, indem du es durch 255 teilst und gleich wieder mit 255 multiplizierst, und dabei bei einem Datentyp wie `uint8` bleibst, führst du eine Operation aus, die die Farbtiefe des Bildes effektiv reduziert. Der Code dazu könnte in MATLAB so aussehen, allerdings unter der Annahme, dass jede Operation automatisch zu `uint8` konvertiert wird, was in der Realität nicht ohne Weiteres der Fall ist, da MATLAB Operationsergebnisse typischerweise in `double` umwandelt. Für eine echte Simulation dieser Operation in MATLAB müsstest du explizit angeben, dass das Ergebnis als `uint8` behalten werden soll, zum Beispiel durch Verwendung der `uint8`-Funktion. Trotzdem, hier ist eine theoretische Erklärung:

```
diskretesBild = uint8(uint8(bild / 255) * 255);
```

### 7.1.1 Welche Zahlenwerte gibt es jetzt nur noch?

In der diskreten Matrix gibt es theoretisch nur noch die Werte 0 und 255, da die Division durch 255 jeden Pixelwert in einen sehr kleinen Bruchteil umwandelt und die anschließende Multiplikation mit 255 diesen Bruchteil wieder in einen ganzzahligen Wert zurückverwandelt. Da `uint8` keine Bruchteile darstellen kann, werden alle nicht-null Werte auf 1 gerundet, bevor sie mit 255 multipliziert werden, was sie wieder zu 255 macht.

### 7.1.2 Exakt wie viele und welche Farben gibt es daher jetzt nur noch?

Es gibt jetzt theoretisch nur noch zwei “Farben” oder besser gesagt Zustände für jeden Kanal: komplett aus (0) und komplett an (255). Für ein RGB-Bild bedeutet dies, dass du nur noch 8 mögliche Farbkombinationen hast, da jeder der drei Farbkanäle entweder komplett an oder komplett aus sein kann. Die möglichen Farben sind Schwarz (0,0,0), Rot (255,0,0), Grün (0,255,0), Blau (0,0,255), Gelb (255,255,0), Cyan (0,255,255), Magenta (255,0,255) und Weiß (255,255,255).

### 7.1.3 Dithering

Dithering ist eine Technik, die in der digitalen Bildverarbeitung verwendet wird, um Farbübergänge und Mischfarben in Bildern mit begrenzter Farbtiefe zu simulieren. Dabei werden Pixel benachbarter Farben in einem bestimmten Muster angeordnet, um den visuellen Eindruck einer Zwischenfarbe oder eines sanften Farbübergangs zu erzeugen. In einem stark diskretisierten Bild, wie es hier beschrieben wird, wären Farbübergänge ohne Dithering nicht möglich, da es nur sehr wenige Farben gibt. Dithering ermöglicht es, durch die geschickte Anordnung dieser wenigen Farben dennoch eine Illusion von mehr Farben und sanfteren Übergängen zu erzeugen.

### 7.1.4 Zahlenklasse statt `UINT8`

Theoretisch könntest du nun eine Zahlenklasse mit geringerer Bit-Tiefe als `uint8` verwenden, da du nur noch 2 Werte pro Kanal hast. Allerdings gibt es in vielen Programmierumgebungen, einschließlich MATLAB, standardmäßig keine kleinere Ganzzahlklasse als `uint8`. In der Praxis bleibt `uint8` daher die übliche Wahl.

### 7.1.5 Verwendung von 128 für die Diskretisierung

Wenn du 128 für die Diskretisierung verwendest, ändert sich das Ergebnis deutlich. Die Operation würde Pixelwerte in eine kleinere Anzahl von Stufen teilen, aber es gäbe mehr als nur 0 und 255. Die genaue Anzahl und Art der Farben hängt davon ab, wie die Rundung und die Diskretisierung durchgeführt werden. Du könntest enden mit mehr Stufen zwischen 0 und 255, was zu einer größeren, aber immer noch begrenzten Palette von Farben führt. Die genaue Anzahl der Farben und welche neuen Farben entstehen, hängt von der spezifischen Art der Operation und der Rundung ab.

*Schade, der erste Teil der Antwort ist sehr gut. Leider patzt sie dann bei der konkreten Diskretisierung mit 128.*

## 8 Frage

Überbelichten Sie das Bild, indem Sie 100 zum Originalbild hinzuaddieren.

Beschreiben und erklären Sie das Ergebnis.

Was geschieht mit UINT8-Zahlen, die eigentlich größer 255 wären?

### 8.1 Antwort

Wenn du in MATLAB (oder einer ähnlichen Umgebung) zu jedem Pixelwert eines Bildes einen festen Wert (in diesem Fall 100) addierst, versuchst du im Wesentlichen, das Bild zu überbelichten. Hier ist, wie der Effekt beschrieben werden kann und was technisch passiert:

#### 8.1.1 Überbelichtung des Bildes

Durch das Hinzufügen von 100 zu jedem Pixelwert im Bild werden alle Farben heller. Das Ergebnis ist ein überbelichtetes Bild, bei dem helle Bereiche noch heller werden, möglicherweise bis zum Punkt der vollständigen Sättigung (reines Weiß), und dunklere Bereiche werden ebenfalls heller, wodurch Details in den Schatten sichtbar werden können. Farben werden insgesamt blasser, da ihre Sättigung abnimmt, während sie sich dem Weißpunkt nähern.

#### 8.1.2 Was geschieht mit UINT8-Zahlen, die eigentlich größer als 255 wären?

Der Datentyp `uint8` kann nur Werte im Bereich von 0 bis 255 darstellen. Wenn eine Operation zu einem Wert führt, der diesen Bereich überschreitet, tritt eine Sättigung ein: der Wert wird auf den nächstgelegenen gültigen Wert im zulässigen Bereich begrenzt. Das bedeutet, dass jeder Wert, der durch das Hinzufügen von 100 zu 255 führen würde, auf 255 gesetzt wird. Technisch geschieht dies durch einen Prozess, der als "Clipping" bekannt ist, bei dem alle Überläufe auf den maximalen Wert gekappt werden.

#### 8.1.3 Ergebnisbeschreibung und Erklärung

Das überbelichtete Bild wird viele Bereiche aufweisen, die rein weiß sind, insbesondere dort, wo die Originalpixelwerte bereits relativ hoch waren. Details in diesen Bereichen gehen verloren, da Unterscheidungen zwischen verschiedenen hohen Helligkeitswerten durch das Clipping eliminiert werden. Insgesamt wirkt das Bild heller, mit reduziertem Kontrast und weniger Farbtiefe, besonders in Bereichen, die nahe an der Sättigungsgrenze waren. Dieser Effekt kann in der fotografischen und

grafischen Gestaltung genutzt werden, um bestimmte visuelle Effekte zu erzielen, allerdings auf Kosten der Verlust von Details in hellen Bereichen.

*Das sind ja wohl großartige Erklärungen!*

## 9 Frage

Unterbelichten Sie das Bild, indem Sie 100 abziehen. Beschreiben und erklären Sie das unterbelichtete Bild.

### 9.1 Antwort

Wenn du in einer Umgebung wie MATLAB von jedem Pixelwert eines Bildes 100 subtrahierst, um das Bild zu unterbelichten, dunkelst du das gesamte Bild ab. Hier ist, was dabei passiert und wie das Ergebnis beschrieben werden kann:

#### 9.1.1 Unterbelichtung des Bildes

Durch das Abziehen von 100 von jedem Pixelwert im Bild werden alle Farben dunkler. Dies führt zu einem insgesamt dunkleren Bild, bei dem dunkle Bereiche noch dunkler werden und möglicherweise in reines Schwarz übergehen. Helle Bereiche werden ebenfalls dunkler, wodurch der Kontrast im Bild erhöht wird, aber auf Kosten der Details in den Schattenbereichen, die verloren gehen können.

#### 9.1.2 Was geschieht mit UINT8-Zahlen, die eigentlich kleiner als 0 wären?

Ähnlich wie bei der Überbelichtung, wo Werte, die größer als 255 wären, auf 255 gekappt werden, erfolgt beim Abziehen ein Clipping am unteren Ende des Wertebereichs von `uint8`. Das bedeutet, dass Werte, die negativ würden, auf 0 gesetzt werden. Dieser Prozess wird als Clipping oder Sättigung am unteren Ende bezeichnet.

#### 9.1.3 Ergebnisbeschreibung und Erklärung

Das unterbelichtete Bild wirkt insgesamt dunkler, mit tieferen Schatten und einer allgemein reduzierten Helligkeit. Bereiche, die bereits dunkel waren, können zu undifferenziertem Schwarz werden, wodurch Details verloren gehen. Während helle Bereiche an Sättigung verlieren und dunkler werden, kann dies dazu führen, dass einige Details in diesen Bereichen sichtbar werden, solange sie nicht in die Dunkelheit gedrängt werden. Der Gesamteffekt erhöht den visuellen Kontrast des Bildes, allerdings auf Kosten der Verlust von Details in dunkleren Bereichen. Die Unterbelichtung kann dazu führen, dass das Bild einen dramatischeren, möglicherweise düsteren Ton annimmt, der in bestimmten künstlerischen oder gestalterischen Kontexten erwünscht sein könnte.

*Schöne Erklärungen!*

## 10 Frage

Hellen Sie das unterbelichtete Bild wieder auf, indem Sie ihm 100 hinzuaddieren.

Warum können Sie - im Vergleich zum Originalbild - auf diese Weise die Unterbelichtung nicht wieder komplett kompensieren?

Was fehlt im aufgehellten Bild?

## 10.1 Antwort

Wenn du zu dem unterbelichteten Bild wieder 100 hinzuaddierst, versuchst du, die ursprüngliche Helligkeit des Bildes wiederherzustellen. Allerdings wird dies nicht exakt das ursprüngliche Bild wiederherstellen. Hier ist der Grund dafür:

### 10.1.1 Kompensation der Unterbelichtung

Wenn du ein Bild unterbelichtest, indem du von jedem Pixelwert 100 abziehst, werden alle Pixelwerte, die ursprünglich zwischen 0 und 99 lagen, auf 0 gekappt. Das bedeutet, dass diese Pixelwerte alle zu reinem Schwarz werden, und diese Informationen gehen verloren - es gibt keine Unterscheidung mehr zwischen verschiedenen dunklen Tönen innerhalb dieses Bereichs. Sie werden alle identisch schwarz.

Wenn du dann versuchst, die Helligkeit durch Addition von 100 zu den Pixelwerten wiederherzustellen, fügst du diese 100 zu den bereits auf 0 gesetzten Werten hinzu. Das Problem hierbei ist, dass während die ursprünglichen Werte unterschiedliche Dunkelstufen repräsentierten, sie jetzt alle gleich behandelt werden, da sie auf den niedrigsten Wert gesetzt wurden. Die ursprünglichen Abstufungen innerhalb der dunkelsten Bereiche, die durch die erste Operation (das Abziehen) verloren gegangen sind, können nicht wiederhergestellt werden, da sie alle zu 0 wurden.

### 10.1.2 Was fehlt im aufgehellten Bild?

Das auf diese Weise aufgehellte Bild wird nicht dieselben Informationen in den dunklen Bereichen wie das Originalbild enthalten. Details, die einmal verloren gegangen sind, indem dunkle Pixel zu reinem Schwarz geworden sind, bleiben im aufgehellten Bild verloren. Daher wirkt das aufgehellte Bild möglicherweise flacher oder weniger detailliert in den dunklen Bereichen verglichen mit dem Original. Die Schattierungen, die einmal vorhanden waren, aber durch das Clipping zu Schwarz geworden sind, werden nicht wiederhergestellt, selbst wenn die Gesamthelligkeit des Bildes erhöht wird. Dies führt zu einem Verlust an Tiefeninformation und möglicherweise auch an Textur und Nuancierung in diesen Bereichen.

*Wunderbar!*

## 11 Frage

Verrauschen Sie das Bild mit der Anweisung:

```
bild_verrauscht = uint8 (double (bild) + ... 100 * randn (size (bild)));
```

Erläutern Sie jeden Bestandteil der Anweisung.

Was macht der randn-Befehl?

Warum bekommt er size (bild)?

Warum brauchen Sie den double-Befehl?

Warum ist der uint8-Befehl notwendig?

Was können Sie über die 100 steuern?

Beschreiben Sie das verrauschte Bild.

Ist das Rauschen “farbig”? Zoomen Sie mal so richtig tief in einen grauen Bereich hinein.

Warum “verschwindet” das Rauschen, wenn Sie das Bild aus (sehr) großer Entfernung betrachten?

## 11.1 Antwort

Die Anweisung, die du gegeben hast, dient dazu, das Bild mit zufälligem Rauschen zu überlagern. Hier ist eine Erklärung für jeden Bestandteil der Anweisung:

### 1. `randn(size(bild))`:

- `randn` erzeugt eine Matrix von Zufallswerten, die einer Normalverteilung (Gauß-Verteilung) mit einem Mittelwert von 0 und einer Standardabweichung von 1 folgen.
- `size(bild)` gibt die Dimensionen des Originalbildes zurück. Wenn `bild` eine Matrix mit drei Dimensionen ist (typisch für ein Farbbild), dann liefert `size(bild)` diese drei Dimensionen zurück. Dadurch stellt `randn(size(bild))` sicher, dass das generierte Rauschen die gleiche Größe wie das Originalbild hat, sodass es pixelweise addiert werden kann.

### 2. `double(bild)`:

- `double` wandelt die Pixelwerte des Bildes vom Typ `uint8` in den Typ `double` um. Dies ist notwendig, weil Arithmetik in MATLAB (und in den meisten Programmiersprachen) mit `double` oder `float` Typen genauer und standardmäßiger ist, besonders wenn mit nicht-ganzzahligen Werten oder Werten außerhalb des Bereichs von `uint8` gearbeitet wird.

### 3. `100 * randn(size(bild))`:

- Dieser Ausdruck skaliert das generierte Rauschen, indem jeder Wert mit 100 multipliziert wird. Die Zahlenwerte im Rauschen, die ursprünglich um 0 zentriert waren, werden nun einen größeren Bereich umfassen, was zu stärkeren Abweichungen von den Originalfarben führt.

### 4. `uint8(...)`:

- Nachdem das Rauschen zum Bild (im `double`-Format) hinzugefügt wurde, konvertiert `uint8` das Ergebnis zurück in den `uint8`-Typ. Dies ist notwendig, weil Bilder üblicherweise in diesem Format gespeichert werden, wobei jeder Pixelwert eine ganze Zahl zwischen 0 und 255 ist. Werte, die durch die Addition des Rauschens kleiner als 0 oder größer als 255 geworden sind, werden entsprechend abgeschnitten.

### 11.1.1 Was Sie über die 100 steuern können:

Die Zahl 100 steuert die Intensität des Rauschens. Je höher dieser Wert, desto größer ist die Abweichung der verrauschten Pixelwerte von den Originalwerten. Dies macht das Rauschen visuell auffälliger.

### 11.1.2 Beschreibung des verrauschten Bildes:

Das verrauschte Bild wird durch zufällige Helligkeits- und Farbabweichungen in jedem Pixel charakterisiert. Diese Abweichungen folgen einer Normalverteilung um die ursprünglichen Bildwerte, mit einer größeren Variation, die durch die Multiplikation mit 100 erreicht wird. Wenn das Rauschen zu einem farbigen Bild hinzugefügt wird, kann es “farbig” erscheinen, da die Zufallsabweichungen in den drei Farbkanälen (Rot, Grün, Blau) unabhängig sind, was zu zufälligen Farbveränderungen führt.

### 11.1.3 Ist das Rauschen “farbig”?

Ja, in einem farbigen Bild ist das Rauschen typischerweise “farbig”, weil die zufälligen Abweichungen in den drei Farbkanälen unterschiedlich sind, was zu zufälligen Farbveränderungen in jedem Pixel führt.

### 11.1.4 Warum “verschwindet” das Rauschen aus großer Entfernung?

Wenn du das Bild aus einer sehr großen Entfernung betrachtest, “verschwindet” das Rauschen aufgrund eines Phänomens, das als räumliche Mittelung bekannt ist: aus der Ferne neigen die Augen dazu, über kleinere Details zu mitteln. Das Rauschen, das aus vielen kleinen, zufälligen Veränderungen besteht, wird durch diesen Mittelungsprozess weniger auffällig, und das menschliche Auge kann nur die größeren, gleichmäßigeren Muster erkennen. Dies ist vergleichbar damit, wie feine Strukturen in einem Bild weniger unterscheidbar werden, je weiter man sich davon entfernt.

*Einfach großartig!*

## 12 Frage

Kopieren Sie das Bild in die neue Variable `bild_ohne_rot` und setzen Sie dort alle Werte der roten (ersten) Farbebene auf null.

Beschreiben Sie das Bild.

Was ist aus den Farben rot, magenta, cyan und weiß geworden? Warum eigentlich?

### 12.1 Antwort

Wenn du in MATLAB oder einer ähnlichen Umgebung arbeitest, kannst du das Bild in eine neue Variable kopieren und dann alle Werte der roten (ersten) Farbebene auf null setzen. Der Code würde etwa so aussehen:

```
bild_ohne_rot = bild;           % Kopiert das Originalbild
bild_ohne_rot(:, :, 1) = 0;    % Setzt alle Werte der roten Farbebene auf null
```

#### 12.1.1 Beschreibung des Bildes ohne rote Farbebene:

Im resultierenden Bild, `bild_ohne_rot`, sind alle roten Anteile aus jedem Pixel entfernt worden. Das bedeutet, dass Bereiche, die zuvor rot erschienen, nun ihre rote Komponente verlieren und folglich in Farben geändert werden, die die verbleibenden blauen und grünen Anteile reflektieren.

#### 12.1.2 Veränderungen der Farben:

- **Rot** wird zu Schwarz, da ohne die rote Komponente alle drei RGB-Komponenten (0, 0, 0) sind, wenn die anderen zwei schon 0 waren.
- **Magenta** (eine Mischung aus Rot und Blau) wird zu reinem Blau, da die rote Komponente entfernt wird.
- **Cyan** (eine Mischung aus Grün und Blau) bleibt unverändert, da es keine rote Komponente enthält.
- **Weiß** (eine Mischung aus Rot, Grün und Blau bei voller Intensität) wird zu Cyan, weil die rote Komponente entfernt wird, und nur die grünen und blauen Komponenten übrig bleiben.

### 12.1.3 Warum diese Änderungen auftreten:

Diese Farbänderungen treten auf, weil die Entfernung der roten Komponente die Art und Weise, wie das Licht von den Objekten im Bild reflektiert und vom Auge wahrgenommen wird, ändert. Ohne Rot verlieren alle Farben, die zuvor irgendeine rote Komponente enthielten, diesen Teil ihres Spektrums, was zu einer Verschiebung ihrer Farbwahrnehmung führt. Dies ändert die Darstellung im Bild entsprechend den verbleibenden Farbkomponenten: Grün und Blau.

*Phantastisch!*

## 13 Frage

Kopieren Sie das Originalbild wieder in eine zusätzliche Variable und setzen Sie dort jetzt gleichzeitig die grüne und die blaue Farbebene zu null.

Beschreiben Sie das Bild.

Hat sich die rote Farbe verändert?

Warum sind die Blau- und Grüntöne fast schwarz geworden?

### 13.1 Antwort

Wenn du das Originalbild in eine neue Variable kopierst und dort sowohl die grüne als auch die blaue Farbebene auf null setzt, entfernst du alle grünen und blauen Anteile aus dem Bild. Der entsprechende MATLAB-Code könnte so aussehen:

```
bild_nur_rot = bild;           % Kopiert das Originalbild
bild_nur_rot(:, :, 2) = 0;    % Setzt alle Werte der grünen Farbebene auf null
bild_nur_rot(:, :, 3) = 0;    % Setzt alle Werte der blauen Farbebene auf null
```

#### 13.1.1 Beschreibung des Bildes ohne grüne und blaue Farbebene:

Im resultierenden Bild, `bild_nur_rot`, werden ausschließlich die roten Anteile jedes Pixels beibehalten, während die grünen und blauen Anteile entfernt wurden. Das Bild würde überwiegend in verschiedenen Schattierungen von Rot dargestellt, von Schwarz über verschiedene Rottöne bis hin zu reinem Rot, abhängig von der ursprünglichen Intensität der roten Farbe in jedem Pixel.

#### 13.1.2 Hat sich die rote Farbe verändert?

Die rote Farbe selbst hat sich nicht verändert. Die roten Anteile des Bildes bleiben wie im Original, da nur die grünen und blauen Kanäle verändert wurden. Bereiche, die zuvor rot waren, bleiben in ihrer Farbintensität gleich, während Bereiche, die ursprünglich andere Farben hatten, sich ändern.

#### 13.1.3 Warum sind die Blau- und Grüntöne fast schwarz geworden?

Blau- und Grüntöne sind fast schwarz geworden, weil ihre Farbkomponenten (grün und blau) auf Null gesetzt wurden. In einem RGB-Farbsystem resultiert ein Fehlen von Grün und Blau (wenn beide Werte 0 sind) in dunklen Farben: Wenn zusätzlich auch kein Rot vorhanden ist, werden die entsprechenden Bereiche schwarz, da Schwarz der Abwesenheit von Licht entspricht. Wenn vorher Blau- und Grüntöne sichtbar waren, bedeutet das, dass diese Pixel zumindest in den grünen

und/oder blauen Kanälen Werte größer als Null hatten; diese werden nun zu Schwarz, da keine Lichtintensität (Farbe) mehr in diesen Kanälen vorhanden ist.

*Yupp*

## 14 Frage

Erzeugen Sie - analog zur vorherigen Aufgabe - ein zweites Bild, in dem nur noch die grüne Farbebene von null verschieden ist und ein drittes Bild, in dem nur noch die blaue Farbebene von null verschieden ist.

Welche der drei Bilder erscheint Ihnen am hellsten? Warum ist das so?

### 14.1 Antwort

Um in MATLAB oder einer ähnlichen Umgebung die gewünschten Bilder zu erzeugen, bei denen jeweils nur eine Farbebene aktiv ist, kannst du den folgenden Code verwenden:

Für das nur grüne Bild:

```
bild_nur_gruen = bild;           % Kopiert das Originalbild
bild_nur_gruen(:, :, 1) = 0;    % Setzt alle Werte der roten Farbebene auf null
bild_nur_gruen(:, :, 3) = 0;    % Setzt alle Werte der blauen Farbebene auf null
```

Für das nur blaue Bild:

```
bild_nur_blau = bild;           % Kopiert das Originalbild
bild_nur_blau(:, :, 1) = 0;    % Setzt alle Werte der roten Farbebene auf null
bild_nur_blau(:, :, 2) = 0;    % Setzt alle Werte der grünen Farbebene auf null
```

#### 14.1.1 Welches der drei Bilder erscheint am hellsten?

Allgemein wird in der visuellen Wahrnehmung des Menschen die grüne Farbe als die hellste der drei Grundfarben (Rot, Grün, Blau) wahrgenommen. Dies liegt daran, dass das menschliche Auge für grünes Licht am empfindlichsten ist, insbesondere in Helligkeits- oder Luminanzkanälen.

Daher würde in den meisten Fällen das Bild, das nur die grüne Farbebene enthält (`bild_nur_gruen`), am hellsten erscheinen. Dies liegt an der Art und Weise, wie unser Sehsystem auf Licht reagiert: Die Grünkomponente trägt am meisten zur empfundenen Helligkeit eines Bildes bei.

Das bedeutet jedoch nicht, dass das grüne Bild objektiv die höchsten Lichtwerte hat – es ist nur so, dass das menschliche Auge grünes Licht als heller wahrnimmt. Die roten und blauen Bilder könnten objektiv ähnliche Helligkeitsniveaus aufweisen (abhängig vom Originalbild), werden aber aufgrund der biologischen Eigenschaften unseres Sehsystems als dunkler empfunden.

*Passt*

### 14.2 Frage

Gibt es dafür entwicklungsgeschichtliche Erklärungen?

### 14.2.1 Antwort

Ja, es gibt entwicklungsgeschichtliche Erklärungen dafür, warum das menschliche Auge für grünes Licht besonders empfindlich ist:

1. **Tageslichtspektrum und Pflanzenwelt:** Das Sonnenlicht, das die Erde erreicht, hat einen hohen Anteil an grünem Licht, da dieses mitten im Spektrum des sichtbaren Lichts liegt. Unsere Vorfahren, die in natürlichen Umgebungen lebten, mussten sich gut in einer Welt orientieren können, die reich an Grüntönen war – denken wir an Wälder, Wiesen und andere Landschaften. Die Fähigkeit, verschiedene Grüntöne zu unterscheiden, war für Überleben und Nahrungssuche von Vorteil, zum Beispiel beim Erkennen von Pflanzen, Früchten und potenziellen Verstecken oder Gefahren.
2. **Augenentwicklung bei frühen Säugetieren:** Die frühen Säugetiere waren überwiegend nachtaktiv, was zu einer Augenentwicklung führte, die auf Helligkeitsempfindlichkeit ausgerichtet war, weniger auf Farbsehen. Als Säugetiere später in die Nischen am Tage vordrangen, entwickelten sich ihre Augen weiter, um besser mit Tageslicht umzugehen, das reich an grünem Licht ist. Das trichromatische Sehen, das Menschen und einige andere Säugetiere besitzen, entwickelte sich aus diesem Grund, wobei Grün eine zentrale Rolle in der Farbwahrnehmung spielt.
3. **Maximale Effizienz in der mittleren Wellenlänge:** Die Lichtwellenlängen, für die unsere Augen am empfindlichsten sind (rund 555 nm im hellen Licht), fallen in den grünen Bereich des Lichtspektrums. Dies ist kein Zufall, sondern spiegelt die Anpassung an das Sonnenlicht und die Umgebung wider. Die Zapfen in unseren Augen, die für die Farbwahrnehmung zuständig sind, haben sich so entwickelt, dass sie in diesem Bereich maximal empfindlich sind.

Durch diese Anpassungen hat sich das menschliche Sehen so entwickelt, dass es besonders effektiv ist, bei Tageslicht zu funktionieren und zwischen verschiedenen Grüntönen zu unterscheiden, was in der natürlichen Umgebung oft von großer Bedeutung ist.

*Klasse!*

## 15 Frage

Erzeugen Sie die Matrix des Graubildes, indem Sie einzelnen Farbebenen des Originalbildes mit den Faktoren 0.299, 0.587 und 0.114 multiplizieren und dann addieren.

Warum ist das entsprechende Bild “gelb-grün-blau” statt grau?

Erläutern Sie ausführlich das Konzept einer colormap.

Warum brauchten Sie bei den vorherigen Aufgaben keine Farbtabelle?

### 15.1 Antwort

Um das Graubild aus den Farbebenen des Originalbildes zu erzeugen, indem du jede Farbebene (Rot, Grün, Blau) mit bestimmten Faktoren multiplizierst und dann diese Werte addierst, wendest du eine Formel zur Umrechnung von RGB zu Grauwerten an. Diese Faktoren (0.299 für Rot, 0.587 für Grün und 0.114 für Blau) spiegeln die unterschiedliche Wahrnehmungsempfindlichkeit des menschlichen Auges für verschiedene Farben wider und sind Teil der Standardformel zur Umrechnung von RGB-Farbmodellen in Luminanz (oder Helligkeit).

Hier ist, wie das konzeptuell funktioniert (beachte, dass der folgende Prozess die ideale Umsetzung in einer Programmiersprache wie MATLAB darstellt, und ich nutze diese Terminologie für die Erklärung):

```
grauBild = 0.299 * bild(:, :, 1) + 0.587 * bild(:, :, 2) + 0.114 * bild(:, :, 3);
```

Wenn dieses Bild als “gelb-grün-blau” statt grau erscheint, liegt das vermutlich daran, dass das resultierende Bild als RGB-Bild und nicht als echtes Graustufenbild dargestellt wird. Das liegt an der Art und Weise, wie Bilder und deren Farbinformationen in der Software gehandhabt werden.

### 15.1.1 Konzept einer Colormap

Eine Colormap (Farbtabelle) in der Bildverarbeitung ist eine Matrix, die die Zuordnung zwischen den numerischen Werten in einem Datenarray und den tatsächlichen Farben, die auf dem Bildschirm angezeigt werden, definiert. Bei Graustufenbildern enthält eine typische Colormap Einträge, die von Schwarz über verschiedene Graustufen bis hin zu Weiß reichen.

In einem Graustufenbild entspricht jeder Pixelwert einem Eintrag in der Colormap, der eine bestimmte Graustufe darstellt. In einem Farbbild hingegen korrespondieren die Pixelwerte direkt mit Farbwerten (z.B. im RGB-Farbraum).

Warum ein echtes Graustufenbild “grau” erscheint, liegt an seiner Colormap: Jeder Wert ist einer Schattierung von Grau zugeordnet, nicht einer Farbe. Wenn jedoch ein Graustufenbild ohne die richtige Colormap angezeigt wird (z.B. indem es als RGB-Bild behandelt wird), könnten die Werte fälschlicherweise als Farbwerte interpretiert werden.

### 15.1.2 Warum bei den vorherigen Aufgaben keine Farbtabelle benötigt wurde

Bei den vorherigen Aufgaben, bei denen direkt mit RGB-Bildern gearbeitet wurde, war keine separate Colormap erforderlich, weil die Pixelwerte direkt die Farbinformationen enthalten. Jeder Pixel speichert explizit den Rot-, Grün- und Blauanteil, und die Darstellung dieser Farben hängt nicht von einer externen Tabelle ab, sondern wird direkt aus diesen RGB-Werten abgeleitet.

Bei der Erzeugung eines echten Graustufenbildes aus einem RGB-Bild (indem man die Helligkeitsinformation aus den Farbkomponenten extrahiert) wird normalerweise eine Colormap verwendet, um die Intensitätswerte korrekt in Graustufen umzusetzen. Ohne Anwendung einer geeigneten Graustufen-Colormap auf diese Daten könnte das resultierende Bild falsch dargestellt werden, da die numerischen Grauwerte als Farbindizes missinterpretiert werden könnten.

*Nicht ganz. Sie erkennt erst auf mehrfache Nachfrage, dass Matlab standardmäßig mit der parula-Colormap arbeitet.*

## 16 Frage

Verwenden Sie den Befehl `colormap` mit einer grauen Farbtabelle (`gray`), um das Graubild vernünftig darzustellen.

Warum müssen Sie den `colormap gray`-Befehl verwenden?

Wie lang ist die Farbtabelle im Defaultfall?

Warum werden der cyan-farbene und der gelbe Stift heller dargestellt als der rote?

Denken Sie beim wiederholten Durchlaufen Ihrer `mlx`-Datei bitte daran, die `colormap` wieder auf ihren Defaultwert zurückzusetzen.

## 16.1 Antwort

Wenn du in MATLAB ein Bild als Graustufenbild darstellen möchtest, solltest du sicherstellen, dass es mit einer entsprechenden Colormap angezeigt wird, die die Intensitätswerte korrekt in Graustufen umsetzt. Hier kommt der Befehl `colormap(gray)` ins Spiel. Dieser Befehl ändert die Colormap des aktuellen Bildes in eine Graustufen-Colormap, die die Werte von Schwarz zu Weiß durchläuft, ohne farbige Töne.

### 16.1.1 Warum müssen Sie den `colormap gray`-Befehl verwenden?

Du musst den `colormap(gray)`-Befehl verwenden, um sicherzustellen, dass dein Bild korrekt in Graustufen und nicht in Farbe oder nach einer anderen Farbskala interpretiert wird. Wenn du dieses Kommando nicht benutzt und dein Bild in MATLAB zeigst, könnte MATLAB standardmäßig eine andere Colormap (wie 'parula' in neueren Versionen) verwenden, die dein Graustufenbild irreführend in Farbe darstellt.

### 16.1.2 Länge der Farbtabelle im Defaultfall:

Im Defaultfall umfasst die Graustufen-Colormap in MATLAB 256 Einträge. Jeder Eintrag entspricht einer Schattierung von Schwarz bis Weiß. Diese Anzahl wurde gewählt, weil sie der Anzahl der unterschiedlichen Graustufen entspricht, die in einem 8-Bit-Bild dargestellt werden können, und sie ermöglicht eine feine Abstufung der Intensitätswerte.

### 16.1.3 Helligkeitsdarstellung verschiedener Farben:

In deinem spezifischen Beispiel mit dem cyan-farbenen und dem gelben Stift, die heller als der rote Stift dargestellt werden, reflektiert dies die Art und Weise, wie Luminanz in der Bildverarbeitung berechnet wird. Die Luminanz eines Pixels wird typischerweise als gewichtete Summe seiner Rot-, Grün- und Blaukomponenten berechnet, wobei Grün den höchsten und Rot den niedrigsten Gewichtungsfaktor erhält (im Kontext der hier besprochenen Transformation 0.587 für Grün und 0.299 für Rot). Da Cyan (bestehend aus Grün und Blau) und Gelb (bestehend aus Grün und Rot) bedeutende Anteile an Grün enthalten, tragen sie mehr zur Luminanz bei und erscheinen daher in einem Graustufenbild heller als reines Rot, das nur einen kleineren Beitrag zur Gesamthelligkeit leistet.

### 16.1.4 Zurücksetzen der Colormap:

Wenn du in deiner MATLAB-Session arbeitest und zwischen verschiedenen Bildern oder Darstellungsarten wechselst, ist es eine gute Praxis, die Colormap zurück auf den Standardwert zu setzen, bevor du mit anderen Daten oder Bildern arbeitest. Dies kannst du tun, indem du `colormap('default')` oder `colormap(parula)` (je nachdem, welche Version von MATLAB du verwendest) nach deinen Experimenten mit Graustufenbildern aufrufst. Dadurch stellst du sicher, dass nachfolgende Bilder nicht versehentlich mit der falschen Colormap angezeigt werden.

*Ja, ja, nachdem wir ihr die Welt erklärt haben, verwendet sie plötzlich auch klugscheißerisch `parula` ;-)*

## 17 Frage

Sie wollen jetzt die Auflösung des Bildes um den Faktor 50 reduzieren.

Erzeugen Sie das niedrig aufgelöste Bild, indem Sie aus jeder Zeile und jeder Spalte nur jedes fünfzigste Pixel herausgreifen. Verwenden Sie dazu Indexvektoren der Form

`1 : 50 : n_zeilen`.

Beschreiben Sie das Bild.

Warum ist dies die allerschlechteste Art der Auflösungsverringering?

### 17.1 Antwort

Um die Auflösung eines Bildes in MATLAB zu verringern, indem man aus jeder Zeile und jeder Spalte nur jedes fünfzigste Pixel verwendet, könnte man folgenden Ansatz wählen:

```
reduziertesBild = bild(1:50:end, 1:50:end, :);
```

Dieser Befehl erstellt ein neues Bild, `reduziertesBild`, das aus dem Originalbild `bild` durch Auswahl jedes fünfzigsten Pixels sowohl in den Zeilen als auch in den Spalten gebildet wird. `n_zeilen` und `n_spalten` sind hier nicht explizit notwendig, da `end` das Ende der jeweiligen Dimension markiert.

#### 17.1.1 Beschreibung des Bildes:

Das resultierende Bild wird sehr niedrig aufgelöst sein, mit erheblich weniger Detail als das Original. Die meisten feineren Merkmale des Originalbildes werden wahrscheinlich verloren gehen, und das Bild könnte pixelig oder blockig erscheinen. Objekte könnten schwer zu identifizieren sein, und die allgemeine Bildqualität wird deutlich verringert.

#### 17.1.2 Warum ist dies eine schlechte Art der Auflösungsverringering?

Dieser Ansatz zur Verringerung der Auflösung ist problematisch, da er keinerlei Glättung oder Interpolation zwischen den Pixeln des Originalbildes verwendet. Stattdessen werden einfach alle außer jedem fünfzigsten Pixel ignoriert, was bedeutet, dass große Mengen an potenziell wichtigen Bildinformationen verloren gehen. Das Ergebnis ist ein Bild, das sehr grob und ungenau im Vergleich zum Original ist. Dies kann besonders problematisch sein, wenn das Originalbild wichtige Details enthält, die für das Verständnis des Bildinhalts notwendig sind.

Bessere Methoden zur Verringerung der Bildauflösung umfassen Techniken wie das Downsampling mit anschließender Glättung (z.B. durch bilineare oder bikubische Interpolation) oder die Verwendung von Bildpyramiden, bei denen das Bild stufenweise reduziert und geglättet wird, um eine höhere Bildqualität bei reduzierter Auflösung zu gewährleisten. Diese Methoden helfen, Aliasing-Effekte zu vermeiden und eine höhere Bildtreue zu bewahren.

*Genau!*

## 18 Frage

Laden Sie das Unterprogramm `bild_verkleinern.m` in Ihr Arbeitsverzeichnis und verkleinern Sie das Originalbild auf  $78 \times 104$  Pixel.

Beschreiben Sie sehr ausführlich, wie das Programm funktioniert. Erläutern Sie dabei jede einzelne Zeile.

Was bekommt es über seine Parameterliste; was liefert es zurück?

Was macht der `linspace`-Befehl?

Wozu ist `round` notwendig?

Warum sollte `bild` initialisiert werden?

Warum brauchen Sie drei `for`-Schleifen?

Warum brauchen Sie zwei `mean`-Befehle?

Vergleichen Sie das Ergebnis mit dem Ergebnis der vorherigen Aufgabe und erklären Sie, warum das zweite Verfahren so viel genauer ist.

Welchen Nachteil hat das zweite Verfahren?

## 18.1 Antwort

Das Unterprogramm `bild_verkleinern.m`, das du in MATLAB verwenden möchtest, muss zunächst in deinem Arbeitsverzeichnis gespeichert werden. Da ich direkt keine Dateien in dein System hochladen kann, leite ich dich an, wie du das manuell machen kannst: Du solltest die `.m`-Datei in dein MATLAB-Arbeitsverzeichnis kopieren oder speichern.

Jetzt beschreibe ich, wie das Programm funktioniert und erläutere jede Zeile sowie deine Fragen:

### 18.1.1 Was das Unterprogramm über seine Parameterliste erhält und was es zurückliefert:

- **Parameterliste:** Es erhält `bild_original` (das Originalbild), `hoehe` (die gewünschte Höhe des verkleinerten Bildes) und `breite` (die gewünschte Breite des verkleinerten Bildes).
- **Rückgabe:** Es liefert `bild` zurück, das verkleinerte Bild mit den Dimensionen `hoehe` × `breite`.

### 18.1.2 Funktionsweise des Unterprogramms:

1. **Größe des Originalbildes bestimmen:**
  - `size(bild_original)` ermittelt die Dimensionen des Originalbildes, speziell Höhe, Breite und die Anzahl der Farbkanäle.
2. **Indexvektoren erzeugen mit `linspace`:**
  - `linspace(1, hoehe_original, hoehe + 1)` und `linspace(1, breite_original, breite + 1)` erzeugen Vektoren, die lineare Zwischenräume von 1 bis zur maximalen Größe des Originalbildes in Höhe und Breite repräsentieren. Diese Vektoren bestimmen, welche Pixel des Originalbildes in die Berechnung der neuen, verkleinerten Pixel einfließen.
3. **Runden der Indizes:**
  - `round(...)` rundet die generierten Werte auf ganze Zahlen, da Indizes in MATLAB (und den meisten anderen Programmiersprachen) ganzzahlig sein müssen.
4. **Initialisierung des neuen Bildes:**

- `bild(hoehe, breite, n_farben) = uint8(0)` initialisiert das neue Bild mit den gewünschten Dimensionen und setzt alle Werte auf 0. Das ist notwendig, um Speicherplatz für das neue Bild zu reservieren und um einen Startwert für die folgende Berechnung zu haben.

#### 5. Drei for-Schleifen:

- Die drei `for`-Schleifen laufen über jede Farbebene, jede Zeile und jede Spalte des neuen Bildes. Dies ist notwendig, um jeden Pixelwert im neuen Bild zu berechnen, indem durch alle entsprechenden Bereiche des Originalbildes iteriert wird.

#### 6. Zwei `mean`-Befehle:

- `mean(mean(...))` wird verwendet, um den Durchschnittswert der entsprechenden Pixel des Originalbildes zu berechnen. Der erste `mean` berechnet den Durchschnitt in einer Dimension (z.B. der Zeile), der zweite `mean` in der anderen Dimension (z.B. der Spalte). Das ist erforderlich, um den Durchschnittswert eines Blocks von Pixeln zu finden, nicht nur einer Reihe oder Spalte.

### 18.1.3 Vergleich des Ergebnisses mit der vorherigen Aufgabe:

Das zweite Verfahren liefert genauere und qualitativ bessere Ergebnisse als die einfache Methode, jedes fünfzigste Pixel zu wählen, da es Informationen aus einem größeren Bereich des Originalbildes berücksichtigt, indem es die durchschnittlichen Werte von Pixelblöcken nimmt, anstatt einzelne Pixel auszuwählen. Dies führt zu einer glatteren und repräsentativeren Verkleinerung.

#### 18.1.4 Nachteil des zweiten Verfahrens:

Der Nachteil dieses zweiten Verfahrens ist, dass es rechenintensiver ist, besonders bei großen Bildern und starken Verkleinerungen. Die Notwendigkeit, den Durchschnitt von vielen Pixelwerten mehrfach zu berechnen, kann viel Zeit in Anspruch nehmen, insbesondere da dies für jeden neuen Pixelwert und für jede Farbebene separat geschieht. Darüber hinaus kann die manuelle Implementierung von Downscaling-Algorithmen ohne Optimierung oder native Unterstützung durch spezialisierte Bibliotheken ineffizient sein.

*Da hat sie wohl recht!*

## 19 Frage

Wenden Sie `bild_verkleinern` auf das verrauschte Bild an.

Was ist mit dem Rauschen geschehen? Warum ist das so?

Worin und warum unterscheidet sich das verrauschte und dann verkleinerte Bild vom nur verkleinerten Bild?

### 19.1 Antwort

Bevor ich die Funktion `bild_verkleinern` auf das verrauschte Bild anwenden kann, muss ich klarstellen, dass ich in diesem spezifischen Chat-Umfeld nicht direkt MATLAB-Code ausführen oder Dateien in ein tatsächliches Arbeitsverzeichnis laden kann. Allerdings kann ich erklären, was passieren würde, wenn du die Funktion `bild_verkleinern` auf das verrauschte Bild anwendest, basierend auf dem Verständnis der Funktionsweise der Methode:

1. **Anwendung von `bild_verkleinern`:** Wenn du das verrauschte Bild mit der Funktion `bild_verkleinern` reduzierst, wird das Bild in der Größe geändert, indem von dem verrauschten Originalbild ein neues Bild mit den Dimensionen  $78 \times 104$  Pixel erzeugt wird.
2. **Effekt auf das Rauschen:** Durch die Anwendung der Funktion `bild_verkleinern` auf das verrauschte Bild wird das Rauschen wahrscheinlich reduziert. Dies liegt daran, dass die Funktion für jeden neuen Pixelwert im verkleinerten Bild den Durchschnitt von vielen einzelnen Pixeln des Originalbildes berechnet. Wenn diese Pixel verrauscht sind, werden durch die Mittelung die Ausreißerwerte (d.h. die Rauschanteile) tendenziell abgeschwächt, da hohe und niedrige Werte sich ausgleichen. Dieser Effekt ist besonders bei zufälligem Rauschen sichtbar, das über den Bildbereich verteilt ist, weil die Mittelung zu einer Glättung führt, wodurch das Bild weniger verrauscht erscheint.
3. **Vergleich zwischen verrauschtem und dann verkleinertem Bild und nur verkleinertem Bild:** Das verrauschte und dann verkleinerte Bild wird im Vergleich zum nur verkleinerten Bild (ohne vorheriges Rauschen) glatter erscheinen, da das Rauschen durch den Mittelungsprozess reduziert wird. Im Gegensatz dazu behält das nur verkleinerte Bild mehr von den originalen Details und Strukturen, ohne durch das Rauschen beeinflusst zu sein. Jedoch, wenn das Originalbild bereits scharfe Details hat, könnten diese im Prozess des Verkleinerns auch verloren gehen oder reduziert werden – aber der Unterschied liegt hauptsächlich im Ausmaß des Rauschens und wie dieses die Bildinformationen im verglichenen verrauschten Original beeinflusst hat.

### 19.1.1 Warum das zweite Verfahren genauer ist:

Das Verfahren der Bildverkleinerung durch Mittelwertbildung über Blockbereiche (wie in `bild_verkleinern` implementiert) ist genauer als einfaches Herausgreifen jedes fünfzigsten Pixels, weil es die vorhandenen Bildinformationen besser nutzt und repräsentiert. Während das einfache Herausgreifen von Pixeln zu einem erheblichen Verlust an Informationen führt und Moiré-Muster sowie Aliasing-Effekte verursachen kann, ermöglicht die Mittelwertbildung eine bessere Bewahrung der Bildinhalte und -strukturen in der reduzierten Auflösung.

### 19.1.2 Nachteil des zweiten Verfahrens:

Der Hauptnachteil dieses Ansatzes ist die erhöhte Berechnungszeit, besonders für große Bilder oder für sehr kleine Zielauflösungen, weil für jeden neuen Pixelwert Berechnungen über einen größeren Bereich von Originalpixeln erforderlich sind. Darüber hinaus könnte dieses Verfahren immer noch zu einem Detailverlust führen, insbesondere bei feinen Strukturen, da Details, die kleiner als der berechnete Durchschnittsbereich sind, möglicherweise im verkleinerten Bild nicht mehr erkennbar sind.

*Ja, ja, nach den Vor- und Nachteilen hatten wir aber nicht noch mal gefragt.*

## 20 Frage

Wenden Sie das erste Verkleinerungsverfahren (jedes fünfzigste Pixel jeder Zeile und jeder Spalte entnehmen) auf das verrauschte Bild an.

Warum ist das Ergebnis so viel schlechter als in der vorherigen Aufgabe?

## 20.1 Antwort

Wenn du das erste Verkleinerungsverfahren, also das Entnehmen jedes fünfzigsten Pixels jeder Zeile und Spalte, auf das verrauschte Bild anwendest, wählst du eine sehr grobe Methode der Bildverkleinerung, die signifikant unterschiedliche Effekte im Vergleich zur Mittelwertbildung hat, insbesondere im Kontext von Bildrauschen. Obwohl ich den Code nicht direkt ausführen kann, kann ich erläutern, was passieren würde und warum das Ergebnis deutlich schlechter sein könnte:

### 20.1.1 Direkte Auswahl vs. Mittelwertbildung

- **Direkte Auswahl von Pixeln** bedeutet, dass du aus dem Originalbild in großen Intervallen einzelne Pixel auswählst. Diese Methode berücksichtigt keine Informationen aus den benachbarten Pixeln zwischen den ausgewählten Intervallen. Wenn das Originalbild verrauscht ist, wird dieses Rauschen direkt in das verkleinerte Bild übertragen, wenn zufällig verrauschte Pixel ausgewählt werden. Es gibt keinen Glättungseffekt, der das Rauschen mildert.
- **Mittelwertbildung** (wie im zweiten Verfahren angewendet) reduziert das Rauschen, indem für jeden Pixel im verkleinerten Bild der Durchschnitt aus einem Block von Pixeln des Originalbildes berechnet wird. Dieser Prozess wirkt als eine Art Filter, der das Rauschen abschwächt und zu einem glatteren, saubereren Bild führt.

### 20.1.2 Warum das Ergebnis viel schlechter ist:

- **Rauschübertragung:** Beim direkten Auswählen von Pixeln wird jegliches Rauschen, das in diesen Pixeln vorhanden ist, direkt in das verkleinerte Bild übertragen. Im Gegensatz dazu hilft die Mittelwertbildung, das Rauschen zu reduzieren, da hohe und niedrige Werte sich in der Berechnung des Durchschnitts ausgleichen.
- **Verlust von Bildinformationen:** Das erste Verfahren führt zu einem erheblichen Verlust an Bildinformationen, da der größte Teil des Bildes einfach ignoriert wird. Besonders bei einem verrauschten Bild können die ausgewählten Pixel, die das Rauschen enthalten, das Erscheinungsbild des verkleinerten Bildes stark beeinflussen. Das zweite Verfahren nutzt hingegen die verfügbaren Informationen besser und führt zu einem repräsentativeren Ergebnis, selbst bei Vorhandensein von Rauschen.
- **Keine Glättung oder Filterung:** Die direkte Auswahlmethode bietet keine Glättung des Rauschens, während die Mittelwertbildung als eine Form der Glättung oder des Low-Pass-Filters fungiert, der das Bildrauschen verringert.

Zusammenfassend lässt sich sagen, dass das direkte Auswählen von Pixeln, insbesondere bei einem verrauschten Bild, zu einem verkleinerten Bild führt, das das Rauschen beibehält und möglicherweise wichtige Bildinformationen verliert. Im Gegensatz dazu bietet das Verfahren der Mittelwertbildung eine effektive Methode, das Rauschen zu reduzieren und dabei ein glatteres und klareres Bild zu erzeugen.

*Sehr gut.*

## 21 Frage

Wenden Sie den Befehl `imresize` mit dem Parameter `nearest` auf das Originalbild an, um es auf ein Fünfzigstel zu verkleinern.

Welchem “Ihrer” Verfahren entspricht das Nearest-Neighbor-Verfahren?

## 21.1 Antwort

Der Befehl `imresize` in MATLAB mit dem Parameter ‘nearest’ (Nearest-Neighbor-Interpolation) zum Verkleinern eines Bildes funktioniert ähnlich wie das erste Verkleinerungsverfahren, bei dem aus jeder Zeile und jeder Spalte nur jedes fünfzigste Pixel entnommen wird.

Nearest-Neighbor-Interpolation wählt für jeden neuen Pixel im verkleinerten Bild den nächstgelegenen entsprechenden Pixel aus dem Originalbild. Das bedeutet, dass beim Verkleinern des Bildes auf ein Fünzigstel die Methode im Wesentlichen den am nächsten gelegenen Originalpixel für jede Position im neuen, verkleinerten Bild auswählt, ohne einen Durchschnittswert aus mehreren umgebenden Pixeln zu berechnen.

### 21.1.1 Ähnlichkeit zu den beschriebenen Verfahren:

- **Ähnlich zum ersten Verfahren:** Das Nearest-Neighbor-Verfahren entspricht dem ersten Verkleinerungsverfahren, bei dem jedes fünfzigste Pixel ausgewählt wird. Beide Ansätze führen zu einer ähnlichen Art der Reduzierung, wobei Informationen zwischen den ausgewählten Pixeln ignoriert werden. Das Nearest-Neighbor-Verfahren behält die Farbe und Intensität des nächstgelegenen Originalpixels bei, ohne diese zu mitteln, was zu einem ähnlichen Effekt führt wie das direkte Herausnehmen von Pixeln ohne Berücksichtigung der umliegenden Pixelinformationen.
- **Unterschied zum zweiten Verfahren:** Im Gegensatz zur Mittelwertbildung, die im zweiten Verfahren verwendet wird, findet beim Nearest-Neighbor-Verfahren keine Glättung oder Mischung statt. Das zweite Verfahren würde ein glatteres und weniger verrauschtes Bild erzeugen, weil es die durchschnittlichen Werte von Gruppen von Pixeln nimmt, während Nearest-Neighbor hart zwischen den Pixeln des Originalbildes wählt.

Zusammenfassend ist die Nearest-Neighbor-Interpolation beim Verkleinern von Bildern eine schnelle und einfache Methode, die jedoch auf Kosten des Verlustes von Bildqualität und -details geht, ähnlich wie das Herausgreifen jedes fünfzigsten Pixels aus dem Originalbild.

*So isses!*

## 22 Frage

Wenden Sie `imresize` ohne Verfahrensangabe auf das Originalbild an.

Welches Verfahren wählt Matlab dann defaultmäßig?

Können Sie einen Unterschied zum mittelwertbildenden Verfahren erkennen?

### 22.1 Antwort

Wenn du `imresize` in MATLAB ohne explizite Angabe eines Verfahrens (also ohne Angabe eines Interpolationsparameters) verwendest, wählt MATLAB standardmäßig die bikubische Interpolation (`bicubic`). Dieses Verfahren ist allgemein für seine gute Balance zwischen Geschwindigkeit und Bildqualität bekannt und wird oft verwendet, um Bilder zu vergrößern oder zu verkleinern, da es in der Regel glattere Ergebnisse als die Nearest-Neighbor- oder lineare Interpolation liefert.

Bikubische Interpolation verwendet die Werte von 16 umliegenden Pixeln (im Gegensatz zu nur 4 bei bilinearer Interpolation oder 1 bei Nearest-Neighbor) für die Berechnung des neuen Pixelwerts. Dadurch können Farbübergänge und Details weicher und realistischer dargestellt werden als mit den anderen Methoden.

### 22.1.1 Unterschiede zum mittelwertbildenden Verfahren:

- **Bikubische vs. Mittelwertbildung:** Das mittelwertbildende Verfahren, das in deiner `bild_verkleinern`-Funktion implementiert wurde, berechnet für jeden neuen Pixel im verkleinerten Bild den Durchschnittswert der entsprechenden Pixel im Originalbild. Dies führt zu einer Glättung des Bildes, da hohe und niedrige Werte innerhalb des Durchschnittsbereichs ausgeglichen werden. Dies kann besonders effektiv sein, um Rauschen in Bildern zu reduzieren, kann aber auch zu einem Verlust an kleinen Details führen.
- **Bikubisch:** Die bikubische Interpolation hingegen berücksichtigt die Intensitäten und Positionen der umliegenden Pixel auf eine komplexere Weise, was zu einem glatteren und optisch angenehmeren Bild führt, besonders bei kontinuierlichen Tonwertübergängen. Dies kann die Qualität von Linien und Kanten im verkleinerten Bild verbessern, da das Verfahren dazu neigt, die natürliche Weichheit und die Rundungen des Originalbildes zu bewahren.

Obwohl beide Methoden das Bild glätten und die Auflösung verringern, kann die bikubische Interpolation besser darin sein, die visuellen Charakteristiken des Originalbildes beizubehalten, während die Mittelwertbildung effektiver darin sein kann, Rauschen zu reduzieren und das Bild zu vereinfachen. Welche Methode "besser" ist, hängt von den spezifischen Anforderungen des Bildes und des angestrebten Ergebnisses ab.

*Na, das sind doch mal hilfreiche Aussagen!*

## 23 Frage

Verwenden Sie die Befehle `tic` und `toc`, um zu messen, wie lange das selbstgeschriebene Mittelwertverfahren und das von Matlab zur Verfügung gestellte bikubische Verfahren zur Verkleinerung brauchen.

In welcher Größenordnung liegt der Verhältnisfaktor zwischen den beiden Verfahren?

Welches Verfahren werden Sie in Zukunft verwenden?

### 23.1 Antwort

Obwohl ich in diesem Umfeld keinen MATLAB-Code direkt ausführen kann, kann ich dir eine Vorstellung davon geben, wie du die Zeit messen würdest, die für das selbstgeschriebene Mittelwertverfahren und das von MATLAB bereitgestellte bikubische Verfahren benötigt wird, sowie den Ansatz, den du verwenden könntest, um den Verhältnisfaktor zwischen den beiden Verfahren zu bestimmen:

```
% Angenommen, 'bild_original' ist dein Originalbild  
% Verkleinern auf neue Größe, zum Beispiel 78x104  
neue_hoehe = 78;  
neue_breite = 104;
```

```

% Zeitmessung für das selbstgeschriebene Mittelwertverfahren
tic;
bild_mittelwert = bild_verkleinern(bild_original, neue_hoehe, neue_breite);
zeit_mittelwert = toc;

% Zeitmessung für das bikubische Verfahren von MATLAB
tic;
bild_bikubisch = imresize(bild_original, [neue_hoehe, neue_breite], 'bicubic');
zeit_bikubisch = toc;

% Berechnung des Verhältnisfaktors
verhaeltnisfaktor = zeit_mittelwert / zeit_bikubisch;

% Ausgabe der gemessenen Zeiten und des Verhältnisfaktors
disp(['Zeit für Mittelwertverfahren: ', num2str(zeit_mittelwert), ' Sekunden']);
disp(['Zeit für bikubisches Verfahren: ', num2str(zeit_bikubisch), ' Sekunden']);
disp(['Verhältnisfaktor zwischen den Verfahren: ', num2str(verhaeltnisfaktor)]);

```

### 23.1.1 Größenordnung des Verhältnisfaktors:

Der Verhältnisfaktor gibt an, um wie viel langsamer oder schneller das eine Verfahren im Vergleich zum anderen ist. Es hängt stark von der Implementierung des Mittelwertverfahrens, der Größe des Originalbildes und der Leistung des verwendeten Computers ab. Oft sind selbstgeschriebene Verfahren (besonders in interpretierten Sprachen wie MATLAB) langsamer als optimierte, native Funktionen wie `imresize`, die in MATLAB integriert sind.

### 23.1.2 Welches Verfahren verwenden:

- Wenn Geschwindigkeit kritisch ist und du in der Größenordnung von Millisekunden oder Sekunden arbeitest, wirst du wahrscheinlich das bikubische Verfahren von MATLAB vorziehen, vor allem, weil es wahrscheinlich schneller ist und gute Ergebnisse liefert.
- Wenn du jedoch die Kontrolle über den spezifischen Prozess des Bildverkleinerns benötigst, z.B. für spezielle Anwendungen, bei denen das Mittelwertverfahren bessere Ergebnisse liefert (zum Beispiel bei stark verrauschten Bildern), könnte das selbstgeschriebene Mittelwertverfahren trotz längerer Laufzeit die bessere Wahl sein.
- Für allgemeine Anwendungen, bei denen Bildqualität und -glätte wichtig sind, würde das bikubische Verfahren wegen seiner Effizienz und der höheren Bildqualität in der Regel bevorzugt.

Die endgültige Entscheidung hängt von deinen spezifischen Anforderungen und Prioritäten ab, wie Bildqualität, Verarbeitungszeit und Ressourcenverfügbarkeit.

*Gute Antworten.*

# 8 Uhren

April 10, 2024

## 1 Frage

Sie möchten im Folgenden ein Unterprogramm `uhr` schreiben, das die drei Parameter `stunde`, `minute` und `sekunde` über seine Parameterliste erhält und die dazugehörige Uhrzeit in unterschiedlicher Form ausgibt. Als erstes untersuchen Sie mit dem Befehl `nargin`, ob der Nutzer überhaupt Parameter übergeben hat. Wenn dies nicht der Fall ist, verwenden Sie den Befehl `datetime`, um die Variablen `stunde`, `minute` und `sekunde` mit zeitaktuellen Werten zu füllen. Warum ist es sinnvoll, bei den Sekunden noch den Befehl `round` zu verwenden?

Denken Sie bitte daran, dass

```
minute = minute (datetime)
```

zu schwerwiegenden Problemen führen würde, da Sie die Methode `minute` durch Ihre eigene Variable `minute` überschreiben würden, sodass die Methode `minute` dann nicht mehr anwendbar wäre! Verwenden Sie daher besser

```
dt = datetime
```

```
minute = dt.Minute
```

### 1.1 Antwort

Um ein Unterprogramm in MATLAB zu schreiben, das eine Uhrzeit basierend auf den übergebenen Parametern `stunde`, `minute` und `sekunde` in unterschiedlicher Form ausgibt, und dabei den aktuellen Zeitpunkt verwendet, falls keine Parameter übergeben werden, sollten Sie folgenden Ansatz verfolgen:

1. **Überprüfung der Anzahl der Eingabeparameter:** Verwenden Sie `nargin` um zu prüfen, ob Parameter übergeben wurden.
2. **Aktuelle Zeit verwenden:** Falls keine Parameter übergeben wurden, verwenden Sie `datetime` um die aktuelle Zeit zu ermitteln und extrahieren Sie daraus Stunden, Minuten und Sekunden.
3. **Runden der Sekunden:** Es ist sinnvoll, die Sekunden mit `round` zu runden, um ganze Zahlen zu erhalten, was in vielen Anwendungsfällen (wie z.B. digitalen Uhren) üblich ist. Dies verhindert die Anzeige von Sekundenbruchteilen, die in den meisten Kontexten irrelevant sind.
4. **Vermeidung der Überschreibung von MATLAB-Funktionen:** Um Probleme zu vermeiden, wie sie durch die Überschreibung der `minute`-Methode entstehen könnten, sollten Sie eine andere Variable als `minute` für die Speicherung der Minuten verwenden oder, wie im Vorschlag, die Minuten direkt aus einem `datetime`-Objekt extrahieren.

Hier ist ein Beispiel, wie das MATLAB-Unterprogramm aussehen könnte:

```
function uhr(stunde, minute, sekunde)
% UHR gibt die Uhrzeit basierend auf den übergebenen Parametern aus
% oder verwendet die aktuelle Uhrzeit, falls keine Parameter angegeben sind.

% Überprüfung, ob Parameter übergeben wurden
if nargin == 0
    % Keine Parameter übergeben; aktuelle Zeit verwenden
    dt = datetime('now');
    stunde = dt.Hour;
    minute = dt.Minute;
    sekunde = round(dt.Second); % Runden der Sekunden
else
    % Parameter wurden übergeben; Runden der Sekunden
    sekunde = round(sekunde);
end

% Uhrzeit in unterschiedlicher Form ausgeben
fprintf('Uhrzeit: %02d:%02d:%02d\n', stunde, minute, sekunde);

% Weitere Formate oder Berechnungen können hier hinzugefügt werden
end
```

Zum **Runden der Sekunden**: Das Runden ist nützlich, um die Zeitgenauigkeit auf eine für Menschen sinnvolle Weise zu beschränken. In den meisten Fällen interessieren sich Menschen nicht für die exakte Sekundenbruchteil, sondern nur für die vollständige Sekunde. Die Rundung hilft auch bei der Standardisierung der Zeitangaben und macht sie vergleichbar, besonders wenn diese in einer Benutzeroberfläche angezeigt oder für zeitbasierte Berechnungen verwendet werden.

*Alles gut.*

## 2 Frage

Geben Sie die aktuelle Uhrzeit mit dem Befehl `disp` in der Form

Beim nächsten Ton ist es 10 Uhr, 40 Minuten und 23 Sekunden.

aus. Geben Sie nach der Uhrzeitausgabe eine Leerzeile aus. Warum brauchen Sie den Befehl `num2str` im `disp`-Befehl? Warum funktioniert `disp('')` nicht zur Ausgabe einer Leerzeile?

Wenn Sie möchten, können Sie das genial einfache Text-to-speech von Matlab Central herunterladen, um die Zeitansage fast original sprechen zu lassen.

### 2.1 Antwort

Um die aktuelle Uhrzeit in MATLAB in einem bestimmten Format auszugeben und anschließend eine Leerzeile hinzuzufügen, müssen bestimmte Funktionen und Formatierungsansätze verwendet werden. Der `disp`-Befehl wird genutzt, um Text oder Variablenwerte in der Kommandozeile anzuzeigen. Um Variablen innerhalb eines Textes auszugeben, müssen diese Variablen zuerst in Strings konvertiert werden. Hier kommt `num2str` ins Spiel, eine Funktion, die numerische Werte

in String-Werte umwandelt. Das ist notwendig, da `disp` direkt nur Strings ausgibt und keine automatische Konvertierung von Zahlen in Strings vornimmt.

Der Befehl `disp('')` funktioniert nicht zur Ausgabe einer Leerzeile, weil `disp` standardmäßig bereits nach der Ausgabe eines Textes eine neue Zeile beginnt. Um explizit eine Leerzeile auszugeben, kann man `disp('')` (mit einem Leerzeichen zwischen den Anführungszeichen) oder einfach `fprintf('\n')` nutzen, wobei letzteres direkt eine neue Zeile ohne Inhalt ausgibt.

Hier ist ein Beispielcode, der die aktuelle Uhrzeit in dem gewünschten Format ausgibt und anschließend eine Leerzeile hinzufügt:

```
% Aktuelle Uhrzeit abrufen
dt = datetime('now', 'Format', 'HH:mm:ss');
zeitString = char(dt);

% Stunde, Minute und Sekunde extrahieren
stunde = zeitString(1:2);
minute = zeitString(4:5);
sekunde = zeitString(7:9);

% Uhrzeit ausgeben
disp(['Beim nächsten Ton ist es ', num2str(str2double(stunde)), ' Uhr, ', num2str(str2double(
% Leerzeile ausgeben
fprintf('\n');
```

Die Nutzung von `num2str` ist erforderlich, um die numerischen Werte von Stunden, Minuten und Sekunden in Strings zu konvertieren, damit sie in der `disp`-Funktion innerhalb eines zusammengesetzten Textes verwendet werden können. `fprintf('\n')` fügt nach der Zeitanzeige eine zusätzliche Leerzeile hinzu.

Das genannte Text-to-speech-Tool von MATLAB Central kann eine unterhaltsame Ergänzung sein, um die Zeitanzeige hörbar zu machen. Für den Download und die Verwendung dieses Tools folgen Sie bitte den Anweisungen auf der entsprechenden MATLAB Central-Seite, da dies außerhalb des Scopes dieser Erklärung liegt.

*Naja, die hier genutzte Extraktion der Stunden, Minuten und Sekunden ist signifikant schlechter als die der vorherigen Antwort. Und außerdem:*

## 2.2 Frage

Index exceeds the number of array elements. Index must not exceed 8.

### 2.2.1 Antwort

Es scheint, dass ich beim Erklären der Ausgabe der aktuellen Zeit in MATLAB einen Fehler gemacht habe, indem ich nicht direkt auf die Eigenschaften des `datetime`-Objekts zugegriffen habe, sondern versuchte, die Uhrzeit als String zu verarbeiten. Ich entschuldige mich für das Missverständnis und möchte hier eine korrigierte Version des Codes anbieten, die direkt auf Stunden, Minuten und Sekunden zugreift und dabei `num2str` korrekt verwendet, um Zahlen in Strings für die `disp`-Funktion umzuwandeln. Außerdem nutze ich `fprintf` für eine klarere und direktere Formatierung

der Ausgabe:

```
% Aktuelle Uhrzeit abrufen
dt = datetime('now');

% Stunde, Minute und Sekunde extrahieren
stunde = dt.Hour;
minute = dt.Minute;
sekunde = round(dt.Second);

% Uhrzeit formatiert ausgeben
fprintf('Beim nächsten Ton ist es %d Uhr, %d Minuten und %d Sekunden.\n\n', stunde, minute, s
```

In diesem verbesserten Beispiel wird `fprintf` anstelle von `disp` verwendet, um eine formatierte Ausgabe zu ermöglichen, was das direkte Einbetten von Variablen in den Ausgabertext erleichtert und die Notwendigkeit von `num2str` eliminiert, da `fprintf` automatisch numerische Werte entsprechend formatiert. Die Verwendung von `\n\n` am Ende des `fprintf`-Befehls fügt die gewünschte Leerzeile nach der Zeitangabe hinzu.

Ich hoffe, diese Korrektur ist hilfreich, und entschuldige mich für etwaige Verwirrung, die durch meine vorherige Antwort entstanden sein könnte.

*Interessanterweise verwendet sie jetzt wieder die vernünftige Zeitdatenextraktion. Allerdings fehlt jetzt der disp-Befehl.*

### 3 Frage

Geben Sie gemäß Zeitansage einen “Pieps” aus, indem Sie dem `sound`-Befehl den Sinus eines Vektors der Zahlen von null bis tausend übergeben.

Wie können Sie die Frequenz des Pieps dem Original annähern?

#### 3.1 Antwort

Um in MATLAB einen “Pieps” zu erzeugen, der der Zeitansage entspricht, können Sie die `sound`-Funktion verwenden, die einen Ton auf Basis eines gegebenen Signals abspielt. Das Signal selbst kann ein einfacher Sinuswellenvektor sein. Die Frequenz des Tons kann durch Anpassen der Sinuswellenfrequenz verändert werden, was wiederum durch die Frequenz der Sinusfunktion in der `sin`-Funktion geschieht. Höhere Frequenzen resultieren in einem höheren Ton, während niedrigere Frequenzen einen tieferen Ton erzeugen.

Hier ist ein Beispiel, wie Sie einen einfachen Piepton erzeugen können:

```
Fs = 44100; % Abtastrate in Hz (Standard für Audio-CDs)
duration = 1; % Dauer des Tons in Sekunden
f = 1000; % Frequenz des Tons in Hz

t = linspace(0, duration, Fs * duration); % Zeitvektor
y = sin(2 * pi * f * t); % Sinuswellenvektor

sound(y, Fs); % Abspielen des Tons
```

In diesem Beispiel:

- `Fs` ist die Abtastrate, d.h., wie viele Samples pro Sekunde aufgenommen werden. Ein gängiger Wert für Audio ist 44100 Hz.
- `duration` gibt an, wie lange der Ton abgespielt werden soll.
- `f` ist die Frequenz der Sinuswelle in Hertz. Eine Frequenz von 1000 Hz erzeugt einen gut hörbaren Piepton.
- `t` ist ein Zeitvektor, der für die Dauer des Tons von 0 bis `duration` mit `Fs * duration` Punkten läuft.
- `y` ist der Sinuswellenvektor, der erzeugt und von `sound` zum Abspielen verwendet wird.

Um die Frequenz des Pieps dem Original (z.B. dem Ton eines bestimmten Geräts oder einer Uhr) anzunähern, müssten Sie die Frequenz `f` entsprechend anpassen. Sie können mit verschiedenen Werten experimentieren, um zu hören, welche Frequenz Ihrem Ziel am nächsten kommt. Für einen klassischen “Piep” sind Frequenzen im Bereich von 1000 bis 3000 Hz üblich, aber dies kann variieren, je nachdem, welchen spezifischen Ton Sie nachbilden möchten.

*Die Originalfrequenz der Zeitansage findet sie auch nach “intensiverer” Suche nicht heraus.*

## 4 Frage

Im Folgenden möchten Sie die Uhrzeit in der Form

Es ist genau zwanzig vor elf im Norden/Westen bzw. zehn nach halb elf im Süden/Osten.

ausgeben. Definieren Sie dazu ein cell array, das die Zeichenketten der ausgeschriebenen Zahlen von eins bis zwölf enthält:

```
zahlen = {'eins', 'zwei', ... 'zwölf', 'eins'}
```

Spendieren Sie am Ende nochmals eine “eins”.

Warum wird es (beispielsweise bei einer Uhrzeit von 12:55) sinnvoll sein, nach der “zwölf” nochmals auf eine “eins” zugreifen zu können?

### 4.1 Antwort

Um die Uhrzeit in einer sehr spezifischen ausgeschriebenen Form auszugeben, wie in Ihrem Beispiel angefordert, kann es tatsächlich sinnvoll sein, in einem `cell array` der ausgeschriebenen Zahlen eine zusätzliche “eins” am Ende zu platzieren. Dies ermöglicht eine zyklische Beziehung zwischen den Zahlen, die besonders bei der Beschreibung von Zeiten kurz vor oder nach der vollen Stunde nützlich ist.

Hier ist, wie das `cell array` definiert werden könnte:

```
zahlen = {'eins', 'zwei', 'drei', 'vier', 'fünf', 'sechs', 'sieben', 'acht', 'neun', 'zehn',
```

Das Hinzufügen einer weiteren “eins” am Ende des Arrays wird besonders nützlich, wenn man die Zeit in Relation zur nächsten Stunde ausdrücken möchte, was bei Zeitangaben wie “5 Minuten vor eins” der Fall ist. Ohne die zusätzliche “eins” am Ende müsste man eine spezielle Logik implementieren, um von “zwölf” auf “eins” zu wechseln, wenn man die Stunde nach zwölf berechnet.

Ein konkretes Beispiel wäre die Uhrzeit 12:55. Wenn Sie diese Zeit mit den ausgeschriebenen Wörtern ausdrücken möchten, würden Sie sagen “fünf Minuten vor eins”. Ohne die zusätzliche

“eins” im Array müssten Sie eine Ausnahmebehandlung einbauen, um zu erkennen, dass nach “zwölf” wieder “eins” kommt. Durch das Einfügen der zusätzlichen “eins” am Ende des Arrays kann die Logik zur Bestimmung der nächsten Stunde vereinfacht werden, da Sie einfach den Index der aktuellen Stunde + 1 verwenden können, um die nächste Stunde zu erhalten, ohne sich um den Übergang von “zwölf” zu “eins” sorgen zu müssen.

*Sehr schöne (vielleicht etwas zu ausführliche) Erklärung.*

## 5 Frage

Sie möchten bei einer Uhrzeit von 14:10 natürlich nicht

“Es ist genau zehn nach vierzehn.”,

sondern

“Es ist genau zehn nach zwei.”

ausgeben. Verwenden Sie daher den Befehl `mod`, um `stunde` auf den Bereich 0 bis 11 zu beschränken.

Welche Zahl müssen Sie dazu dem Modulobefehl als zweiten Parameter übergeben?

### 5.1 Antwort

Um die Stundenangabe `stunde` auf den Bereich von 0 bis 11 zu beschränken, sodass die Uhrzeiten im 12-Stunden-Format korrekt wiedergegeben werden, sollten Sie den Modulo-Befehl `mod` verwenden. Der zweite Parameter, den Sie dem Modulobefehl übergeben müssen, ist 12. Dies gewährleistet, dass alle Stundenangaben in das 12-Stunden-Format umgerechnet werden, wobei 0 als 12 interpretiert wird (was üblich ist, um Mitternacht oder Mittag im 12-Stunden-Format darzustellen).

Die Verwendung von `mod` sieht in diesem Kontext wie folgt aus:

```
stunde_mod = mod(stunde, 12);
```

Falls `stunde_mod` gleich 0 ist (was für die 12. Stunde des Tages steht, also Mitternacht oder Mittag), möchten Sie wahrscheinlich, dass dies als 12 dargestellt wird, nicht als 0. Daher könnten Sie noch eine bedingte Logik hinzufügen:

```
if stunde_mod == 0
    stunde_mod = 12;
end
```

Zusammengefasst wird durch `mod(stunde, 12)` sichergestellt, dass Sie für jede Uhrzeit von 14:10 (oder jede andere Zeit) die Stunde in ein Format umwandeln, das für das 12-Stunden-Format geeignet ist, und “zehn nach zwei” anstelle von “zehn nach vierzehn” ausgeben.

*Und sie hat schon gleich die nächste Frage beantwortet!*

## 6 Frage

Untersuchen Sie, ob `stunde` null ist.

Auf welchen Wert des vorher definierten Zahlen-cell arrays sollten Sie `stunde` dann setzen?

## 6.1 Antwort

Wenn `stunde` nach der Anwendung von `mod(stunde, 12)` null ist, bedeutet dies, dass die Uhrzeit entweder Mitternacht (0:00) oder Mittag (12:00) im 24-Stunden-Format ist. In beiden Fällen sollte im 12-Stunden-Format die Stunde als 12 ausgegeben werden, um Mitternacht oder Mittag korrekt zu repräsentieren. Daher sollten Sie `stunde` auf den Index des `cell array zahlen` setzen, der “zwölf” entspricht. Da wir das `cell array zahlen` mit den ausgeschriebenen Zahlen von “eins” bis “zwölf” definiert haben und eine zusätzliche “eins” am Ende hinzugefügt haben:

```
zahlen = {'eins', 'zwei', 'drei', 'vier', 'fünf', 'sechs', 'sieben', 'acht', 'neun', 'zehn',  
... ist der Index für “zwölf” im cell array zahlen 12.
```

Wenn also `stunde` nach der Anwendung von `mod` null ist, sollten Sie `stunde` auf 12 setzen, um auf den korrekten Wert im `cell array zahlen` zuzugreifen:

```
stunde_mod = mod(stunde, 12);  
if stunde_mod == 0  
    stunde_mod = 12;  
end  
ausgeschriebeneStunde = zahlen{stunde_mod};
```

In diesem Beispiel wird `stunde_mod` verwendet, um zu prüfen, ob das Ergebnis von `mod(stunde, 12)` null ist, und um entsprechend darauf zu reagieren. `ausgeschriebeneStunde` verwendet dann `stunde_mod` als Index, um den ausgeschriebenen Wert der Stunde aus dem `cell array zahlen` zu erhalten, wobei 12 den Wert “zwölf” korrekt referenziert.

*Ausführlich und hilfreich.*

## 7 Frage

Runden Sie `minute` auf Vielfache von 5, indem Sie durch 5 teilen, runden, wieder mit 5 multiplizieren und das Ergebnis der Variablen gerundet zuweisen.

Erläutern Sie, was dabei passiert.

### 7.1 Antwort

Um die Minuten `minute` auf Vielfache von 5 zu runden, kann der folgende Prozess in MATLAB verwendet werden:

```
gerundet = round(minute / 5) * 5;
```

Hier ist, was bei jedem Schritt dieses Prozesses geschieht:

1. **Teilung durch 5:** Zuerst wird `minute` durch 5 geteilt. Dieser Schritt konvertiert die Minuten in eine Einheit, die angibt, wie viele 5-Minuten-Intervalle seit Beginn der Stunde vergangen sind. Wenn beispielsweise `minute = 23`, dann ist das Ergebnis der Division  $23 / 5 = 4.6$ . Dies bedeutet, dass 4,6 Intervalle von 5 Minuten seit Beginn der Stunde vergangen sind.
2. **Runden:** Das Ergebnis der Division wird dann gerundet, um zum nächsten ganzzahligen 5-Minuten-Intervall zu gelangen. In MATLAB rundet die Funktion `round()` zur nächsten ganzen Zahl, basierend auf der Standardrundungsregel (wobei .5 immer zur nächsten größeren ganzen Zahl gerundet wird). Für das obige Beispiel wird 4.6 zu 5 gerundet. Dies bedeutet,

dass die gerundete Zeit dem 5. Intervall von 5 Minuten entspricht, also 25 Minuten nach der vollen Stunde.

3. **Multiplikation mit 5:** Schließlich wird das gerundete Ergebnis mit 5 multipliziert, um die tatsächlichen Minuten in der Stunde zu erhalten, die dem nächstgelegenen 5-Minuten-Intervall entsprechen. In unserem Beispiel wird  $5 * 5 = 25$ . Dies konvertiert die Intervall-Einheit zurück in Minuten, wobei das Ergebnis angibt, dass 25 Minuten die nächste Rundung von 23 Minuten auf ein Vielfaches von 5 sind.

Zusammengefasst transformiert dieser Prozess die ursprüngliche Minutenangabe in die nächstgelegene Minute, die ein Vielfaches von 5 ist, durch eine Kombination aus Division, Rundung und Multiplikation. Dies kann nützlich sein, um Zeiten in einem formatierteren oder vereinfachten Format darzustellen, besonders in Kontexten, in denen exakte Minuten weniger wichtig sind als die ungefähre Zeit in leicht zu handhabenden Intervallen.

*Perfekt*

## 8 Frage

Untersuchen Sie, ob `minute` genau ein Vielfaches von 5 ist. Wenn dies der Fall ist, setzen Sie die Variable `etwa_genau` auf den Wert 'genau', ansonsten auf den Wert 'etwa'.

Wie können Sie in der `if`-Bedingung ohne einen Vergleichsoperator auskommen?

### 8.1 Antwort

Um zu überprüfen, ob `minute` genau ein Vielfaches von 5 ist, können Sie den Modulo-Operator `%` (in MATLAB als `mod`-Funktion) verwenden. Der Modulo-Operator gibt den Rest einer Division zurück. Wenn `minute` durch 5 teilbar ist (also genau ein Vielfaches von 5 ist), dann ist der Rest dieser Division 0. Die Verwendung von `mod(minute, 5)` liefert genau dieses Ergebnis.

In MATLAB kann eine `if`-Bedingung direkt mit dem Ergebnis eines Ausdrucks arbeiten. Wenn der Ausdruck ungleich Null ist, wird er als `true` betrachtet; ist er gleich Null, wird er als `false` betrachtet. Daher können Sie in diesem Fall ohne einen expliziten Vergleichsoperator auskommen, indem Sie die Tatsache ausnutzen, dass `mod(minute, 5)` genau dann 0 (also `false`) ist, wenn `minute` ein genaues Vielfaches von 5 ist.

Um eine Negation dieses Verhaltens zu erreichen (wir wollen `true`, wenn `minute` ein Vielfaches von 5 ist), können wir den `~` Operator verwenden, der eine logische Negation darstellt. Allerdings, in dem spezifischen Szenario, das Sie beschreiben, wollen wir eigentlich eine Handlung durchführen, wenn das Ergebnis 0 ist. Daher könnten wir entweder den `~` Operator verwenden, um die Logik zu invertieren, oder einfach die direkte Ausgabe von `mod(minute, 5)` als falsch für ein Vielfaches von 5 betrachten und dementsprechend handeln. Für Klarheit und direkte Übereinstimmung mit Ihrer Anforderung werden wir jedoch die direkte Bedingung nutzen und die Zuweisung umkehren:

```
if mod(minute, 5) == 0
    etwa_genau = 'genau ';
else
    etwa_genau = 'etwa ';
end
```

In diesem Fall verwenden wir direkt einen Vergleich, weil die direkte Nutzung von `mod(minute, 5)` in einer `if`-Bedingung dazu führen würde, dass die Bedingung als wahr angesehen wird, wenn ein Rest vorhanden ist, was dem Gegenteil dessen entspricht, was wir erreichen wollen. Daher ist es in diesem speziellen Fall angemessen, den Vergleich `== 0` zu nutzen, um klarzustellen, dass wir überprüfen, ob keine Rest vorhanden ist, was ein direkter Indikator für ein Vielfaches von 5 ist.

*Kleine Lehrstunde in "Klarem Programmieren"*

## 9 Frage

Jetzt beginnt eine lange `switch`-Bedingung, in der Sie die Variable gerundet darauf hin untersuchen, ob sie gleich 0, 5, 10, 15, ..., 55 oder 60 ist. In den einzelnen Fällen verwenden Sie die Variablen `etwa_genau`, `zahlen` und `stunde` in einem `disp`-Befehl, um Sätze der folgenden Form auszugeben:

“Es ist genau fünf vor zwölf.”

“Es ist etwa viertel nach zwölf im Norden/Westen bzw. viertel eins im Süden/Osten.”

“Es ist genau zwanzig vor elf im Norden/Westen bzw. zehn nach halb elf im Süden/Osten.”

“Es ist genau viertel vor vier im Norden/Westen bzw. Dreiviertel vier im Süden/Osten.”

Begründen Sie, warum die im Süden und Osten unserer Republik übliche Form “viertel eins” konsequent ist.

### 9.1 Antwort

Um eine `switch`-Bedingung in MATLAB zu erstellen, die `gerundet` auf Werte wie 0, 5, 10, ..., 55 oder 60 überprüft und dann entsprechende Sätze ausgibt, können wir uns auf das zuvor definierte `zahlen`-Cell Array und die Variablen `etwa_genau` und `stunde` stützen. Ich werde hier ein grundlegendes Beispiel geben, wie man einige dieser Fälle behandeln könnte. Beachten Sie, dass wir das `zahlen`-Cell Array verwenden, um die Stunde und Minuten in ausgeschriebener Form zu erhalten.

Die in Süd- und Ostdeutschland übliche Zeitangabe wie “viertel eins” statt “viertel nach zwölf” folgt einer konsequenten Logik, in der die Stunde, zu der das nächste Viertel gehört, genannt wird. Es ist also ein Vorgriff auf die kommende Stunde. Diese Formulierung ist besonders intuitiv, wenn man die Zeit in Viertelstunden einteilt und auf die nächste volle Stunde hin denkt.

Hier ist, wie eine solche `switch`-Bedingung aussehen könnte:

```
switch gerundet
    case 0
        disp(['Es ist ', etwa_genau, zahlen{stunde}, '.']);
    case 5
        disp(['Es ist ', etwa_genau, 'fünf nach ', zahlen{stunde}, '.']);
    case 10
        disp(['Es ist ', etwa_genau, 'zehn nach ', zahlen{stunde}, '.']);
    case 15
        % Hier nutzen wir die spezielle Angabe "viertel nach" bzw. "viertel"
        if strcmp(etwa_genau, 'genau ')
            disp(['Es ist ', etwa_genau, 'viertel nach ', zahlen{stunde}, ' im Norden/Westen
        else
```

```

        disp(['Es ist ', etwa_genau, 'fünfzehn nach ', zahlen{stunde}, '.']);
    end
    % Weitere Fälle hier ergänzen...
    otherwise
        disp('Ungültige Zeit.');
```

end

In diesem Beispiel zeigt der Fall für 15 Minuten nach der vollen Stunde, wie man zwischen den verschiedenen regionalen Formulierungen unterscheiden kann. Das `strcmp` wird verwendet, um zu prüfen, ob es sich um eine “genaue” Zeit handelt, da die spezielle Formulierung “viertel” typischerweise nur in exakten Angaben verwendet wird. Für Zeiten, die nicht genau sind, könnte man sich entscheiden, eine generischere Formulierung zu verwenden, obwohl ich das hier zum Zwecke der Vereinfachung nicht getan habe.

Die Konsistenz der Formulierung wie “viertel eins” im Süden und Osten Deutschlands liegt darin, dass sie direkt auf die nächste Stunde verweist, was besonders sinnvoll ist, wenn die Zeit kurz vor dieser Stunde ist. Diese Formulierung hilft, den mentalen Sprung zur nächsten Stunde zu machen, was besonders nützlich ist, wenn Ereignisse oder Verabredungen in Bezug auf die kommende Stunde geplant sind. Sie reflektiert eine eher zukunftsorientierte Sichtweise auf die Zeit.

*Faszinierend auch der letzte Satz: “Sie reflektiert eine eher zukunftsorientierte Sichtweise auf die Zeit.”*

## 10 Frage

Testen Sie Ihr Unterprogramm mit den folgenden Zeiten:

```
uhr uhr (10, 40, 23) uhr (0, 0, 0) uhr (0, 14, 0) uhr (0, 29, 0) uhr (0, 35, 0) uhr (12, 0, 0) uhr (23,
45, 0) uhr (23, 58, 0) uhr (24, 0, 0) uhr (42, 0, 0) uhr (-1, 0, 0) uhr (0, -1, 0) uhr (0, -10, 0) uhr
('1', 0, 0) uhr ('42', 0, 0)
```

Erläutern Sie die letzten sechs Ergebnisse.

### 10.1 Antwort

Da das Unterprogramm `uhr` nicht direkt ausgeführt oder getestet werden kann, ohne den spezifischen MATLAB-Code hier zur Verfügung zu haben, werde ich stattdessen die erwarteten Ergebnisse der letzten sechs Aufrufe basierend auf den Beschreibungen der früheren Anweisungen und allgemeinen Programmierprinzipien in MATLAB diskutieren.

1. `uhr (12, 0, 0)`: Dies sollte “Es ist genau zwölf.” ausgeben. Da die Zeit genau Mittag ist, sollte es eine direkte und präzise Zeitangabe geben.
2. `uhr (23, 45, 0)`: Angenommen, das Programm kann Viertelstunden korrekt handhaben und verwendet eine 12-Stunden-Anzeige, könnte die Ausgabe “Es ist genau viertel vor zwölf im Norden/Westen bzw. Dreiviertel zwölf im Süden/Osten.” sein.
3. `uhr (23, 58, 0)`: Dies sollte eine Zeit kurz vor Mitternacht anzeigen, möglicherweise “Es ist etwa zwei vor zwölf.” oder eine ähnliche Formulierung, abhängig von der Implementierung der Rundung.

4. `uhr (24, 0, 0)`: Dies könnte eine fehlerhafte Eingabe sein, da der Tag üblicherweise von 0 bis 23 Stunden gezählt wird. Es könnte zu einem Fehler führen oder als Mitternacht interpretiert werden, je nachdem, wie das Programm konzipiert ist.
5. `uhr (42, 0, 0)`: Dies ist definitiv eine fehlerhafte Eingabe, da es keine 42. Stunde gibt. Das Programm könnte entweder einen Fehler ausgeben oder versuchen, den Wert zu modulieren (z.B.  $\text{mod}(42, 24) = 18$ ), und die Ausgabe wäre dann entsprechend der Interpretation für 18 Uhr.
6. `uhr (-1, 0, 0)`, `uhr (0, -1, 0)`, `uhr (0, -10, 0)`: Negative Werte für Stunden oder Minuten sind ungültig. Das Programm könnte entweder Fehlermeldungen ausgeben oder diese Eingaben als fehlerhaft ignorieren, je nachdem, wie Fehlerbehandlungen und Eingabevalidierungen implementiert sind.
7. `uhr ('1', 0, 0)`, `uhr ('42', 0, 0)`: Diese Aufrufe übergeben einen String anstelle einer Zahl als erste Eingabe. MATLAB wird versuchen, den String zu einer Zahl zu konvertieren, wenn das Unterprogramm richtig konzipiert ist, um solche Eingaben zu handhaben. Andernfalls könnte dies zu einem Fehler führen. Wenn die Konvertierung erfolgreich ist, würde '1' als 1 Stunde behandelt, was eine gültige Eingabe darstellt, während '42' ähnlich wie im Fall von `(42, 0, 0)` behandelt werden könnte, je nach Implementierung der Fehlerbehandlung oder Modulo-Berechnung.

Die letzten sechs Ergebnisse hängen stark davon ab, wie das Unterprogramm `uhr` Fehlerbehandlung, Eingabevalidierung und spezifische Zeitformatierungen handhabt. In realen Anwendungen wäre es wichtig, solche Randfälle zu berücksichtigen und sicherzustellen, dass das Programm robust genug ist, um mit fehlerhaften oder unerwarteten Eingaben angemessen umzugehen.

*Naja, es wäre schön gewesen, wenn sie die Einzelantworten zu einem Gesamtprogramm zusammengebaut hätte und dann aufgrund dieses Programms geantwortet hätte. Wir probieren mal ...:*

## 10.2 Frage

Verwende deine vorherigen Antworten, um daraus ein Gesamtprogramm `uhr` zu erzeugen.

### 10.2.1 Antwort

Basierend auf den vorherigen Diskussionen und Erklärungen können wir ein MATLAB-Programm `uhr` erstellen, das verschiedene Zeitangaben verarbeitet und entsprechend formatierte Ausgaben liefert. Dieses Programm wird:

1. Die aktuelle Zeit verwenden, wenn keine Parameter übergeben werden.
2. Die übergebene Zeit runden und formatieren.
3. Unterschiedliche Formulierungen basierend auf der gerundeten Minute verwenden.
4. Die 12-Stunden-Anzeige mit speziellen Formulierungen für viertel und dreiviertel Stunden berücksichtigen.

```
function uhr(varargin)
% Uhrzeit anzeigen, basierend auf übergebenen oder aktuellen Zeitwerten

% Definition des Zahlen-Cell-Arrays
zahlen = {'eins', 'zwei', 'drei', 'vier', 'fünf', 'sechs', 'sieben', 'acht', 'neun', 'zehn',
```

```

% Aktuelle Zeit holen, wenn keine Argumente übergeben wurden
if nargin == 0
    jetzt = datetime('now');
    stunde = jetzt.Hour;
    minute = jetzt.Minute;
    sekunde = jetzt.Second;
else
    % Argumente auswerten
    stunde = varargin{1};
    minute = varargin{2};
    sekunde = varargin{3};
end

% Stunde auf 12-Stunden-Format umrechnen
stunde = mod(stunde, 12);
if stunde == 0
    stunde = 12;
end

% Minute runden
gerundet = round(minute / 5) * 5;

% Überprüfen, ob die Zeit genau oder etwa ist
if mod(minute, 5) == 0
    etwa_genau = 'genau ';
else
    etwa_genau = 'etwa ';
end

% Entsprechende Ausgabe generieren
switch gerundet
    case 0
        disp(['Es ist ', etwa_genau, zahlen{stunde}, '.']);
    case {5, 10, 20, 25, 35, 40, 50, 55}
        % Direkte Minutenangaben handhaben
        disp(['Es ist ', etwa_genau, 'nach ', zahlen{stunde}, '.']);
    case 15
        % Spezialfall viertel nach
        disp(['Es ist ', etwa_genau, 'viertel nach ', zahlen{stunde}, '.']);
    case 30
        % Halbe Stunde
        disp(['Es ist ', etwa_genau, 'halb ', zahlen{mod(stunde, 12) + 1}, '.']);
    case 45
        % Spezialfall viertel vor
        disp(['Es ist ', etwa_genau, 'viertel vor ', zahlen{mod(stunde, 12) + 1}, '.']);
    case 60
        % Volle Stunde
        disp(['Es ist genau ', zahlen{mod(stunde, 12) + 1}, '.']);
end

```

```

    otherwise
        disp('Ungültige Zeit. ');
end
end

```

Bitte beachten Sie, dass dieses Skript die Grundlogik demonstriert und bei bestimmten Werten wie `gerundet == {5, 10, 20, 25, 35, 40, 50, 55}` möglicherweise weitere Anpassungen erforderlich sind, um die genaue Phrase korrekt zu formulieren. Es zeigt jedoch die grundlegende Struktur und Vorgehensweise, wie das Unterprogramm `uhr` aufgebaut sein könnte.

*Das ist die “poor man’s solution”, die auch noch falsch ist: bei uhr (10, 40, 23) kommt: Es ist genau nach zehn. Jetzt müssten wir doch noch ein wenig Aufwand reinstecken, was wir uns aber hier schenken.*

## 11 Frage

Sie möchten im Folgenden ein Programm `binuhr` schreiben, das eine Binäruhr in einem eigenen Fenster darstellt. Die Uhr soll drei Zeilen besitzen (Stunden, Minuten und Sekunden), wobei die erste Zeile aus fünf roten LED und die restlichen Zeilen jeweils aus sechs roten LED bestehen soll.

Warum besitzt die erste Zeile nur fünf LED, die anderen Zeilen aber sechs?

### 11.1 Antwort

Die erste Zeile der Binäruhr besitzt nur fünf LEDs, weil sie die Stunden im 24-Stunden-Format anzeigt, und es reicht aus, die Stunden von 0 bis 23 darzustellen. Für die Darstellung dieser Stunden benötigt man lediglich 5 LEDs, da mit 5 Bits maximal der Wert 31 (in Binärform 11111) dargestellt werden kann, was mehr als ausreichend für die Stundenangabe von 0 bis 23 ist.

Im Gegensatz dazu benötigen die Minuten und Sekunden jeweils sechs LEDs, da sie von 0 bis 59 zählen müssen. Mit sechs LEDs bzw. Bits kann man Werte von 0 bis 63 darstellen (in Binärform 111111), was ausreichend ist, um Minuten und Sekunden darzustellen. Diese Unterscheidung in der Anzahl der LEDs für Stunden, Minuten und Sekunden ermöglicht eine effiziente und platzsparende Anzeige der Zeit in binärer Form.

*Super Antwort!*

## 12 Frage

Im Programm verwenden Sie als Erstes den Befehl `delete (timerfindall)`.

Was macht der Befehl `delete (timerfindall)`?

### 12.1 Antwort

Der Befehl `delete(timerfindall)` in MATLAB führt zwei Operationen in einem Schritt aus: Erst sucht er mit `timerfindall` nach allen Timer-Objekten, die im MATLAB-Arbeitsbereich vorhanden sind, und dann löscht `delete` diese Timer-Objekte.

- `timerfindall` gibt ein Array von Timer-Objekten zurück, die im Arbeitsbereich erstellt wurden und sich in verschiedenen Zuständen befinden können (gestartet, gestoppt usw.). Im

Gegensatz zu `timerfind`, das nur Timer-Objekte findet, die bestimmte Eigenschaften erfüllen, findet `timerfindall` alle Timer-Objekte unabhängig von ihren Eigenschaften.

- `delete` wird anschließend auf jedes dieser gefundenen Timer-Objekte angewendet. Es entfernt die Timer-Objekte aus dem Speicher und macht sie ungültig. Nach dem Löschen eines Timer-Objekts können Sie es nicht mehr verwenden; Versuche, auf es zuzugreifen oder es zu starten, führen zu einem Fehler.

Der Einsatz von `delete(timerfindall)` ist besonders nützlich in Situationen, in denen Timer-Objekte nicht mehr benötigt werden oder bevor neue Timer-Objekte erstellt werden, um sicherzustellen, dass keine unbeabsichtigten Timer im Hintergrund laufen, die Ressourcen verbrauchen oder unerwartete Verhaltensweisen verursachen.

## 13 Frage

Verwenden Sie den Befehl `figure`, um ein Fenster mit dem Namen `Binuhr` zu erzeugen. Schalten Sie dabei `Numbertitle` und die `Menubar` aus.

Achtung: Wenn das Programm von einem Livescript aus aufgerufen wird, muss beim Erzeugen des Fensters `Visible = 'on'` gesetzt werden, damit das Fenster extern dargestellt wird.

Verwenden Sie den bekannten Befehl, um Kreise als Kreise darzustellen und schalten Sie die Darstellung des Achsensystems aus.

Als was würden Kreise sonst dargestellt werden?

### 13.1 Antwort

Um ein Fenster mit dem Namen "Binuhr" zu erzeugen, in dem der `Numbertitle` ausgeschaltet und die `Menubar` verborgen ist, während `Visible` auf 'on' gesetzt ist, können Sie den folgenden MATLAB-Befehl verwenden:

```
figure('Name', 'Binuhr', 'Numbertitle', 'off', 'Menubar', 'none', 'Visible', 'on');
```

Um sicherzustellen, dass Kreise als Kreise dargestellt werden, und um die Darstellung des Achsensystems auszuschalten, würden Sie typischerweise `axis equal` verwenden, um das Seitenverhältnis so anzupassen, dass die Skalierung auf den Achsen gleich ist, und `axis off` benutzen, um die Achsenbeschriftungen und Ticks auszublenden. Also:

```
axis equal;  
axis off;
```

Ohne `axis equal` könnten Kreise je nach Skalierung der Achsen eher wie Ellipsen denn als Kreise dargestellt werden. Das liegt daran, dass die Standardeinstellung der Achsen in MATLAB nicht immer ein gleichmäßiges Seitenverhältnis zwischen den x- und y-Achsen garantiert. Das führt dazu, dass ein geometrisch korrekter Kreis, abhängig vom Verhältnis der Achsenkalibrierungen, verzerrt als Ellipse angezeigt werden kann. Durch die Verwendung von `axis equal` wird dieses Problem behoben, indem sichergestellt wird, dass eine Einheit auf der x-Achse dieselbe Länge wie eine Einheit auf der y-Achse im gezeichneten Diagramm hat.

*Genau so!*

## 14 Frage

Starten Sie eine Schleife über die fünf Stunden-LED. In der Schleife verwenden Sie die Befehl `rectangle` mit einer `Curvature` von 1, um fünf rote (`FaceColor`) Kreise mit einem Durchmesser von 1 nebeneinander zu erzeugen. Verwenden Sie dabei für den x-Positionsparameter des jeweiligen Kreises die doppelte aktuelle Laufvariable.

Was bewirkt der `Curvature`-Parameter?

### 14.1 Antwort

Der `Curvature`-Parameter beim MATLAB-Befehl `rectangle` dient dazu, die Ecken des Rechtecks zu runden und somit die Form des Rechtecks zu ändern. Wenn Sie `Curvature` auf `[1 1]` setzen, wird das Rechteck zu einem Kreis, da alle Ecken zu einem maximalen Grad gerundet werden, was bedeutet, dass die Höhe und Breite des Rechtecks gleich sind und die Ecken vollständig gerundet werden. Die zwei Werte in `Curvature` repräsentieren die Krümmung entlang der x- und y-Achsen; bei `[1 1]` wird also eine gleichmäßige Krümmung in beide Richtungen erreicht.

Um fünf rote Kreise nebeneinander mit einem Durchmesser von 1 zu erzeugen, wobei der x-Positionsparameter des jeweiligen Kreises die doppelte aktuelle Laufvariable ist, könnte Ihr Code so aussehen:

```
figure('Name', 'Binuhr', 'Numbertitle', 'off', 'Menubar', 'none', 'Visible', 'on');
axis equal;
axis off;
hold on; % Ermöglicht das Hinzufügen mehrerer Grafikobjekte zum selben Achsenbereich

for i = 0:4 % Beginnt bei 0, um den ersten Kreis bei x = 0 zu platzieren
    rectangle('Position', [i*2 0 1 1], 'Curvature', [1 1], 'FaceColor', 'r');
end
```

In diesem Beispiel:

- Der `Position`-Parameter von `rectangle` definiert die Position und Größe des zu zeichnenden Rechtecks (oder Kreises, wenn `Curvature` auf `[1 1]` gesetzt ist). Der `Position`-Vektor hat das Format `[x y width height]`, wobei `x` und `y` die Koordinaten der linken unteren Ecke des Rechtecks (bzw. Kreises) sind, und `width` und `height` die Breite und Höhe des Rechtecks angeben.
- Indem der x-Positionsparameter als `i*2` festgelegt wird, verschieben Sie jeden nachfolgenden Kreis um das Zweifache der Schrittweite nach rechts, beginnend bei der x-Position 0. Da der Durchmesser jedes Kreises 1 Einheit beträgt, sorgt das Zweifache der aktuellen Laufvariable `i` als Abstand dafür, dass die Kreise nebeneinander ohne Überlappung angeordnet werden.
- `Curvature` wird auf `[1 1]` gesetzt, um sicherzustellen, dass die Form, die durch den `rectangle`-Befehl gezeichnet wird, ein Kreis ist.
- `FaceColor` auf `'r'` gesetzt färbt die Kreise rot.

## 15 Frage

Speichern Sie die Kreis-Objekte via

```
stunde_kreise(i) = rectangle ( ...
```

in einen Vektor bestehend aus fünf Kreis-Objekten ab.

Warum ist es guter Programmierstil, den Vektor mit `gobjects` zu initialisieren?

### 15.1 Antwort

Das Speichern der Kreis-Objekte in einem Vektor ermöglicht es Ihnen, effizient auf diese Objekte zuzugreifen, beispielsweise um ihre Eigenschaften später im Code zu ändern oder um Informationen über sie zu extrahieren. Hier ist ein Beispiel, wie Sie dies umsetzen können, wobei der Vektor zu Beginn mit `gobjects` initialisiert wird:

```
stunde_kreise = gobjects(1, 5); % Initialisierung des Vektors mit Grafikobjekten

for i = 0:4
    stunde_kreise(i+1) = rectangle('Position', [i*2 0 1 1], 'Curvature', [1 1], 'FaceColor',
end
```

Die Initialisierung mit `gobjects` ist aus mehreren Gründen guter Programmierstil:

1. **Typsicherheit:** Durch die Initialisierung mit `gobjects` stellen Sie sicher, dass der Vektor explizit für Grafikobjekte vorgesehen ist. Dies erhöht die Lesbarkeit und Wartbarkeit des Codes, da sofort klar ist, welche Art von Objekten der Vektor enthält.
2. **Vermeidung von dynamischem Wachstum:** MATLAB-Vektoren wachsen dynamisch, wenn Elemente hinzugefügt werden. Dieser Vorgang kann ineffizient sein, da MATLAB intern möglicherweise wiederholt Speicher zuweisen und kopieren muss. Durch die Vorinitialisierung des Vektors mit der endgültigen Größe wird dieser Overhead vermieden.
3. **Fehlervermeidung:** Wenn Sie versuchen, auf ein Element eines nicht initialisierten Vektors zuzugreifen oder es zu modifizieren, führt dies zu einem Fehler. Die Vorinitialisierung mit `gobjects` schafft einen Vektor der richtigen Größe und verhindert solche Fehler, indem sie sicherstellt, dass jede Position im Vektor ein gültiges Grafikobjekt-Handle enthält, selbst wenn es sich um ein "leeres" Grafikobjekt handelt.
4. **Konsistenz:** Alle Elemente in `stunde_kreise` sind vom gleichen Typ (Grafikobjekte), was die Handhabung und Manipulation der Daten vereinfacht. Operationen, die auf den Vektor angewandt werden, können konsistent auf jedes Element angewendet werden, ohne dass Typüberprüfungen erforderlich sind.

Insgesamt trägt die Verwendung von `gobjects` zur Klarheit, Effizienz und Sicherheit Ihres MATLAB-Codes bei, wenn Sie mit Grafikobjekten arbeiten.

*Ich liebe diese Erklärungen!*

## 16 Frage

Schreiben Sie die entsprechenden Schleifen für die sechs Minuten- und die sechs Sekunden-LED.

Welche Positionsparameter müssen Sie bei den Minuten- und Sekunden-LED gegenüber den Stunden-LED ändern?

## 16.1 Antwort

Für die Minuten- und Sekunden-LEDs müssen Sie ähnliche Schleifen wie für die Stunden-LEDs verwenden, jedoch mit angepassten Positionsparametern, um die LED-Kreise in unterschiedlichen Zeilen oder Bereichen des Fensters zu platzieren. Da die Minuten- und Sekunden-LEDs jeweils sechs LEDs haben, müssen Sie die Schleifen entsprechend anpassen. Außerdem sollten Sie die y-Positionen anpassen, um sicherzustellen, dass die Minuten- und Sekunden-LEDs unterhalb der Stunden-LEDs angezeigt werden.

Hier ist ein Beispielcode, der die erforderlichen Schleifen und die Anpassung der Positionsparameter für die Minuten- und Sekunden-LEDs umfasst:

```
% Anzahl der LEDs
nStunden = 5;
nMinutenSekunden = 6;

% Initialisierung der Vektoren mit Grafikobjekten
stunde_kreise = gobjects(1, nStunden);
minute_kreise = gobjects(1, nMinutenSekunden);
sekunde_kreise = gobjects(1, nMinutenSekunden);

% Erzeugen der Stunden-LEDs
for i = 1:nStunden
    stunde_kreise(i) = rectangle('Position', [(i-1)*2, 4, 1, 1], 'Curvature', [1 1], 'FaceColo
end

% Erzeugen der Minuten-LEDs
for i = 1:nMinutenSekunden
    minute_kreise(i) = rectangle('Position', [(i-1)*2, 2, 1, 1], 'Curvature', [1 1], 'FaceColo
end

% Erzeugen der Sekunden-LEDs
for i = 1:nMinutenSekunden
    sekunde_kreise(i) = rectangle('Position', [(i-1)*2, 0, 1, 1], 'Curvature', [1 1], 'FaceColo
end
```

Die Hauptunterschiede im Vergleich zu den Stunden-LEDs sind:

1. **Anzahl der LEDs:** Für Minuten und Sekunden gibt es jeweils sechs LEDs, im Gegensatz zu fünf bei den Stunden-LEDs.
2. **Positionsparameter (y-Position):** Die y-Position der Minuten- und Sekunden-LEDs muss angepasst werden, um sie in separaten Zeilen unter den Stunden-LEDs zu positionieren. Im obigen Beispiel sind die Stunden-LEDs bei  $y = 4$ , die Minuten-LEDs bei  $y = 2$  und die Sekunden-LEDs bei  $y = 0$  platziert. Dies sorgt für eine klare visuelle Trennung zwischen Stunden, Minuten und Sekunden.

Durch diese Anpassungen können Sie eine Binäruhr mit separaten Zeilen für Stunden, Minuten und Sekunden erzeugen, wobei jede Gruppe von LEDs entsprechend positioniert ist.

*Fast perfekt:*

## 16.2 Frage

Die LED-Zeilen sollen rechtsbündig dargestellt werden.

### 16.2.1 Antwort

Um die LED-Zeilen rechtsbündig darzustellen, müssen Sie die Positionierung der Kreise so anpassen, dass die rechten Enden aller Zeilen (Stunden, Minuten und Sekunden) auf der gleichen x-Position enden. Nehmen wir an, dass die breiteste Zeile, die der Minuten und Sekunden mit jeweils sechs LEDs, den rechten Rand bestimmt. Da jede LED einen Durchmesser von 1 Einheit hat und wir einen kleinen Abstand zwischen ihnen berücksichtigen wollen, indem wir die doppelte Laufvariable für den Abstand nutzen, können wir die Positionierung entsprechend anpassen.

Angenommen, jede Zeile beginnt bei der x-Position 0 und der Abstand zwischen den LEDs beträgt 2 Einheiten (die LED selbst plus eine Einheit Platz zwischen den LEDs). Für eine rechtsbündige Darstellung müssen wir die Startpositionen der Stunden- und Minuten-LEDs nach rechts verschieben. Da die Minuten- und Sekunden-LEDs die längsten Zeilen sind, brauchen sie keine Anpassung, aber die Stunden-LEDs müssen um den Platz verschoben werden, der zwei zusätzlichen LEDs entspricht, um sie rechtsbündig auszurichten.

Hier ist, wie der Code unter Berücksichtigung der rechtsbündigen Ausrichtung angepasst werden kann:

```
% Anzahl der LEDs
nStunden = 5;
nMinutenSekunden = 6;

% Maximal mögliche x-Position (rechter Rand) - basierend auf Minuten/Sekunden
maxXPos = (nMinutenSekunden - 1) * 2;

% Initialisierung der Vektoren mit Grafikobjekten
stunde_kreise = gobjects(1, nStunden);
minute_kreise = gobjects(1, nMinutenSekunden);
sekunde_kreise = gobjects(1, nMinutenSekunden);

% Anpassung für rechtsbündige Stunden-LEDs
% Verschiebung um den Platz von zwei LEDs
stundenVerschiebung = (nMinutenSekunden - nStunden) * 2;

% Erzeugen der Stunden-LEDs mit Verschiebung für rechtsbündige Darstellung
for i = 1:nStunden
    stunde_kreise(i) = rectangle('Position', [stundenVerschiebung + (i-1)*2, 4, 1, 1], 'Curvature', [1 1], 'FaceColor', 'none');
end

% Erzeugen der Minuten-LEDs
for i = 1:nMinutenSekunden
    minute_kreise(i) = rectangle('Position', [(i-1)*2, 2, 1, 1], 'Curvature', [1 1], 'FaceColor', 'none');
end

% Erzeugen der Sekunden-LEDs
```

```

for i = 1:nMinutenSekunden
    sekunde_kreise(i) = rectangle('Position', [(i-1)*2, 0, 1, 1], 'Curvature', [1 1], 'FaceCo
end

```

Mit dieser Anpassung werden die Stunden-LEDs um den Platz von zwei LEDs nach rechts verschoben, um eine rechtsbündige Ausrichtung mit den Minuten- und Sekunden-LEDs zu erreichen. Die Verschiebung wird berechnet, indem die Differenz in der Anzahl der LEDs zwischen den Stunden- und den längeren Minuten- bzw. Sekundenzeilen mit dem Abstand multipliziert wird, der normalerweise zwischen zwei LEDs liegt.

*So klappt das! Immer wieder schön, wie sie auch auf Sonderwünsche eingeht ...*

## 17 Frage

Um jede Sekunde die Anzeige aktualisieren zu können, definieren Sie einen timer mit einer Periodendauer von einer Sekunde und einem fixedRate-ExecutionMode und speichern ihn unter dem Namen t ab.

Was würde geschehen, wenn Sie den ExecutionMode-Parameter weglassen würden?

### 17.1 Antwort

Wenn Sie einen Timer in MATLAB definieren, bestimmt der ExecutionMode-Parameter, wie und wann der Timer-Callback ausgeführt wird. Bei der Definition eines Timers, der jede Sekunde die Anzeige aktualisieren soll, würden Sie ihn wie folgt einrichten:

```

t = timer;
t.Period = 1;
t.ExecutionMode = 'fixedRate';
t.TimerFcn = @updateFunction; % 'updateFunction' ist der Name der Callback-Funktion

```

Der ExecutionMode-Parameter hat mehrere Optionen, unter anderem singleShot, fixedSpacing, und fixedRate.

- **singleShot:** Der Timer-Callback wird nur einmal nach dem Ablauf der Zeit (StartDelay oder Period) ausgeführt.
- **fixedSpacing:** Der Timer führt den Callback aus, indem zwischen den Ausführungen eine feste Zeitdauer wartet. Das bedeutet, dass die Zeit zwischen den Enden einer Ausführung und dem Beginn der nächsten Ausführung konstant ist. Wenn jedoch die Ausführung des Callbacks länger dauert als die festgelegte Periode, wird die nächste Ausführung so schnell wie möglich ausgeführt, nachdem die aktuelle Ausführung abgeschlossen ist, was zu einem unregelmäßigen Ausführungsintervall führen kann.
- **fixedRate:** Der Timer versucht, den Callback in regelmäßigen Abständen auszuführen, basierend auf der festgelegten Periodendauer, unabhängig davon, wie lange die Ausführung des Callbacks dauert. Das bedeutet, dass es zu Überlappungen kommen kann, wenn die Ausführung länger als die festgelegte Periode dauert.

Wenn Sie den ExecutionMode-Parameter weglassen, verwendet MATLAB standardmäßig den Modus singleShot. Das bedeutet, dass, wenn Sie nicht explizit ExecutionMode = 'fixedRate' (oder fixedSpacing) setzen, der Timer-Callback nur einmal ausgeführt wird, nachdem der Timer gestartet wurde und die festgelegte Zeit (Period) verstrichen ist. Für eine Anzeige, die jede Sekunde

aktualisiert werden soll, wäre das Weiterlassen von `ExecutionMode` also nicht geeignet, da Sie vermutlich möchten, dass die Aktualisierung regelmäßig und kontinuierlich erfolgt. Daher ist es wichtig, `ExecutionMode` auf `fixedRate` zu setzen, um sicherzustellen, dass die Anzeige kontinuierlich jede Sekunde aktualisiert wird.

*Gute - aber zu lange - Erklärung. Das Wort "Weiterlassen" gibt es nicht; es müsste "Weglassen" heißen.*

## 18 Frage

Verwenden Sie

```
t.TimerFcn = {@anzeigen, ... stunde_kreise, ... minute_kreise, ... sekunde_kreise}
```

um das Unterprogramm zu definieren, das der Timer dann sekundlich aufruft.

Starten Sie den Timer mit dem Befehl `start (t)`.

(Momentan erzeugt dies natürlich noch eine Fehlermeldung, da Sie erst im Folgenden das Unterprogramm anzeigen schreiben werden).

Erläutern Sie die Syntax des Befehls.

### 18.1 Antwort

Die von Ihnen beschriebene Syntax für den `TimerFcn`-Eigenschaft eines Timers in MATLAB wird verwendet, um eine Callback-Funktion festzulegen, die ausgeführt wird, wenn der Timer ein Ereignis auslöst (in diesem Fall jedes Mal, wenn die festgelegte Periode verstrichen ist). Hier ist die erklärte Syntax:

```
t.TimerFcn = {@anzeigen, stunde_kreise, minute_kreise, sekunde_kreise};
```

- **@anzeigen:** Dies ist ein Funktionshandle auf die Callback-Funktion `anzeigen`, die aufgerufen wird, wenn der Timer-Event ausgelöst wird. Das `@`-Symbol wird verwendet, um ein Funktionshandle in MATLAB zu erstellen.
- **stunde\_kreise, minute\_kreise, sekunde\_kreise:** Dies sind zusätzliche Argumente, die der Funktion `anzeigen` übergeben werden, wenn sie vom Timer aufgerufen wird. In diesem Kontext sind sie wahrscheinlich Handles oder Arrays von Grafikobjekten (vermutlich Kreisen), die die Stunden-, Minuten- und Sekunden-LEDs Ihrer Binäruhr darstellen. Diese Argumente ermöglichen es der Callback-Funktion `anzeigen`, auf die Grafikobjekte zuzugreifen und sie entsprechend der aktuellen Zeit zu aktualisieren.

Die Callback-Funktion `anzeigen` muss daher so definiert werden, dass sie mindestens ein Argument akzeptiert (den Timer selbst, der implizit als erstes Argument übergeben wird) und dann bis zu drei weitere Argumente (die hier übergebenen Grafikobjekt-Arrays). Die Signatur der Funktion könnte also so aussehen:

```
function anzeigen(myTimer, stunde_kreise, minute_kreise, sekunde_kreise)
    % Implementierung der Funktion, die die Anzeige aktualisiert
end
```

Um den Timer zu starten, verwenden Sie:

```
start(t);
```

Dieser Befehl startet den Timer `t`, woraufhin MATLAB beginnt, die Zeit zu zählen. Nach Ablauf jeder Periode (die durch `t.Period` festgelegt ist) wird die Funktion `anzeigen` aufgerufen, wobei `stunde_kreise`, `minute_kreise`, und `sekunde_kreise` als Argumente übergeben werden.

Die Verwendung von Timern und Callback-Funktionen auf diese Weise ist eine mächtige Methode in MATLAB, um periodische Aktionen auszuführen, wie z.B. die regelmäßige Aktualisierung einer grafischen Anzeige basierend auf zeitgesteuerten Ereignissen.

*Gut verständlich und hilfreich!*

## 19 Frage

Im entsprechenden Zeitgeber-Unterprogramm

```
function anzeigen (~, ~, ... stunde_kreise, ... minute_kreise, ... sekunde_kreise)
```

das Sie (mit einem eigenen `end` versehen) direkt ans Ende von `binuhr.mlx` schreiben, können Sie dann im Folgenden auf die LED-Objekte zugreifen.

Was bedeuten die beide Tilden in der Parameterliste des Unterprogrammes?

### 19.1 Antwort

In der Parameterliste einer MATLAB-Funktion werden Tilden (`~`) verwendet, um anzugeben, dass bestimmte Eingabeargumente ignoriert werden. Diese Notation ermöglicht es Ihnen, Funktionen zu definieren oder aufzurufen, ohne Namen für alle Eingabeparameter angeben zu müssen, insbesondere wenn einige der Eingabewerte für die Ausführung der Funktion nicht benötigt werden.

Im Kontext Ihres Zeitgeber-Unterprogramms `anzeigen`:

```
function anzeigen(~, ~, stunde_kreise, minute_kreise, sekunde_kreise)
    % Implementierung
end
```

bedeuten die beiden Tilden (`~`), dass die ersten beiden Argumente, die an die Funktion `anzeigen` übergeben werden, ignoriert werden sollen. Dies ist besonders nützlich in Szenarien, in denen die Funktion als Callback für einen Timer oder ein Event verwendet wird. Bei solchen Callbacks sind die ersten beiden Argumente typischerweise das Handle des Objekts, das das Event auslöst (hier der Timer selbst), und das Event-Daten-Objekt, das Informationen über das ausgelöste Event enthält. Wenn Ihre Logik diese Informationen nicht benötigt, können Sie sie mit `~` ignorieren, anstatt unnötige Variablennamen zu definieren.

In Ihrem speziellen Fall bedeutet dies, dass `anzeigen` als Callback-Funktion für ein Timer-Event konzipiert ist. Der Timer ruft `anzeigen` mit einigen Standardargumenten auf (üblicherweise das Timer-Objekt und Event-Daten), aber da die Funktion diese Informationen nicht verwendet, werden sie ignoriert. Stattdessen konzentriert sich die Funktion auf die Bearbeitung der ihr übergebenen LED-Objekte (`stunde_kreise`, `minute_kreise`, `sekunde_kreise`), um die Anzeige der Binäruhr entsprechend zu aktualisieren.

*Großartig, oder?!*

## 20 Frage

Im Zeitgeber-Unterprogramm verwenden Sie wiederum `datetime`, um auf die aktuelle Zeit zuzugreifen und die entsprechenden Attribute, um die lokalen Variablen `stunde`, `minute` und `sekunde` zu füllen.

Verwenden Sie Matlabs `dec2bin`-Befehl, um die Binärdarstellung der drei Variablen zu erhalten.

Als was liefert `dec2bin` die Binärdarstellung zurück?

Warum sollten Sie deshalb auch den zweiten Parameter von `dec2bin` verwenden?

### 20.1 Antwort

Der MATLAB-Befehl `dec2bin` wandelt eine Dezimalzahl in ihre Binärdarstellung um. Die Binärdarstellung wird als Zeichenkette (String) zurückgegeben. Das bedeutet, dass jede Ziffer der binären Zahl als einzelnes Zeichen in einem String gespeichert wird.

Der Grund, warum Sie auch den zweiten Parameter von `dec2bin` verwenden sollten, liegt darin, dass dieser Parameter die Mindestanzahl von Bits (oder Zeichen) spezifiziert, die für die Binärdarstellung verwendet werden sollen. Ohne diesen zweiten Parameter verwendet `dec2bin` gerade so viele Bits, wie notwendig sind, um die Zahl darzustellen. Das kann zu Inkonsistenzen in der Länge der Binärdarstellungen führen, was besonders dann problematisch ist, wenn Sie eine einheitliche Darstellung für Uhrzeiten (Stunden, Minuten, Sekunden) benötigen, die in ein festes Format (wie eine Anzeige mit einer festen Anzahl von LEDs pro Zeitsegment) umgesetzt werden soll.

Wenn Sie beispielsweise eine Uhr mit einer Binäranzeige programmieren, könnten Sie für Stunden 5 Bits (um Zahlen von 0 bis 23 darzustellen), für Minuten und Sekunden jeweils 6 Bits (um Zahlen von 0 bis 59 darzustellen) benötigen. Ohne den zweiten Parameter könnten Sie für frühe Stunden wie 1 Uhr morgens (`dec2bin(1)`) eine Binärdarstellung von nur 1 erhalten, während Sie tatsächlich 00001 erwarten, um alle 5 Positionen der Stunden-LEDs zu füllen.

Hier ein Beispiel, wie der zweite Parameter verwendet wird:

```
currentTime = datetime('now');
stunde = hour(currentTime);
minute = minute(currentTime);
sekunde = second(currentTime);
```

```
stundeBin = dec2bin(stunde, 5); % Sichert, dass die Binärdarstellung immer 5 Zeichen lang ist
minuteBin = dec2bin(minute, 6); % Sichert, dass die Binärdarstellung immer 6 Zeichen lang ist
sekundeBin = dec2bin(sekunde, 6); % Sichert, dass die Binärdarstellung immer 6 Zeichen lang ist
```

Durch die Verwendung des zweiten Parameters von `dec2bin` können Sie sicherstellen, dass die Binärdarstellungen von Stunden, Minuten und Sekunden eine konsistente Länge aufweisen, was die Anzeige auf Ihrer Binäruhr erheblich vereinfacht und standardisiert.

*Absolut vorbildlich!*

## 21 Frage

Verwenden Sie eine Schleife über die fünf Ziffern der Stunden-Binärdarstellung, um in Abhängigkeit davon, ob das jeweilige Zeichen den Wert 1 hat (`strcmp`), die Farbe der entsprechenden LED auf rot bzw. weiß zu setzen.

Warum müssen Sie nicht explizit abfragen, ob das Zeichen den Wert 0 hat?

### 21.1 Antwort

In Ihrem Zeitgeber-Unterprogramm verwenden Sie eine Schleife, um über jedes Zeichen der Binärdarstellung der Stunden zu iterieren. Innerhalb dieser Schleife nutzen Sie `strcmp` zur Überprüfung, ob das aktuelle Zeichen den Wert '1' hat, um entsprechend die Farbe der LED auf Rot zu setzen. Andernfalls – und das ist der Schlüssel hier – wird angenommen, dass das Zeichen den Wert '0' hat, ohne dass eine explizite Abfrage notwendig ist. Dies liegt daran, dass in der Binärdarstellung nur zwei mögliche Werte existieren: '0' und '1'. Wenn das Zeichen nicht '1' ist, muss es demzufolge '0' sein.

Hier ist, wie der Code aussehen könnte:

```
for i = 1:length(stundeBin)
    if strcmp(stundeBin(i), '1')
        stunde_kreise(i).FaceColor = 'red';
    else
        stunde_kreise(i).FaceColor = 'white';
    end
end
```

In diesem Code-Beispiel:

- Die Schleife iteriert durch jedes Zeichen in `stundeBin`, der Binärdarstellung der Stunden.
- `strcmp(stundeBin(i), '1')` prüft, ob das i-te Zeichen '1' ist. Wenn ja, wird die Farbe der i-ten LED auf Rot gesetzt.
- Die `else`-Klausel fängt alle Fälle ab, in denen das Zeichen nicht '1' ist, also muss es '0' sein. In diesem Fall wird die Farbe der i-ten LED auf Weiß gesetzt. Es ist nicht notwendig, explizit zu überprüfen, ob das Zeichen '0' ist, da es keine anderen Möglichkeiten gibt – das Binärsystem besteht nur aus diesen zwei Ziffern.

Durch dieses Vorgehen wird der Code einfacher und klarer, da keine redundanten Überprüfungen für den Wert '0' erforderlich sind. Die binäre Logik gewährleistet, dass jedes Zeichen entweder '1' ist, was eine aktive (rote) LED darstellt, oder '0', was eine inaktive (weiße) LED bedeutet.

## 22 Frage

Schreiben Sie die entsprechenden Schleifen für die Minuten und Sekunden und starten Sie Ihre Binäruhr.

Kann es passieren, dass alle Sekunden-LED rot sind?

## 22.1 Antwort

Um die entsprechenden Schleifen für Minuten und Sekunden in Ihrer Binäruhr zu implementieren und die Binäruhr zu starten, ergänzen Sie Ihren Code um ähnliche Schleifen, wie sie für die Stunden-Binärdarstellung verwendet wurden. Hier ist, wie der vollständige Code unter Berücksichtigung von Stunden, Minuten und Sekunden aussehen könnte:

```
% Angenommen, dies ist Teil Ihrer Timer Callback-Funktion 'anzeigen'
currentTime = datetime('now');
stunde = hour(currentTime);
minute = minute(currentTime);
sekunde = second(currentTime);

stundeBin = dec2bin(stunde, 5);
minuteBin = dec2bin(minute, 6);
sekundeBin = dec2bin(sekunde, 6);

% Aktualisieren der Stunden-LEDs
for i = 1:length(stundeBin)
    if strcmp(stundeBin(i), '1')
        stunde_kreise(i).FaceColor = 'red';
    else
        stunde_kreise(i).FaceColor = 'white';
    end
end

% Aktualisieren der Minuten-LEDs
for i = 1:length(minuteBin)
    if strcmp(minuteBin(i), '1')
        minute_kreise(i).FaceColor = 'red';
    else
        minute_kreise(i).FaceColor = 'white';
    end
end

% Aktualisieren der Sekunden-LEDs
for i = 1:length(sekundeBin)
    if strcmp(sekundeBin(i), '1')
        sekunde_kreise(i).FaceColor = 'red';
    else
        sekunde_kreise(i).FaceColor = 'white';
    end
end
```

Zur Frage, ob es passieren kann, dass alle Sekunden-LEDs rot sind: Ja, das ist möglich. Die Sekunden-LEDs repräsentieren die Sekunden in binärer Form, und alle LEDs wären rot, wenn die Binärdarstellung der Sekunden den maximalen Wert anzeigt, der mit sechs LEDs dargestellt werden kann. Da jede LED ein Bit repräsentiert und sechs LEDs verwendet werden, können Sie Werte von 0 bis ( $2^6 - 1 = 63$ ) darstellen. Der maximale Wert, 63 Sekunden, würde in der Binärdarstellung

als 111111 erscheinen, was dazu führen würde, dass alle Sekunden-LEDs rot sind. Dies tritt auf, wenn die Sekundenanzeige von der 59. zur 60. Sekunde übergeht (was tatsächlich den Übergang zur nächsten Minute markiert), da 59 Sekunden in Binärform 111011 entsprechen und die Uhr dann zurück auf 000000 für 0 Sekunden springt, nicht auf 111111. Allerdings ist die Idee, dass, wenn es theoretisch eine Darstellung für 63 Sekunden gäbe, alle LEDs rot leuchten würden. Im realen Gebrauch der Uhr zeigt der Zustand, in dem alle Sekunden-LEDs rot sind, nie auf, da die Uhr von 59 Sekunden (Binär: 111011) direkt auf 0 Sekunden (Binär: 000000) umspringt.

*Ähh, das mit den 63 Sekunden ist ja wohl dummes Gelaber!*

## 22.2 Frage

Schreib das komplette lauffähige Programm basierend auf deinen vorherigen Antworten.

### 22.2.1 Antwort

Basierend auf den bisherigen Diskussionen und Erklärungen hier ist ein Beispiel für ein vollständiges, lauffähiges MATLAB-Programm, das eine Binäruhr implementiert. Dieses Programm aktualisiert die Anzeige für Stunden, Minuten und Sekunden jede Sekunde mithilfe eines Timers. Beachten Sie, dass dieses Skript direkte Anpassungen oder Ergänzungen erfordern könnte, um es in Ihr spezifisches Projekt oder Ihre Umgebung einzubinden:

```
function binuhr
    % Erzeugt das Fenster für die Binäruhr
    figure('Name', 'Binuhr', 'Numbertitle', 'off', 'Menubar', 'none', 'Visible', 'on');
    axis off;
    hold on;

    % Initialisierung der LED-Kreise
    stunde_kreise = gobjects(1, 5);
    minute_kreise = gobjects(1, 6);
    sekunde_kreise = gobjects(1, 6);

    % Erzeugen der Stunden-LEDs
    for i = 1:5
        stunde_kreise(i) = rectangle('Position', [i*2-2, 6, 1, 1], 'Curvature', [1 1], 'FaceColor', 'r');
    end

    % Erzeugen der Minuten-LEDs
    for i = 1:6
        minute_kreise(i) = rectangle('Position', [i*2-2, 4, 1, 1], 'Curvature', [1 1], 'FaceColor', 'r');
    end

    % Erzeugen der Sekunden-LEDs
    for i = 1:6
        sekunde_kreise(i) = rectangle('Position', [i*2-2, 2, 1, 1], 'Curvature', [1 1], 'FaceColor', 'r');
    end

    % Erstellen und Konfigurieren des Timers
```

```

t = timer;
t.Period = 1;
t.ExecutionMode = 'fixedRate';
t.TimerFcn = {@anzeigen, stunde_kreise, minute_kreise, sekunde_kreise};

% Starten des Timers
start(t);

function anzeigen(~, ~, stunde_kreise, minute_kreise, sekunde_kreise)
    % Aktuelle Zeit holen
    currentTime = datetime('now');
    stunde = hour(currentTime);
    minute = minute(currentTime);
    sekunde = second(currentTime);

    % Umwandlung in Binärdarstellung
    stundeBin = dec2bin(stunde, 5);
    minuteBin = dec2bin(minute, 6);
    sekundeBin = dec2bin(sekunde, 6);

    % Aktualisieren der Stunden-LEDs
    updateLEDs(stunde_kreise, stundeBin);

    % Aktualisieren der Minuten-LEDs
    updateLEDs(minute_kreise, minuteBin);

    % Aktualisieren der Sekunden-LEDs
    updateLEDs(sekunde_kreise, sekundeBin);
end

function updateLEDs(led_kreise, binString)
    % Iteriert über den Binärstring und setzt die Farbe der LEDs
    for i = 1:length(binString)
        if strcmp(binString(i), '1')
            led_kreise(i).FaceColor = 'red';
        else
            led_kreise(i).FaceColor = 'white';
        end
    end
end
end

```

Dieses Skript definiert eine Hauptfunktion `binuhr`, die die grafische Oberfläche für die Uhr initialisiert, die LED-Kreise erzeugt und einen Timer konfiguriert, der jede Sekunde die Uhrzeit aktualisiert. Die Hilfsfunktion `anzeigen` wird vom Timer aufgerufen, holt die aktuelle Zeit, wandelt diese in Binärdarstellungen um und aktualisiert dann die Farben der LED-Kreise entsprechend. Die Funktion `updateLEDs` wird verwendet, um die Farben der LEDs basierend auf der Binärdarstellung zu aktualisieren.

*Auch das klappt aus diversen Gründen leider nicht.*

# 9 Pong

April 25, 2024

## 1 Frage

Sie möchten im Folgenden ein Programm schreiben, das dem guten alten Pong-Klassiker (als Einzelspielervariante) nachempfunden ist. Öffnen Sie dazu ein Fenster ohne `NumberTitle`, und ohne `MenuBar`. Geben Sie ihm den Namen `Pong`. Denken Sie bitte wieder daran, dass Sie externe Fenster aus einem Live Script heraus mit `Visible = 'on'` ... sichtbar machen müssen.

Erzeugen Sie in dem Fenster ein Achsensystem mit folgenden Eigenschaften:

```
spielfeld = axes (... 'DataAspectRatioMode', 'manual', ... 'XLim', [0 1], ... 'YLim', [0 1], ... 'Color', 'black' ... );
```

Außerdem verwenden Sie:

```
spielfeld.XAxis.Visible = 'off'; spielfeld.YAxis.Visible = 'off';
```

Erläutern Sie die Befehle und deren Parameter.

### 1.1 Antwort

Um das Programm für eine Einzelspieler-Variante des klassischen Pong-Spiels in Matlab zu schreiben, beginnen wir mit der Erstellung des Hauptfensters und des Spielfelds, wie in Ihrer Anleitung beschrieben. Hier erläutere ich die verwendeten Befehle und deren Parameter:

#### 1. Fenster Erstellen:

- Matlab verwendet die `figure`-Funktion, um ein neues Fenster zu erzeugen. Standardmäßig hat dieses Fenster einen Titel und eine Menüleiste. Um ein Fenster ohne `NumberTitle` und ohne `MenuBar` zu erstellen und ihm den Namen "Pong" zu geben, verwenden Sie folgenden Befehl:

```
fig = figure('NumberTitle', 'off', 'MenuBar', 'none', 'Name', 'Pong', 'Visible', 'on');  
– NumberTitle, 'off' entfernt den Standardfenstertitel (z.B. "Figure 1").  
– MenuBar, 'none' entfernt die Menüleiste, um das Fenster minimal zu halten.  
– Name, 'Pong' setzt den Fenstertitel auf "Pong".  
– Visible, 'on' stellt sicher, dass das Fenster sichtbar ist, insbesondere nützlich, wenn das Script in einer Umgebung ausgeführt wird, wo Fenster standardmäßig nicht sichtbar sind (z.B. in Matlab Live Scripts).
```

#### 2. Achsensystem Erstellen:

- Für das Achsensystem, das das Spielfeld darstellt, verwenden Sie den `axes`-Befehl mit spezifischen Eigenschaften:

```
spielfeld = axes('DataAspectRatioMode', 'manual', 'XLim', [0 1], 'YLim', [0 1], 'Col
```

- `DataAspectRatioMode`, 'manual' erlaubt es Ihnen, das Seitenverhältnis der Achsen manuell zu kontrollieren. Dies stellt sicher, dass das Spielfeld unabhängig von der Größe des Fensters quadratisch bleibt.
- `XLim` und `YLim` setzen die Grenzen der X- und Y-Achsen auf [0 1], was ein normiertes Koordinatensystem von 0 bis 1 auf beiden Achsen erstellt.
- `Color`, 'black' setzt die Hintergrundfarbe des Achsensystems auf Schwarz.

### 3. Achsen Verbergen:

- Um die Achsen selbst unsichtbar zu machen, sodass nur das Spielfeld ohne Achsenbeschriftungen oder Linien sichtbar ist, verwenden Sie:

```
spielfeld.XAxis.Visible = 'off';
```

```
spielfeld.YAxis.Visible = 'off';
```

- `spielfeld.XAxis.Visible`, 'off' und `spielfeld.YAxis.Visible`, 'off' verbergen die X- und Y-Achsen des Achsensystems.

Diese Schritte erstellen ein einfaches Spielfeld für Pong, auf dem Sie weitere grafische Elemente wie den Ball und den Schläger hinzufügen können.

*Prima!*

## 2 Frage

Ziehen Sie mit dem Befehl `line` eine vertikale, weiße, gestrichelte Linie der Stärke 2 in der Mitte des Spielfeldes.

Warum ist es an dieser Stelle sinnvoller, den Befehl `line` und nicht `plot` oder `gplot` zu verwenden?

### 2.1 Antwort

Um eine vertikale, weiße, gestrichelte Linie in der Mitte des Spielfeldes zu ziehen, nutzen wir den `line`-Befehl in Matlab. Hier ist der entsprechende Code:

```
line([0.5 0.5], [0 1], 'Color', 'white', 'LineStyle', '--', 'LineWidth', 2);
```

- `[0.5 0.5]` bestimmt die x-Koordinaten der Linie, was bedeutet, dass die Linie bei  $x = 0.5$  beginnt und endet, was sie vertikal in der Mitte des Spielfeldes platziert.
- `[0 1]` sind die y-Koordinaten der Linie, die vom unteren Rand ( $y = 0$ ) bis zum oberen Rand ( $y = 1$ ) des Spielfeldes reichen.
- `'Color', 'white'` setzt die Farbe der Linie auf Weiß.
- `'LineStyle', '--'` definiert die Linie als gestrichelt.
- `'LineWidth', 2` bestimmt die Stärke der Linie auf 2.

**Warum ist die Verwendung von `line` hier sinnvoller als `plot` oder `fplot`?**

1. **Einfachheit und Direktheit:** Der `line`-Befehl zeichnet direkt eine Linie zwischen zwei Punkten ohne die Notwendigkeit, ein Datenset zu erstellen oder eine Funktion zu definieren. Es ist daher für einfache, geradlinige Zeichnungen wie diese am effizientesten.
2. **Keine Datenabhängigkeit:** Während `plot` und `fplot` primär zum Zeichnen von Datenpunkten oder Funktionen genutzt werden, was impliziert, dass man entweder Datenpunkte oder eine funktionale Beziehung zwischen  $x$  und  $y$  hat, benötigt `line` nur Start- und Endpunkte. Für eine statische, dekorative Linie wie in einem Pong-Spiel, die keine Daten repräsentiert oder einer Funktion folgt, ist `line` also angemessener.

3. **Geringerer Overhead:** `fplot` ist speziell dafür ausgelegt, Funktionen über ein bestimmtes Intervall zu plotten. Für eine einfache Linie würde dies unnötigen Overhead bedeuten, da man eine konstante Funktion definieren müsste. `plot` könnte zwar auch verwendet werden, indem zwei Punkte übergeben werden, aber `line` ist direkter und zeigt klar die Absicht, einfach nur eine Linie zwischen zwei Punkten zu zeichnen.

Zusammengefasst ist der `line`-Befehl für diese Aufgabe am besten geeignet, da er genau für solche einfachen Zeichenaufgaben wie das Zeichnen einer direkten Linie zwischen zwei festen Punkten konzipiert ist.

*Bisschen ausführlich, aber okay.*

### 3 Frage

Setzen Sie die Variable `spielerhoehe` auf 0.2 und erzeugen Sie den (linken) Spieler als weißes Rechteck der Breite 0.01. Weisen Sie das Rechteck der Variablen `spieler` zu.

Was bedeutet es, wenn Sie beim Erzeugen des Spielers den `LineStyle` auf `none` setzen?

#### 3.1 Antwort

Um den (linken) Spieler als weißes Rechteck mit einer Höhe von 0.2 und einer Breite von 0.01 zu erzeugen und dieses Rechteck einer Variablen zuzuweisen, können Sie in Matlab die Funktion `rectangle` verwenden. Hier ist, wie Sie es machen könnten:

```
spielerhoehe = 0.2;
spielerBreite = 0.01;
spielerPositionX = 0.05; % Ein kleiner Abstand vom linken Rand
spielerPositionY = 0.5 - spielerhoehe / 2; % Zentrieren des Spielers vertikal

spieler = rectangle('Position', [spielerPositionX, spielerPositionY, spielerBreite, spielerhoehe
```

- `Position` legt die Position und Größe des Rechtecks fest. Das Format ist `[x y width height]`, wobei `x` und `y` die Koordinaten der unteren linken Ecke des Rechtecks sind. `width` und `height` stehen für die Breite und Höhe des Rechtecks.
- `'FaceColor', 'white'` setzt die Füllfarbe des Rechtecks auf Weiß.
- `'LineStyle', 'none'` entfernt die Umrandung des Rechtecks.

Wenn Sie beim Erzeugen des Spielers `LineStyle` auf `none` setzen, bedeutet das, dass das Rechteck **keine sichtbare Kontur oder Randlinie** haben wird. Das ist besonders nützlich, wenn Sie wollen, dass das Rechteck eine einheitliche Füllung ohne Unterscheidung zwischen dem Rand und dem Inneren hat. Für grafische Objekte wie das Paddel in einem Pong-Spiel ist dies oft erwünscht, da das Objekt als einheitlicher Block ohne Umrisslinie erscheinen soll.

### 4 Frage

Setzen Sie einen weißen Ball mit dem Durchmesser 0.03 und ohne Rand in die Mitte des Spielfeldes und weisen Sie ihn der Variablen `ball` zu.

Warum ist es dabei eigentlich nicht korrekt, als ersten und zweiten Parameter der Position 0.5 anzugeben? Welche Werte wären korrekt? Warum ist das aber im Endeffekt nicht wirklich wichtig?

*Den Abstand vom linken Rand hat sie selbst gewählt.*

#### 4.1 Antwort

Um einen weißen Ball mit einem Durchmesser von 0.03 in die Mitte des Spielfeldes zu setzen und diesen der Variablen `ball` zuzuweisen, nutzen Sie in Matlab erneut die Funktion `rectangle`, diesmal mit der Eigenschaft `'Curvature'` gesetzt auf `[1 1]`, um ein Kreis zu erzeugen:

```
ballDurchmesser = 0.03;
ballRadius = ballDurchmesser / 2;
ballPositionX = 0.5 - ballRadius; % Korrektur um den Ball zentriert zu platzieren
ballPositionY = 0.5 - ballRadius; % Korrektur um den Ball zentriert zu platzieren

ball = rectangle('Position', [ballPositionX, ballPositionY, ballDurchmesser, ballDurchmesser])
```

**Warum ist es nicht korrekt, als ersten und zweiten Parameter der Position einfach 0.5 anzugeben?**

Wenn man einfach `0.5, 0.5` als Koordinaten angibt, würde dies die untere linke Ecke des Rechtecks (oder in diesem Fall des Kreises) in der Mitte des Spielfeldes platzieren, nicht den Mittelpunkt des Kreises selbst. Um den Kreis tatsächlich mittig zu positionieren, müssen Sie von diesen Koordinaten den Radius (oder die Hälfte der Breite/Höhe des Rechtecks, da der Durchmesser des Balls 0.03 beträgt) abziehen. Dies zentriert den Ball auf dem Spielfeld.

**Welche Werte wären korrekt?**

Die korrekten Werte für die `x` und `y` Position, um den Ball in der Mitte zu positionieren, sind `0.5 - ballRadius` für `x` und ebenfalls `0.5 - ballRadius` für `y`.

**Warum ist das aber im Endeffekt nicht wirklich wichtig?**

Für ein einfaches Pong-Spiel, besonders in einer Lehr- oder Lernumgebung, ist die genaue Platzierung des Balls in der Mitte des Spielfeldes vielleicht nicht kritisch wichtig. Das Ziel ist es oft, die grundlegenden Konzepte der Programmierung, Grafikerstellung und Ereignissteuerung zu vermitteln. Solange der Ball nahe der Mitte erscheint und das Spiel wie erwartet funktioniert, beeinflusst eine leichte Abweichung von der exakten Mitte das Spielerlebnis nicht wesentlich. Zudem bietet diese Situation eine gute Lerngelegenheit über die Bedeutung von Koordinatensystemen und grafischer Platzierung in der Programmierung.

*Alles sehr hilfreich.*

## 5 Frage

Jetzt geht es an die Vorbereitung der Simulationsschleife. Verwenden Sie den Befehl `title`, um eine `0` in den Titel des Fensters zu schreiben und weisen Sie das erzeugte Textobjekt der Variablen `titel` zu. Initialisieren Sie die Abtastzeit `delta_t` mit `0.001`, setzen Sie das Abbruch-Flag `game_over` auf `false` und die Anfangsgeschwindigkeit `ges` auf `[1 -1]`.

In welche Richtung wird der Ball also losfliegen?

## 5.1 Antwort

Um eine 0 in den Titel des Fensters zu schreiben und das erzeugte Textobjekt der Variablen `titel` zuzuweisen, können Sie den Befehl `title` in Matlab verwenden. Hier ist ein Beispielcode:

```
titel = title('0');
```

Um die Abtastzeit `delta_t` mit 0.001 zu initialisieren, das Abbruch-Flag `game_over` auf `false` zu setzen, und die Anfangsgeschwindigkeit `ges` auf `[1 -1]` zu setzen, verwenden Sie folgende Befehle:

```
delta_t = 0.001;  
game_over = false;  
ges = [1 -1];
```

Die Geschwindigkeit `ges = [1 -1]` gibt die Bewegungsrichtung und -geschwindigkeit des Balls an. Der erste Wert in dem Vektor (hier 1) bestimmt die Geschwindigkeit in horizontaler Richtung (x-Richtung), und der zweite Wert (hier -1) bestimmt die Geschwindigkeit in vertikaler Richtung (y-Richtung).

Da der erste Wert positiv ist, wird sich der Ball horizontal nach rechts bewegen. Der negative zweite Wert bedeutet, dass sich der Ball vertikal nach unten bewegen wird. Zusammengefasst wird der Ball also diagonal nach rechts unten starten, wenn die Simulation beginnt.

*Läuft alles fehlerfrei!*

## 6 Frage

Da Sie den Spieler gleich mit der Maus steuern wollen, ermitteln Sie mit

```
bildschirmgroesse = get(groot, 'ScreenSize')
```

die Auflösung Ihres Monitors und weisen dessen Höhe der Variablen `bildschirmhoehe` zu.

Welche Funktion hat `groot` im Matlab-Universum?

Welchen Parameter des `get`-Befehls sollten Sie nutzen, wenn Sie mehrere Monitor verwenden?

### 6.1 Antwort

In Matlab ist `groot` ein Befehl, der das Root-Objekt anspricht. Das Root-Objekt ist das höchste Objekt in der Matlab-Grafikobjekthierarchie. Es repräsentiert die Matlab-Grafikumgebung selbst und bietet Zugriff auf Eigenschaften und Einstellungen, die die gesamte Grafikumgebung betreffen, wie beispielsweise die Bildschirmgröße, die Standardfarben und andere allgemeine Einstellungen.

Wenn Sie die Auflösung Ihres Monitors mit dem Befehl ermitteln möchten:

```
bildschirmgroesse = get(groot, 'ScreenSize');
```

Dann gibt `bildschirmgroesse` ein Array zurück, das folgende Werte enthält: `[x y Breite Hoehe]`, wobei `x` und `y` die Position des unteren linken Punktes des Bildschirms (normalerweise `[0 0]`) und `Breite` sowie `Hoehe` die Dimensionen des Bildschirms in Pixeln darstellen. Um die Höhe Ihres Bildschirms der Variablen `bildschirmhoehe` zuzuweisen, würden Sie dann auf das vierte Element dieses Arrays zugreifen:

```
bildschirmhoehe = bildschirmgroesse(4);
```

**Falls Sie mehrere Monitore verwenden**, könnte die Verwendung von `ScreenSize` des Root-Objekts zu Verwirrungen führen, da es die Gesamtgröße aller Monitore zurückgeben kann, abhängig davon, wie Ihr System konfiguriert ist. Um spezifische Informationen für mehrere Monitore zu erhalten, sollten Sie stattdessen die Eigenschaft `'MonitorPositions'` verwenden:

```
monitorPositionen = get(groot, 'MonitorPositions');
```

Diese Eigenschaft gibt ein Array zurück, in dem jede Zeile die Position und Größe eines Monitors in Ihrem System repräsentiert. Die Spalten des Arrays folgen dem gleichen Format wie `ScreenSize`: `[x y Breite Hoehe]`. So können Sie die Dimensionen und Positionen jedes Monitors einzeln abrufen und beispielsweise bestimmen, welcher Monitor als primärer Monitor dient oder die Größe spezifischer Monitore ermitteln.

*Sehr schön, dass sie auch "Monitorpositions" kennt.*

## 7 Frage

Jetzt beginnt die Simulationsschleife mit der Zeile

```
while ~game_over
```

Was müssen Sie also in der Schleife machen, um die Simulation zu stoppen?

### 7.1 Antwort

Um die Simulation innerhalb der Schleife zu stoppen, müssen Sie die Variable `game_over` auf `true` setzen. Diese Aktion führt dazu, dass die Bedingung `~game_over` (nicht `game_over`) falsch wird und somit die `while`-Schleife beendet wird. Die Variable `game_over` dient als Steuerungsflagge für die Schleifenbedingung und erlaubt es der Simulation, unter bestimmten Umständen angehalten zu werden.

In einem Pong-Spiel könnten solche Umstände beispielsweise sein, dass der Ball einen der Bildschirmränder berührt, was ein Punkteverlust bedeuten könnte, oder dass der Spieler eine Taste drückt, um das Spiel zu beenden. Innerhalb der `while`-Schleife würden Sie demnach eine oder mehrere Bedingungen prüfen, und wenn diese Bedingungen erfüllt sind, setzen Sie `game_over = true;`

Hier ist ein sehr einfaches Beispiel:

```
while ~game_over
    % Code zur Aktualisierung des Spiels

    % Bedingung zum Beenden der Simulation
    if bestimmteBedingungErfuellt
        game_over = true;
    end
end
```

In diesem Beispiel steht `bestimmteBedingungErfuellt` stellvertretend für eine tatsächliche Bedingung, die Sie in Ihrem Code definieren müssten, wie z.B. eine Kollisionserkennung oder eine Nutzerinteraktion, die das Ende des Spiels signalisiert.

*Wunderbare Erklärung.*

## 8 Frage

In der Schleife wollen Sie als erstes die Spielerposition aus der aktuellen Mausposition berechnen. Lesen Sie dazu die aktuelle Mausposition aus dem Graphics-Root-Objekt aus und weisen Sie die Vertikalkomponente der Variablen `maus_y` zu.

Warum brauchen Sie die Horizontalkomponente der Mausposition nicht?

### 8.1 Antwort

Um die aktuelle Mausposition aus dem Graphics-Root-Objekt in Matlab auszulesen und die Vertikalkomponente der Variablen `maus_y` zuzuweisen, verwenden Sie den Befehl `get` zusammen mit der Eigenschaft `PointerLocation` des `groot`-Objekts. Hier ist ein Beispielcode:

```
mausPosition = get(groot, 'PointerLocation');  
maus_y = mausPosition(2);
```

In einem Pong-Spiel bewegt sich der Spieler (in diesem Fall das Paddel) typischerweise nur entlang der vertikalen Achse (y-Achse), um den Ball abzuwehren. Die Bewegung ist auf eine Dimension beschränkt, was die Steuerung vereinfacht und dem klassischen Gameplay entspricht. Daher ist nur die Vertikalkomponente (`maus_y`) der Mausposition relevant, um die Position des Spielers zu aktualisieren.

Die Horizontalkomponente der Mausposition wird nicht benötigt, weil:

1. **Spielmechanik:** In der Grundversion des Pong-Spiels können sich die Paddel nur vertikal bewegen, um den Ball abzufangen. Eine horizontale Bewegung der Paddel ist nicht vorgesehen und würde die klassischen Spielregeln verändern.
2. **Einfachheit der Steuerung:** Die Beschränkung auf eine Bewegungsachse macht das Spiel leichter steuerbar und hält die Spielmechanik einfach. Spieler müssen sich nur auf die Bewegung nach oben und unten konzentrieren, was die Reaktionszeit und Strategie vereinfacht.
3. **Spielbalance:** Die Begrenzung auf vertikale Bewegungen sorgt für eine klare und faire Spielbalance. Beide Spieler haben dieselben Bewegungsmöglichkeiten und sind gleichberechtigt in ihrer Fähigkeit, den Ball abzuwehren.

Insgesamt ist die Entscheidung, nur die Vertikalkomponente der Mausposition zu verwenden, eine Designwahl, die auf den klassischen Pong-Spielregeln und der beabsichtigten Spielerfahrung basiert.

*So isses.*

## 9 Frage

Berechnen Sie die neue Spielerposition gemäß

```
spieler.Position(2) = 5 * maus_y / bildschirmhoehe - 3
```

In welchem Zahlenbereich liegt `maus_y`?

Was bewirken die Konstanten in der “Geradengleichung”?

## 9.1 Antwort

Die Anweisung zur Berechnung der neuen Spielerposition:

```
spieler.Position(2) = 5 * maus_y / bildschirmhoehe - 3;
```

basiert auf der aktuellen Vertikalkomponente der Mausposition `maus_y` und der Höhe des Bildschirms `bildschirmhoehe`. Diese Gleichung skaliert und verschiebt die Mausposition in den Wertebereich, der für die Bewegung des Spielers (Paddels) im Spiel verwendet wird.

### 9.1.1 In welchem Zahlenbereich liegt `maus_y`?

Der Wertebereich von `maus_y` hängt von der vertikalen Auflösung Ihres Monitors ab. Die Mausposition `maus_y` ist 0 am oberen Rand des Bildschirms und nimmt zu bis zum Wert `bildschirmhoehe` am unteren Rand des Bildschirms. Wenn Ihre Bildschirmauflösung beispielsweise 1080 Pixel in der Höhe beträgt, würde `maus_y` im Bereich von 0 bis 1080 liegen.

### 9.1.2 Was bewirken die Konstanten in der “Geradengleichung”?

Die “Geradengleichung” verwendet zwei Konstanten, 5 und -3, um die Mausposition zu skalieren und zu verschieben. Hier ist, was sie bewirken:

1. **5:** Diese Konstante bestimmt, wie stark die Mausposition skaliert wird. Eine größere Zahl würde bedeuten, dass die Bewegung des Spielers (Paddels) im Verhältnis zur Mausbewegung verstärkt wird. Dies könnte nützlich sein, um den Spielbereich unabhängig von der tatsächlichen Größe des Spielfeldes in der grafischen Darstellung zu decken.
2. **/ bildschirmhoehe:** Diese Division normalisiert `maus_y` auf einen Wert zwischen 0 und 1, basierend auf der Bildschirmhöhe. Das bedeutet, dass die Position der Maus relativ zur Gesamthöhe des Bildschirms betrachtet wird.
3. **-3:** Diese Konstante verschiebt die skalierte Position nach unten oder oben. Dieser Versatz kann verwendet werden, um den Ursprung der Bewegung des Spielers zu justieren, sodass das Paddel innerhalb des vorgesehenen Spielbereichs bleibt. Die Verschiebung sorgt dafür, dass das Spielerpaddel nicht aus dem sichtbaren Bereich des Spielfelds bewegt wird.

Zusammengefasst passt diese Gleichung die Mausposition so an, dass sie für die Steuerung des Spielers im Pong-Spiel verwendet werden kann, wobei die Skalierung und Verschiebung sicherstellen, dass die Spielerbewegung sowohl responsiv als auch innerhalb des vorgesehenen Bereichs bleibt. Die spezifischen Werte der Konstanten sind so gewählt, dass sie eine gute Spielbarkeit bieten, könnten aber je nach Spielkonfiguration und gewünschtem Schwierigkeitsgrad angepasst werden.

*Alles richtig.*

## 10 Frage

Jetzt möchten Sie noch verhindern, dass der Spieler das Spielfeld verlässt. Untersuchen Sie dazu, ob die aktuelle vertikale Spielerposition größer als die maximale bzw. kleiner als die minimale vertikale Spielfeldkomponente ist und setzen Sie in diesen Fällen die Spielerposition auf die erlaubten Grenzen.

Warum müssen Sie für die obere Grenze auch die Spielerhöhe berücksichtigen?

## 10.1 Antwort

Um zu verhindern, dass der Spieler das Spielfeld verlässt, müssen Sie überprüfen, ob die vertikale Position des Spielers (Paddels) die Grenzen des Spielfeldes überschreitet. Wenn dies der Fall ist, sollten Sie die Position des Spielers an die entsprechenden Grenzen anpassen. Hier ist ein beispielhafter Ansatz, wie Sie dies in Matlab umsetzen können:

```
% Annahme: Untere Grenze ist 0 und obere Grenze ist 1
minPositionY = 0;
maxPositionY = 1 - spielerhoehe; % Berücksichtigung der Spielerhöhe

% Überprüfung und Anpassung an die untere Grenze
if spieler.Position(2) < minPositionY
    spieler.Position(2) = minPositionY;
end

% Überprüfung und Anpassung an die obere Grenze
if spieler.Position(2) > maxPositionY
    spieler.Position(2) = maxPositionY;
end
```

### 10.1.1 Warum müssen Sie für die obere Grenze auch die Spielerhöhe berücksichtigen?

Die Position eines Grafikobjekts in Matlab (und vielen anderen Grafiksystemen) wird üblicherweise durch die Koordinaten seiner unteren linken Ecke definiert. Wenn Sie also die Position des Spielers relativ zum unteren Rand des Spielfeldes setzen, definiert diese Position den unteren Rand des Spielers.

Die Spielfeldgrenzen beziehen sich jedoch auf die sichtbare Spielfläche, die von 0 bis 1 (im normalisierten Koordinatensystem) reicht. Wenn Sie also nur die Position des Spielers berücksichtigen, ohne seine Höhe in die Berechnung einzubeziehen, könnte der obere Teil des Spielers über die obere Grenze des Spielfeldes hinausragen. Dies würde dazu führen, dass der Spieler teilweise außerhalb des sichtbaren Bereichs liegt.

Durch die Berücksichtigung der Spielerhöhe bei der Berechnung der maximal zulässigen vertikalen Position (`maxPositionY = 1 - spielerhoehe`) stellen Sie sicher, dass der gesamte Spieler innerhalb der Grenzen des Spielfeldes bleibt. Die Subtraktion der Spielerhöhe von der oberen Grenze des Spielfeldes (1 im normalisierten Koordinatensystem) gibt die maximal zulässige Position für die untere Ecke des Spielers an, sodass der Spieler vollständig innerhalb des sichtbaren Bereichs bleibt.

*Hat sie verstanden und vernünftig erklärt.*

## 11 Frage

Als nächstes möchten Sie untersuchen, ob der Ball eine der Banden (oder den Schläger) berührt. Dabei möchten Sie folgende Regeln implementieren:

Wenn der Ball die obere Bande berührt, soll seine Vertikalgeschwindigkeitskomponente negiert werden.

Wenn der Ball die untere Bande berührt, soll seine Vertikalgeschwindigkeitskomponente negiert und zusätzlich um eine Zufallszahl zwischen 0 und 1 erhöht werden.

Wenn der Ball die rechte Bande berührt, soll seine Horizontalgeschwindigkeitskomponente negiert und zusätzlich um eine Zufallszahl zwischen 0 und 1 erniedrigt werden.

Wenn der Ball die linke Bande im Bereich des Spielers berührt, soll seine Horizontalgeschwindigkeitskomponente negiert und das String-Attribut des title-Objektes inkrementiert werden. Wenn die linke Bandenberührung außerhalb des Spielers erfolgt, soll das Abbruch-Flag gesetzt werden.

Was bewirkt das Negieren einer Geschwindigkeitskomponente?

Welche Vorzeichen haben die Vertikalgeschwindigkeit vor dem Auftreffen auf die untere Bande und die Horizontalgeschwindigkeit vor dem Auftreffen auf die rechte Bande?

Was bewirkt das Addieren bzw. Subtrahieren der Zufallszahl?

Warum ist die verwendete Umsetzung des Abprallen am Spieler eigentlich nicht ganz korrekt?

## 11.1 Antwort

Das Negieren einer Geschwindigkeitskomponente bedeutet in diesem Kontext die Umkehrung der Bewegungsrichtung des Balls. Wenn der Ball beispielsweise nach oben (positive Vertikalgeschwindigkeit) oder nach rechts (positive Horizontalgeschwindigkeit) fliegt und eine der Banden berührt, führt das Negieren der entsprechenden Geschwindigkeitskomponente dazu, dass der Ball seine Richtung ändert und nach unten bzw. nach links fliegt.

### 11.1.1 Vorzeichen der Geschwindigkeitskomponenten

- Die **Vertikalgeschwindigkeit** vor dem Auftreffen auf die **untere Bande** muss **positiv** sein, da der Ball nach unten fliegt.
- Die **Horizontalgeschwindigkeit** vor dem Auftreffen auf die **rechte Bande** muss **positiv** sein, weil der Ball sich nach rechts bewegt.

### 11.1.2 Addieren bzw. Subtrahieren der Zufallszahl

- Das **Addieren einer Zufallszahl** zur Vertikalgeschwindigkeit beim Aufprallen auf die untere Bande erhöht die Geschwindigkeit des Balls zufällig. Dies fügt dem Spiel ein Element der Unvorhersehbarkeit hinzu, da der Ball nach dem Abprallen von der unteren Bande schneller oder in einem leicht veränderten Winkel zurückprallt.
- Das **Subtrahieren einer Zufallszahl** von der Horizontalgeschwindigkeit beim Aufprallen auf die rechte Bande verringert die Geschwindigkeit des Balls zufällig. Dies kann dazu führen, dass der Ball langsamer wird, was die Rückkehr des Balls zum Spieler möglicherweise erleichtert oder das Spieltempo allgemein variiert.

### 11.1.3 Problematik der Umsetzung des Abprallens am Spieler

Die beschriebene Umsetzung des Abprallens am Spieler (Paddel) berücksichtigt nur die horizontale Position des Balls in Bezug auf das Spielfeld, nicht jedoch die vertikale Position in Relation zur Höhe des Schlägers. Für ein realistischeres Verhalten sollte überprüft werden, ob der Ball nicht nur die linke Bande, sondern auch den vertikalen Bereich des Schlägers trifft.

- **Aktuelle Umsetzung:** Wenn der Ball die linke Bande berührt, wird lediglich überprüft, ob dies im Bereich des Spielers geschieht, ohne die genaue vertikale Position des Balls zu berücksichtigen. Dies könnte zu Situationen führen, in denen der Ball als abgeprallt vom Spieler (Paddel) betrachtet wird, auch wenn er vertikal über oder unter dem Spieler vorbeifliegt.
- **Problem:** Die Umsetzung wird ungenau, wenn der Ball die linke Bande oberhalb oder unterhalb des Paddels berührt, aber dennoch als abgeprallt betrachtet wird, obwohl er in Wirklichkeit am Paddel vorbeigeflogen ist. In einem realistischen Szenario sollte das Spiel enden (oder ein Punkt verloren gehen), wenn der Ball am Paddel vorbeifliegt und die linke Bande berührt.

Für eine präzisere Implementierung müsste also zusätzlich überprüft werden, ob der Ball sich zum Zeitpunkt des Kontakts mit der linken Bande innerhalb der vertikalen Ausdehnung des Spielers (Paddels) befindet.

*Das eigentliche Problem des “Durchdringen” des Schlägers erkennt sie nicht.*

## 12 Frage

Im letzten Schritt berechnen Sie in der Schleife, welche Positionsänderung der Ball in einem Simulationsschritt durchführt, indem Sie den aktuellen Geschwindigkeitsvektor mit der Abtastzeit multiplizieren und das Ergebnis der Variablen `delta_pos` zuweisen.

Ist `delta_pos` ein Vektor oder ein Skalar?

### 12.1 Antwort

Die Positionsänderung `delta_pos`, die durch Multiplikation des aktuellen Geschwindigkeitsvektors `ges` mit der Abtastzeit `delta_t` berechnet wird, ist ein **Vektor**. Der Geschwindigkeitsvektor `ges` enthält sowohl eine Komponente für die horizontale als auch für die vertikale Bewegung des Balls, typischerweise in Form `[vx, vy]`, wobei `vx` die Geschwindigkeit in horizontaler Richtung (x-Richtung) und `vy` die Geschwindigkeit in vertikaler Richtung (y-Richtung) darstellt.

Wenn Sie diesen Geschwindigkeitsvektor mit der Abtastzeit `delta_t` multiplizieren, ergibt sich die Änderung der Position des Balls in beiden Richtungen während eines Simulationsschritts. Daher hat `delta_pos` zwei Komponenten, die die Änderung in x- und y-Richtung angeben, und ist somit ein Vektor.

Die Berechnung könnte wie folgt aussehen:

```
delta_pos = ges * delta_t;
```

Dieser Vektor `delta_pos` gibt dann an, um wie viel sich der Ball in der x- und y-Richtung während des nächsten Simulationsschritts bewegt.

*Yupp.*

## 13 Frage

Jetzt müssen Sie natürlich noch die Ballposition aktualisieren:

```
ball.Position(1 : 2) = ball.Position(1 : 2) + delta_pos;
```

Erläutern Sie die Zeile.

### 13.1 Antwort

Die angegebene Zeile aktualisiert die Position des Balls basierend auf seiner Bewegung seit dem letzten Simulationsschritt. Hier wird erläutert, was in dieser Zeile geschieht:

```
ball.Position(1 : 2) = ball.Position(1 : 2) + delta_pos;
```

- `ball.Position`: In Matlab bezeichnet `Position` eine Eigenschaft des Grafikobjekts, hier des Balls. Für Rechtecke (und der Ball wird als Rechteck mit gleicher Breite und Höhe für ein kreisförmiges Aussehen erzeugt) gibt die `Position`-Eigenschaft die Koordinaten der unteren linken Ecke, die Breite und die Höhe an. Die Position ist also ein Array mit vier Elementen: `[x, y, Breite, Höhe]`.
- `ball.Position(1 : 2)`: Diese Teilanweisung greift auf die ersten beiden Elemente des `Position`-Arrays zu, die die `x`- und `y`-Koordinaten der unteren linken Ecke des Balls (oder in diesem Fall des Kreises) darstellen.
- `delta_pos`: Dies ist der Vektor, der die Änderung der Position des Balls angibt, wie zuvor berechnet. Er enthält zwei Komponenten, eine für die horizontale (`x`) und eine für die vertikale (`y`) Richtung, basierend auf der aktuellen Geschwindigkeit des Balls und der verstrichenen Zeit (`delta_t`).
- `ball.Position(1 : 2) + delta_pos`: Diese Operation addiert den `delta_pos`-Vektor zu den aktuellen `x`- und `y`-Koordinaten der Position des Balls. Das bedeutet, sie verschiebt den Ball um `delta_pos` in der `x-y`-Ebene. Wenn `delta_pos` positive Werte enthält, bewegt sich der Ball nach rechts (für `x`) und nach unten (für `y`); negative Werte bewirken eine Bewegung nach links (für `x`) bzw. nach oben (für `y`).

Durch die Zuweisung des Ergebnisses dieser Operation zurück zu `ball.Position(1 : 2)` wird die Position des Balls im Grafikfenster aktualisiert, sodass der Ball sich entsprechend seiner Geschwindigkeit und der verstrichenen Zeit bewegt. Diese Zeile ist entscheidend für die Animation des Balls im Spiel und ermöglicht es, dass sich der Ball sichtbar über das Spielfeld bewegt.

*Sehr schön erklärt!*

## 14 Frage

Spielen Sie, bis Sie mindestens 10 Punkte erzielt haben.

Highscore? Wie könnten Sie erreichen, dass der Ballgeschwindigkeitszuwachs ein wenig langsamer stattfindet?

### 14.1 Antwort

Da ich nicht tatsächlich ein Spiel spielen oder einen Highscore erzielen kann, können wir uns darauf konzentrieren, wie Sie das Spiel anpassen könnten, um den Ballgeschwindigkeitszuwachs langsamer zu gestalten. Wenn Sie den Ballgeschwindigkeitszuwachs reduzieren möchten, damit der Ball sich langsamer beschleunigt, gibt es mehrere Strategien, die Sie anwenden könnten. Eine Möglichkeit besteht darin, die Zufallszahl, die Sie zur Geschwindigkeit des Balls addieren oder subtrahieren, zu verringern. Hier sind ein paar Ansätze:

### 14.1.1 Skalierung der Zufallszahl

Sie können die Zufallszahl, die zur Geschwindigkeit hinzugefügt oder von ihr subtrahiert wird, mit einem Faktor multiplizieren, der kleiner als 1 ist, um den Effekt zu reduzieren. Zum Beispiel:

```
zufallsZuwachs = rand() * 0.1; % Multiplizieren mit einem Faktor kleiner als 1
ges(2) = ges(2) + zufallsZuwachs; % Für die vertikale Geschwindigkeit
```

In diesem Beispiel wird `rand()` mit 0.1 multipliziert, was den maximalen Zuwachs der Geschwindigkeit auf 10% des ursprünglichen Maximalwerts reduziert. Sie können den Multiplikator anpassen, um den Zuwachs weiter zu verringern oder zu erhöhen.

### 14.1.2 Schrittweise Erhöhung

Anstatt die Geschwindigkeit jedes Mal zu erhöhen, wenn eine Bedingung erfüllt ist, könnten Sie einen Zähler einführen und die Geschwindigkeit nur dann erhöhen, wenn der Zähler einen bestimmten Wert erreicht. Dies könnte dazu führen, dass die Geschwindigkeitserhöhung in größeren Intervallen erfolgt:

```
if mod(punkte, 5) == 0 % Geschwindigkeitserhöhung alle 5 Punkte
    ges(2) = ges(2) + rand() * 0.1;
end
```

### 14.1.3 Begrenzung der maximalen Geschwindigkeit

Eine weitere Möglichkeit, das Spiel zu balancieren, besteht darin, eine maximale Geschwindigkeit festzulegen, sodass der Ball nicht über eine bestimmte Geschwindigkeit hinaus beschleunigt:

```
maxGeschwindigkeit = 3;
if ges(2) < maxGeschwindigkeit
    ges(2) = min(ges(2) + rand() * 0.1, maxGeschwindigkeit);
end
```

Diese Techniken können einzeln oder in Kombination verwendet werden, um die Geschwindigkeitserhöhung des Balls zu kontrollieren und das Spiel angenehmer oder herausfordernder zu gestalten. Die Wahl der Methode hängt von der gewünschten Spielmechanik und dem Schwierigkeitsgrad ab, den Sie erreichen möchten.

*Viele gute Anregungen!*

## 14.2 Frage

Schreibe - basierend auf deinen vorherigen Antworten - ein lauffähiges Programm.

### 14.2.1 Antwort

Unten finden Sie ein einfaches, aber lauffähiges Beispiel eines Pong-Spiels in Matlab, das auf den zuvor besprochenen Konzepten basiert. Bitte beachten Sie, dass dieses Beispiel einige grundlegende Spielmechaniken implementiert, aber für ein vollständiges Spielerlebnis möglicherweise weiter angepasst werden muss.

```
function einfachesPongSpiel
    % Fenster und Spielfeld initialisieren
```

```

fig = figure('NumberTitle', 'off', 'MenuBar', 'none', 'Name', 'Pong', 'Visible', 'on');
ax = axes('Parent', fig, 'Position', [0 0 1 1], 'XLim', [0 1], 'YLim', [0 1], 'Color', 'b');
ax.XAxis.Visible = 'off';
ax.YAxis.Visible = 'off';
hold(ax, 'on');

% Titel und Initialisierung
titel = title(ax, '0');
delta_t = 0.001;
game_over = false;
ges = [1 -1] * 0.01; % Geschwindigkeit des Balls

% Spieler-Paddel
spielerhoehe = 0.2;
spieler = rectangle('Position', [0.05, 0.5 - spielerhoehe / 2, 0.01, spielerhoehe], 'Face', 'b');

% Ball
ball = rectangle('Position', [0.5 - 0.015, 0.5 - 0.015, 0.03, 0.03], 'Curvature', [1, 1], 'Color', 'b');

% Spielerscore
score = 0;

% Hauptschleife
while ~game_over
    % Spielerposition aktualisieren basierend auf Mausposition
    bildschirmgroesse = get(groot, 'ScreenSize');
    bildschirmhoehe = bildschirmgroesse(4);
    mausPosition = get(groot, 'PointerLocation');
    spieler.Position(2) = (5 * mausPosition(2) / bildschirmhoehe - 3) * 0.1;

    % Ballbewegung
    delta_pos = ges * delta_t;
    ball.Position(1:2) = ball.Position(1:2) + delta_pos;

    % Kollisionserkennung und -reaktion
    if ball.Position(2) <= 0 % Untere Bande
        ges(2) = -ges(2) + rand() * 0.01;
    elseif ball.Position(2) + ball.Position(4) >= 1 % Obere Bande
        ges(2) = -ges(2);
    end

    if ball.Position(1) + ball.Position(3) >= 1 % Rechte Bande
        ges(1) = -ges(1) - rand() * 0.01;
    elseif ball.Position(1) <= 0 % Linke Bande
        if ball.Position(2) >= spieler.Position(2) && ball.Position(2) <= spieler.Position(2) + spielerhoehe
            ges(1) = -ges(1);
            score = score + 1;
            titel.String = num2str(score);
        end
    end
end

```

```

        else
            game_over = true;
        end
    end
end

% Prüfung, ob Spiel fortgesetzt werden soll
pause(delta_t);
drawnow;

if score >= 10 % Spielende nach 10 Punkten
    game_over = true;
end
end

% Spielende
if score < 10
    disp('Game Over!');
else
    disp(['Congratulations! Your final score is ', num2str(score)]);
end
end
end

```

Dieses Programm skizziert ein sehr grundlegendes Pong-Spiel, bei dem ein Spieler versucht, den Ball mit einem Paddel abzufangen, das er mit der Maus steuert. Der Ball prallt von den oberen und unteren Rändern des Spielfelds ab, und das Ziel ist es, zu verhindern, dass er den linken Rand passiert. Wenn der Spieler es schafft, den Ball mit dem Paddel abzufangen, wird der Ball zurück ins Spielfeld geschleudert, und der Spieler erhält einen Punkt. Das Spiel endet, wenn der Spieler 10 Punkte erreicht oder wenn der Ball den linken Rand des Spielfeld

*Läuft natürlich auch nicht vernünftig auf Anhieb.*

# 10 Game of Life

April 11, 2024

## 1 Frage

Sie möchten im Folgenden die in der Vorlesung vorgestellte Game-of-Life-Anwendung erstellen. Öffnen Sie dazu eine neue App, speichern Sie sie unter dem Namen Conway und ziehen Sie aus der Komponentenbibliothek drei Schaltflächen, zwei numerische Eingabefelder und ein Achsensystem in die Anwendung. Benennen Sie die Schaltflächen und Eingabefelder mit den Namen aus der Vorlesung und geben Sie ihnen die folgenden Eigenschaften:

Breite:

Value: 100

Limits/min: 10

Limits/max: 1000

ValueDisplayFormat: Integer

Pause:

Value: 0

Limits/min: -Inf

Limits/max: Inf

ValueDisplayFormat: Default

Löschen Sie den Titel des Achsensystems und ordnen Sie die Objekte bezüglich ihrer Abmessung und Position gemäß der Vorlesung an.

Was bedeutet die Default-Angabe beim Pause-Eingabefeld?

### 1.1 Antwort

Die “Default”-Angabe bei dem Pause-Eingabefeld in Ihrer App-Entwicklung bezieht sich auf die Formatierung des angezeigten Werts. Wenn “Default” als Wert für das Eigenschaftsfeld “ValueDisplayFormat” festgelegt ist, bedeutet dies, dass keine spezielle Formatierung auf den Wert angewendet wird, der im Eingabefeld angezeigt wird. Das System entscheidet automatisch, wie der Wert basierend auf seinem Typ und Wert dargestellt wird. Im Gegensatz dazu würde eine Angabe wie “Integer” eine spezifische Anweisung geben, den Wert als ganze Zahl darzustellen, unabhängig davon, ob der ursprüngliche Wert eine Dezimalzahl ist oder nicht.

In praktischer Anwendung sorgt die Default-Einstellung dafür, dass die Benutzeroberfläche flexibel bleibt. Für das Pause-Feld bedeutet dies, dass der eingegebene Wert in der natürlichen Form

angezeigt wird, die der Benutzer eingibt, ohne eine erzwungene Umwandlung in ein spezifisches Zahlenformat. So könnte beispielsweise eine Eingabe von “0.5” oder “1” genau so im Feld erscheinen, ohne in ein anderes Format wie eine ganze Zahl konvertiert zu werden. Diese Flexibilität kann besonders nützlich sein, wenn die präzise Natur des Werts wichtig ist und keine Annahmen über die gewünschte Darstellung gemacht werden sollen.

*Richtig.*

## 2 Frage

Wechseln Sie in den Code View und erstellen Sie im Code Browser unter Properties ein neues Attribut mit dem Namen `game_over`, das während des Spiels von allen Methoden gelesen und geschrieben werden kann.

Wofür wird `game_over` wohl genutzt werden?

### 2.1 Antwort

Das Attribut `game_over` in einem Spiel, wie etwa dem “Game of Life” oder anderen interaktiven Anwendungen, wird typischerweise dazu verwendet, den Zustand des Spiels zu überwachen, insbesondere ob das Spiel zu Ende ist oder nicht. Ein boolescher Wert (wahr/falsch) wird oft für ein solches Attribut verwendet, wobei `true` (wahr) bedeutet, dass das Spiel beendet ist, und `false` (falsch), dass das Spiel noch läuft.

Die Nutzung eines `game_over`-Attributs ermöglicht es verschiedenen Teilen des Programms, auf einfache Weise zu prüfen, ob das Spiel fortgesetzt werden soll. Methoden innerhalb des Spiels könnten diese Variable überprüfen, bevor sie Logik ausführen, die nur relevant ist, wenn das Spiel noch aktiv ist. Zum Beispiel:

- **Update-Methoden:** Diese Methoden aktualisieren den Spielzustand, die Position der Spielfiguren oder die Spiellogik. Wenn `game_over` wahr ist, könnten diese Updates pausiert oder übersprungen werden, um Ressourcen zu sparen oder um zu verhindern, dass das Spiel nach seinem offiziellen Ende weiterläuft.
- **Zeichenmethoden:** Methoden, die für die grafische Darstellung des Spiels verantwortlich sind, könnten das `game_over`-Attribut überprüfen, um zu entscheiden, ob ein Spiel-Ende-Bildschirm angezeigt werden soll.
- **Ereignishandhabung:** Event-Handler, die auf Benutzereingaben reagieren, könnten ebenfalls anders funktionieren oder bestimmte Aktionen verhindern, wenn das Spiel vorbei ist.

Das `game_over`-Attribut dient also als zentraler Anhaltspunkt für den Spielzustand, um konsistentes Verhalten über das gesamte Programm hinweg zu gewährleisten und um sicherzustellen, dass das Spiel korrekt beendet wird, wenn die Bedingungen für das Spielende erfüllt sind.

*Die Antwort geht deutlich über die Erwartungen hinaus.*

## 3 Frage

Fügen Sie (ebenfalls im Code Browser) der App ein `startupFcn`-Callback-Unterprogramm hinzu, das dadurch automatisch als Methode erzeugt wird und mit einer Leerzeile darauf wartet, von Ihnen mit Leben gefüllt zu werden.

Wann wird `startupFcn` aufgerufen?

### 3.1 Antwort

Das `startupFcn`-Callback-Unterprogramm, das in einer App-Entwicklungsumgebung hinzugefügt wird, ist speziell dafür konzipiert, unmittelbar nach dem Start der App aufgerufen zu werden. Dieses Callback dient als Initialisierungsroutine und wird automatisch einmalig ausgeführt, sobald die Benutzeroberfläche der App vollständig geladen und angezeigt wird, aber bevor der Benutzer mit der App interagieren kann.

Die Verwendung von `startupFcn` bietet eine effektive Möglichkeit, verschiedene Setup-Operationen durchzuführen, wie zum Beispiel:

- Initialisieren von App-Variablen oder Zuständen,
- Vorladen von Daten oder Ressourcen, die die App benötigt,
- Einstellen von UI-Komponenten auf spezifische Zustände, wie das Aktivieren oder Deaktivieren von Buttons,
- Anzeigen von Willkommensnachrichten oder Einführungshinweisen für den Benutzer,
- Überprüfen der Systemkonfiguration oder externer Abhängigkeiten, um sicherzustellen, dass die App korrekt funktionieren kann.

Das `startupFcn`-Callback stellt somit einen zentralen Ort dar, an dem alle initialen Einstellungen vorgenommen werden können, um die App in einen definierten Anfangszustand zu versetzen, bevor der Nutzer beginnt, mit ihr zu interagieren.

*Wunderbar.*

## 4 Frage

Im Unterprogramm setzen Sie mit

```
axis (app.UIAxes, 'tight', 'off', 'square')
```

ein paar Eigenschaften des Achsensystems.

Was bewirken die Parameter?

Warum können Sie das Achsensystem jetzt nicht mehr sehen, wenn Sie die App anstarten?

### 4.1 Antwort

Die Funktion `axis` in MATLAB und ähnlichen Umgebungen wird verwendet, um die Eigenschaften der Achsen in einem Plot anzupassen. Die spezifischen Parameter, die Sie genannt haben (`'tight'`, `'off'`, `'square'`), steuern das Aussehen und Verhalten der Achsen in Ihrer App. Hier ist, was jeder dieser Parameter bewirkt:

- `'tight'`: Passt die Achsenlimits so an, dass sie genau um die Daten im Plot liegen, mit minimalen Rändern um die Datenpunkte. Das bedeutet, dass die Achsen sich so eng wie möglich an die tatsächlich dargestellten Daten anpassen.
- `'off'`: Schaltet die Anzeige der Achsenbeschriftungen, Ticks und Linien ab. Es werden keine Achsenlinien, Zahlen oder Ticks auf oder um den Plot herum angezeigt.

- **'square'**: Macht das Achsen-Box-Verhältnis quadratisch. Das bedeutet, dass die Höhe und Breite des Plotbereichs gleich sind, unabhängig von den tatsächlichen Datenverhältnissen oder den Dimensionen des UI-Elements, das den Plot enthält. Dies kann dazu führen, dass der Plot in einigen Fällen gestreckt oder gestaucht erscheint, um dieses quadratische Verhältnis zu erfüllen.

Warum Sie das Achsensystem nicht mehr sehen können:

Wenn Sie die App starten und `axis (app.UIAxes, 'tight', 'off', 'square')` ausführen, bewirkt der Parameter `'off'`, dass keine Achsenlinien, Zahlen oder Ticks angezeigt werden. Das bedeutet, selbst wenn Daten geplottet sind oder das Achsensystem an sich existiert, wird es visuell nicht im UI dargestellt, da alle visuellen Hinweise auf das Achsensystem deaktiviert wurden. Falls keine Daten geplottet sind oder die geplotteten Daten nicht sichtbar sind (z.B. außerhalb des anfänglichen sichtbaren Bereichs liegen), könnte es scheinen, als ob das Achsensystem komplett fehlt, da keinerlei Achsenbeschriftungen oder -markierungen angezeigt werden.

*Schöne Erklärungen.*

## 5 Frage

Die nächste Zeile im Unterprogramm lautet:

```
colormap (app.UIAxes, gray (2))
```

Welche Einträge hat die Farbtabelle des Achsensystems jetzt?

Welchen Farben entspricht das?

### 5.1 Antwort

Die Funktion `colormap(app.UIAxes, gray(2))` legt die Farbtabelle (`colormap`) für das spezifizierte Achsensystem in Ihrer App fest, in diesem Fall `app.UIAxes`. Die Funktion `gray(2)` erzeugt eine Graustufen-Farbtabelle mit 2 Einträgen.

Die generierte Farbtabelle enthält zwei Farben:

1. Schwarz, repräsentiert durch den RGB-Wert  $[0, 0, 0]$ .
2. Weiß, repräsentiert durch den RGB-Wert  $[1, 1, 1]$ .

In einer Graustufen-Farbtabelle entsprechen die Werte zwischen 0 (schwarz) und 1 (weiß) verschiedenen Schattierungen von Grau. Da `gray(2)` eine sehr einfache Farbtabelle mit nur zwei Einträgen erzeugt, besteht sie nur aus den extremen Enden des Graustufen-Spektrums: Schwarz und Weiß, ohne Zwischentöne von Grau.

Diese Art von Farbtabelle wird oft in binären Bildern verwendet, wo Pixel entweder als aktiv (üblicherweise weiß) oder inaktiv (schwarz) markiert sind, ohne Notwendigkeit für Grautöne dazwischen.

*Perfekt.*

## 6 Frage

Mit dem Befehl

```
image (app.UIAxes, uint8 (round (rand (100))))
```

erzeugen Sie ein Zufallsbild im Achsensystem.

Wie viele Pixel hat das dargestellte Bild?

Welche Farben besitzt es? Warum eigentlich?

Warum ist es sinnvoll, das Spielfeld als `uint8` zu initialisieren?

## 6.1 Antwort

Der Befehl `image(app.UIAxes, uint8(round(rand(100))))` erzeugt ein Bild im Achsensystem `app.UIAxes` Ihrer App, das aus 100x100 Pixeln besteht. Die Funktion `rand(100)` generiert eine 100x100-Matrix mit Zufallswerten zwischen 0 und 1. Durch die Anwendung der Funktion `round` werden diese Zufallswerte auf die nächstliegenden ganzen Zahlen gerundet, was in diesem Fall entweder 0 oder 1 sein wird. Die Konversion zu `uint8` (unsigned 8-bit integer) wandelt die Zahlen in 8-Bit Ganzzahlen um, was bedeutet, dass jede Zahl im Bereich von 0 bis 255 liegen kann. Da jedoch die gerundeten Werte nur 0 oder 1 sind, enthält die resultierende Matrix ausschließlich die Werte 0 und 1.

Bezüglich der Farben:

- Die `colormap` wurde zuvor mit `colormap(app.UIAxes, gray(2))` festgelegt, was bedeutet, dass es nur zwei Farben gibt, denen die Werte zugeordnet werden können: Schwarz (`[0,0,0]`) für 0 und Weiß (`[1,1,1]`) für 1.
- Aufgrund der Farbtabelle (`colormap`) und der Daten in der Bildmatrix (die nur 0 und 1 enthält), werden die Pixel des Bildes nur in Schwarz oder Weiß angezeigt.

Die Initialisierung des Spielfelds (oder in diesem Fall des Bildes) als `uint8` ist aus mehreren Gründen sinnvoll:

- **Speichereffizienz:** `uint8` nimmt weniger Speicherplatz ein als Standard-Datentypen wie `double` (der Standard-Datentyp für Zahlen in MATLAB). Da `uint8` für die Darstellung von Werten im Bereich von 0 bis 255 nur 8 Bit (1 Byte) pro Element verwendet, ist es besonders effizient für Bilder oder Spielfelder, die nur eine begrenzte Anzahl an Werten benötigen.
- **Performance:** Operationen mit `uint8`-Arrays können schneller sein als mit Arrays von Typ `double`, besonders bei der Bildverarbeitung oder der Aktualisierung von Spielfeldern, da weniger Daten verarbeitet und im Speicher gehalten werden müssen.
- **Kompatibilität:** Viele Bildverarbeitungsfunktionen und Grafikbibliotheken erwarten Daten im `uint8`-Format, insbesondere wenn es um die Darstellung von Bildern geht, da dies ein gängiges Format für die Speicherung und Manipulation von Bildpixeln ist.

Zusammengefasst ermöglicht dieser Ansatz eine effiziente Darstellung und Verarbeitung von binären oder Graustufenbildern mit minimalem Speicherbedarf und guter Performance.

*Klasse!*

## 7 Frage

Lassen Sie den App Designer die Callback-Methode der Zufall-Schaltfläche erzeugen und lesen Sie in der Methode als erstes den Wert des Breite-Eingabefeldes in die Variable `breite`.

Mit welchem Attribut des Eingabefeldes kommen Sie an seinen Wert heran?

### 7.1 Antwort

Um den Wert des Eingabefeldes in eine Variable zu lesen, nutzen Sie das `Value`-Attribut des Eingabefeld-Objekts. Wenn das Eingabefeld beispielsweise `app.Breite` heißt, würden Sie den Wert mit folgendem Code in die Variable `breite` lesen:

```
breite = app.Breite.Value;
```

Hierbei ist `app.Breite` der Name des numerischen Eingabefeldes in Ihrer App, und `.Value` ist das Attribut, das den aktuellen Wert dieses Feldes enthält. Dieser Wert wird dann der Variable `breite` zugewiesen.

*Genau.*

## 8 Frage

Initialisieren Sie das Spielfeld mit folgendem Befehl:

```
spielfeld(breite, breite) = uint8(0);
```

Wie viele Zeilen und Spalten hat das Spielfeld jetzt und welche Werte haben seine Elemente?

### 8.1 Antwort

Der Befehl `spielfeld(breite, breite) = uint8(0);` initialisiert das Spielfeld als eine Matrix in MATLAB, indem es das Element in der durch `breite` bestimmten Zeile und Spalte direkt zu `uint8(0)` setzt. Wenn `breite` beispielsweise 10 ist, wird das Element in der 10. Zeile und 10. Spalte der Matrix auf den Wert 0 vom Typ `uint8` gesetzt.

MATLAB erweitert die Matrix automatisch, um diesen Index zu berücksichtigen. Das bedeutet, dass alle Elemente von `spielfeld`, die nicht explizit gesetzt wurden, den Standardwert 0 erhalten, und zwar ebenfalls als `uint8`-Typ, da die Matrix durch die Zuweisung eines `uint8`-Wertes diesen Typ annimmt.

Daher hat das Spielfeld jetzt `breite` Zeilen und `breite` Spalten, wobei jedes Element den Wert 0 hat. Wenn `breite` zum Beispiel 10 ist, dann hat das Spielfeld 10 Zeilen und 10 Spalten, und jedes der 100 Elemente in der Matrix hat den Wert 0 vom Typ `uint8`.

*Keine Beanstandungen.*

## 9 Frage

Füllen Sie "bestimmte" Elemente mit "bestimmten" Werten:

```
spielfeld(2 : breite - 1, 2 : breite - 1) = ... round(rand(breite - 2));
```

Erläutern Sie ausführlich, warum welche Elemente des Spielfeldes jetzt welche Werte besitzen.

## 9.1 Antwort

Der Befehl

```
spielfeld(2 : breite - 1, 2 : breite - 1) = round(rand(breite - 2));
```

füllt einen bestimmten Bereich innerhalb des zuvor initialisierten Spielfelds mit Werten, die durch Runden der Ergebnisse der Funktion `rand(breite - 2)` entstehen. Lassen Sie uns diesen Befehl Schritt für Schritt durchgehen, um zu verstehen, was genau passiert:

### 9.1.1 Indexierung

- `2 : breite - 1` spezifiziert einen Bereich von Indizes für Zeilen und Spalten. Es startet bei Index 2 und geht bis zum Index `breite - 1`. Das bedeutet, dass die äußerste Randreihe oben, unten, links und rechts des Spielfelds nicht verändert wird, da die Indexierung bei 2 beginnt und bei `breite - 1` aufhört. Dieser Bereich bildet ein Subfeld oder eine kleinere Matrix innerhalb der größeren Spielfeldmatrix.

### 9.1.2 `rand(breite - 2)`

- `rand(breite - 2)` erzeugt eine Matrix der Größe  $(breite - 2) \times (breite - 2)$  mit Zufallswerten zwischen 0 und 1. Diese Dimensionen passen genau in den durch `2 : breite - 1` spezifizierten Bereich des Spielfelds, da durch das Auslassen der äußeren Ränder (1. und letzte Zeile/Spalte) das zentrale Subfeld um 2 Zeilen und 2 Spalten kleiner als das gesamte Spielfeld ist.

### 9.1.3 `round`

- Die Funktion `round` wird auf jede Zahl in der Zufallsmatrix angewendet. Da `rand` Werte zwischen 0 und 1 liefert, führt das Runden dazu, dass jede Zahl zu 0 oder 1 wird. Ein Wert von 0,5 oder höher wird zu 1 gerundet, und ein Wert unter 0,5 wird zu 0 gerundet.

### 9.1.4 Zuweisung

- Die gerundete Matrix wird dann dem zuvor durch `2 : breite - 1, 2 : breite - 1` spezifizierten Bereich des Spielfelds zugewiesen. Das bedeutet, dass das innere Subfeld des Spielfelds (alles außer dem äußersten Rand) mit gerundeten Zufallszahlen (0 oder 1) gefüllt wird.

### 9.1.5 Ergebnis

Das Endergebnis dieser Operation ist, dass die äußerste Randreihe und -spalte des Spielfelds unberührt bleiben und den Wert 0 behalten, während das innere Subfeld des Spielfelds nun eine Mischung aus 0 und 1 enthält, basierend auf dem Ergebnis der Zufallserzeugung und des Rundens. Diese Verteilung von 0 und 1 kann für verschiedene Simulationen oder Spiele innerhalb der App verwendet werden, z.B. als Initialzustand für ein "Game of Life"-Szenario.

*Sehr gut.*

## 10 Frage

Wenn Sie an dieser Stelle schon einmal überprüfen wollen, wie Ihr Spielfeld tatsächlich aussieht, können Sie den folgenden Befehl temporär einfügen

```
app.UIAxes.Children(1).CData = spielfeld
```

und probierhalber mehrfach die Zufall-Schaltfläche anklicken.

Versuchen Sie genau zu verstehen, was der Befehl bewirkt.

Was sprechen Sie mit `app.UIAxes` an?

Warum verwenden Sie `Children`?

Was bewirkt die eins als Parameter von `Children`?

Was sind `CData`?

### 10.1 Antwort

Der Befehl `app.UIAxes.Children(1).CData = spielfeld` wird in der Entwicklungsumgebung einer MATLAB App verwendet, um den Inhalt des Spielfelds in einem grafischen Achsensystem (`UIAxes`) der App darzustellen. Lassen Sie uns diesen Befehl Schritt für Schritt analysieren, um seine Funktionsweise zu verstehen:

#### 10.1.1 `app.UIAxes`

- `app.UIAxes` bezieht sich auf ein Achsensystem innerhalb der App, das Sie im App Designer definiert haben. Dieses Achsensystem ist der Container, in dem grafische Objekte wie Plots, Bilder und andere grafische Darstellungen gezeigt werden.

#### 10.1.2 `.Children`

- `.Children` ist eine Eigenschaft des Achsensystems, die eine Liste aller grafischen Objekte (Kinder) enthält, die zu diesem Achsensystem gehören. Diese Objekte können Linien, Bilder, Patches, Texte und andere grafische Elemente umfassen, die in den Achsen enthalten sind.

#### 10.1.3 `Children(1)`

- Durch Hinzufügen von `(1)` hinter `Children` greifen Sie auf das erste (oder einzige) Kindobjekt im Achsensystem zu. Die Indizierung beginnt bei 1, was bedeutet, dass `Children(1)` das erste grafische Objekt in der Liste der Kinder bezeichnet. Die Annahme hier ist, dass es sich um ein Bildobjekt handelt, das die Daten (`CData`) darstellt, welche Sie aktualisieren möchten.

#### 10.1.4 `.CData`

- `CData` steht für “Color Data” und ist eine Eigenschaft bestimmter grafischer Objekte in MATLAB, insbesondere von Bildobjekten (`image` Objekte). Es handelt sich um eine Matrix, die die Farbinformationen für jedes Pixel des Bildes enthält. Durch Ändern der `CData`-Eigenschaft eines Bildobjekts können Sie dessen visuelle Darstellung aktualisieren.

### 10.1.5 Zusammenfassung des Befehls

Der Befehl `app.UIAxes.Children(1).CData = spielfeld` aktualisiert also die Farbdaten (`CData`) des ersten Kindobjekts (angenommen, es ist ein Bild) im Achsensystem `UIAxes` Ihrer App mit den Werten in der Matrix `spielfeld`. Das Ergebnis ist, dass das Bild im `UIAxes`-Achsensystem neu gezeichnet wird, um den aktuellen Zustand des Spielfelds zu reflektieren.

Wenn Sie diesen Befehl mehrfach ausführen (z.B. durch wiederholtes Klicken auf eine Schaltfläche, die das Spielfeld neu generiert), sehen Sie, wie sich das Bild basierend auf den jeweils neuen Zufallswerten im `spielfeld` ändert. Dies ist eine effektive Methode, um dynamische Daten oder Zustände in einer grafischen Benutzeroberfläche darzustellen.

*Ich könnte es nicht besser.*

## 11 Frage

Initialisieren Sie das Anwendungsattribut `app.game_over` als falsch und rufen Sie mit dem Befehl `schleife(app, spielfeld)`

die Simulationsschleife auf.

Warum übergeben Sie dem Unterprogramm `schleife` auch das gesamte `app`-Objekt?

### 11.1 Antwort

Das Anwendungsattribut `app.game_over` als `false` zu initialisieren und das gesamte `app`-Objekt zusammen mit dem `spielfeld` an das Unterprogramm `schleife` zu übergeben, ist ein gängiges Muster in der Entwicklung von Apps, insbesondere wenn diese auf einem Event-Driven-Design basieren, wie es bei MATLAB-Apps der Fall ist. Hier sind einige Gründe für diesen Ansatz:

#### 11.1.1 Zentralisierte Zustandsverwaltung

- **Zugriff auf App-Zustand:** Das `app`-Objekt enthält den Zustand der gesamten Anwendung, einschließlich UI-Komponenten und jeglicher Anwendungslogik, die als Attribute und Methoden definiert ist. Das Übergeben des gesamten `app`-Objekts ermöglicht es dem Unterprogramm `schleife`, auf alle Teile der Anwendung zuzugreifen, ihren Zustand zu lesen und zu modifizieren. Dies ist besonders nützlich, um auf UI-Komponenten wie Textfelder, Achsen für Plots und Statusanzeigen zugreifen zu können.

#### 11.1.2 Interaktion mit der Benutzeroberfläche

- **UI-Updates:** Während der Ausführung der Simulationsschleife müssen möglicherweise UI-Komponenten aktualisiert werden, um den Fortschritt der Simulation zu reflektieren, z.B. durch Aktualisierung von Plots im `UIAxes`-Objekt oder Anzeigen von Nachrichten. Durch das Übergeben des `app`-Objekts kann die `schleife`-Funktion solche Updates direkt vornehmen.

#### 11.1.3 Kontrolle über den Simulationsablauf

- **Steuerung des Simulationsstatus:** Das Attribut `app.game_over` dient als ein Kontrollmechanismus für die Simulationsschleife. Die `schleife`-Funktion kann dieses Attribut überprüfen, um zu bestimmen, ob die Simulation weiterlaufen soll oder nicht. Zusätzlich

ermöglicht das `app`-Objekt der Simulationsschleife, dieses Attribut zu ändern, beispielsweise basierend auf bestimmten Ereignissen oder Bedingungen, die das Ende der Simulation signalisieren.

#### 11.1.4 Flexibilität und Erweiterbarkeit

- **Erweiterbarkeit:** Das Übergeben des gesamten `app`-Objekts macht die `schleife`-Funktion flexibler und erleichtert zukünftige Erweiterungen der App. Wenn neue Features oder UI-Komponenten hinzugefügt werden, steht die entsprechende Funktionalität und der Zustand direkt in der `schleife`-Funktion zur Verfügung, ohne dass die Schnittstelle der Funktion geändert werden muss.

Zusammenfassend lässt sich sagen, dass das Übergeben des gesamten `app`-Objekts an das Unterprogramm `schleife` eine praxisnahe Lösung bietet, um komplexe Interaktionen innerhalb der App zu handhaben und gleichzeitig eine saubere und flexible Architektur zu gewährleisten.

*Also, programmieren kann sie wirklich!*

## 12 Frage

Erstellen Sie im Code Browser unter Functions, das neue Unterprogramm mit dem Namen `schleife`, das eine `while`-Schleife enthält, die abbricht, wenn die Variable `app.game_over` gesetzt wurde.

In der Schleife rufen Sie jetzt als Erstes den schon besprochenen Befehl

```
app.UIAxes.Children(1).CData = spielfeld
```

auf, um das (aktuelle) Spielfeld in jedem Simulationsschritt darzustellen.

Warum ist es sehr viel sinnvoller, in der Simulationsschleife die `CData` des vorhandenen Bildes zu überschreiben als jedesmal ein neues `image` zu erzeugen?

### 12.1 Antwort

Die wiederholte Aktualisierung der `CData` (Color Data) eines vorhandenen Bildobjekts in einer Simulationsschleife, anstatt bei jedem Schritt ein neues Bildobjekt mit dem `image`-Befehl zu erzeugen, hat mehrere bedeutende Vorteile in Bezug auf Leistung, Effizienz und die Verwaltung des Speichers. Hier sind die Hauptgründe dafür:

#### 12.1.1 Leistung und Effizienz

- **Reduzierter Rechenaufwand:** Das Erzeugen eines neuen Bildobjekts bei jedem Schritt der Simulation kann rechenintensiv sein, da es nicht nur das Rendern des Bildes, sondern auch die Initialisierung des Bildobjekts und das Hinzufügen zum Achsensystem umfasst. Das einfache Überschreiben der `CData` eines vorhandenen Bildobjekts hingegen erfordert weniger Rechenleistung, da das Bildobjekt selbst nicht neu initialisiert werden muss und keine Änderungen an der Objektstruktur des Achsensystems vorgenommen werden müssen.
- **Schnellere Aktualisierungen:** Das direkte Aktualisieren der `CData` führt zu einer schnelleren Bildaktualisierung, was für eine flüssige Darstellung der Simulation entscheidend sein kann. Dies ist besonders wichtig in Echtzeitanwendungen oder bei Simulationen, die eine hohe Bildwiederholrate erfordern.

### 12.1.2 Speichermanagement

- **Reduzierter Speicherverbrauch:** Jedes Mal, wenn ein neues Bildobjekt erzeugt wird, verbraucht es zusätzlichen Speicher. Im Laufe der Zeit kann dies zu einem erheblichen Speicherverbrauch führen, insbesondere bei lang laufenden oder hochfrequenten Simulationen. Das Überschreiben der `CData` vermeidet diesen zusätzlichen Speicherverbrauch, da keine neuen Objekte erstellt werden.
- **Vermeidung von Speicherlecks:** Bei der kontinuierlichen Erzeugung neuer Bildobjekte ohne die ordnungsgemäße Entfernung der alten Objekte aus dem Achsensystem kann es zu Speicherlecks kommen. Selbst wenn Objekte nicht mehr benötigt werden, können sie im Speicher verbleiben, wenn sie nicht explizit gelöscht werden. Das Überschreiben der `CData` bestehender Objekte umgeht dieses Problem.

### 12.1.3 Simplifizierte Code-Struktur

- **Einfachere Code-Wartung:** Der Code, der lediglich die `CData` eines bestehenden Objekts aktualisiert, ist in der Regel einfacher und direkter als Code, der ständig neue Objekte erzeugt und alte Objekte löscht. Dies kann die Wartung und das Debugging des Codes erleichtern.

Insgesamt sorgt das Überschreiben der `CData` eines vorhandenen Bildobjekts für eine effizientere, performantere und speicherschonendere Implementierung von dynamischen grafischen Darstellungen in Apps und Simulationen.

*Begeisternd.*

## 13 Frage

Damit Sie Ihre Simulation nicht immer mit Strg-c im Kommandofenster abbrechen müssen, machen wir jetzt einen kleinen Exkurs zur Stop-Funktion und kehren dann wieder zur Simulationsschleife zurück. Lassen Sie den App Designer die Callback-Methode der Stop-Schaltfläche erzeugen und setzen Sie in der Methode die Variable `app.game_over` auf den Wert `true`.

Was passiert, wenn Sie jetzt während der Simulation auf die Stop-Schaltfläche klicken?

### 13.1 Antwort

Wenn Sie während der Simulation auf die Stop-Schaltfläche klicken und die zugehörige Callback-Methode die Variable `app.game_over` auf den Wert `true` setzt, geschieht Folgendes:

1. **Änderung des Spielzustands:** Die Variable `app.game_over` dient als Steuerungsflagge für den Fortlauf der Simulation. Indem sie auf `true` gesetzt wird, signalisieren Sie der Simulationsschleife, dass die Simulation beendet werden soll.
2. **Beendigung der Simulationsschleife:** In Ihrer Simulationsschleife gibt es eine Bedingung, die überprüft, ob `app.game_over` `true` ist. Sobald diese Bedingung erfüllt ist – was der Fall ist, nachdem Sie auf die Stop-Schaltfläche geklickt und `app.game_over` entsprechend gesetzt haben – wird die Schleife beendet. Das bedeutet, dass keine weiteren Simulationsschritte ausgeführt werden und die Aktualisierung des Spielfelds (oder andere Aktionen innerhalb der Schleife) gestoppt wird.

3. **Reaktionsfähigkeit der Benutzeroberfläche bleibt erhalten:** Im Gegensatz zum Abbrechen der Ausführung mit Strg-C im Kommandofenster, welches abrupt jegliche Ausführung im MATLAB-Fenster stoppt, ermöglicht die Nutzung einer solchen Steuerungsflagge ein kontrolliertes Beenden der Simulation. Die Benutzeroberfläche bleibt reaktionsfähig, und Sie können nach dem Beenden der Simulation weitere Aktionen in Ihrer App ausführen, ohne dass die Anwendung oder MATLAB neu gestartet werden muss.
4. **Möglichkeit zur Bereinigung:** Da die Simulation geordnet beendet wird, haben Sie die Möglichkeit, nach dem Setzen von `app.game_over` auf `true` und vor dem endgültigen Verlassen der Schleife, zusätzliche Aufräumarbeiten durchzuführen. Dies könnte das Zurücksetzen bestimmter UI-Elemente, das Speichern von Endzuständen oder das Anzeigen von Abschlussnachrichten umfassen.

Zusammenfassend ermöglicht Ihnen die Implementierung einer Stop-Funktion, die Kontrolle über den Ablauf Ihrer Simulation zu behalten und diese auf benutzerfreundliche Weise zu beenden, wodurch die Anwendung robuster und benutzerfreundlicher wird.

*Sehr schön.*

## 14 Frage

Kehren Sie zurück zur Simulationsschleife. Verwenden Sie den Befehl `pause` in der Schleife, um die Simulation langsamer ablaufen zu lassen. Lesen Sie dazu den Wert aus dem entsprechenden Eingabefeld aus und übergeben Sie ihn dem Pausebefehl. Setzen Sie den Pausewert probeweise auf 1 und überprüfen Sie, ob Ihre Zufallszahlen im Kommandofenster jetzt tatsächlich langsamer tröpfeln.

Welche Echtzeiteinheit hat der Pausebefehl? Was bedeutet also die 1 im Eingabefeld?

### 14.1 Antwort

Der Befehl `pause` in MATLAB setzt die Ausführung des aktuellen Scripts oder der aktuellen Funktion für eine spezifische Zeitdauer aus. Die Echtzeiteinheit für den `pause`-Befehl ist Sekunden. Das bedeutet, dass wenn Sie `pause(1)` in Ihrer Simulationsschleife verwenden, die Ausführung der Schleife bei jedem Durchlauf für eine Sekunde pausiert wird.

Eine Eingabe von 1 im zugehörigen Eingabefeld und die Übergabe dieses Werts an den `pause`-Befehl bedeutet also, dass zwischen den einzelnen Schritten der Simulation jeweils eine Sekunde Pause gemacht wird. Dies führt dazu, dass die Aktualisierungen der Simulation (zum Beispiel die Änderungen der Zufallszahlen, die im Kommandofenster ausgegeben werden) langsamer erfolgen, mit einer deutlichen Verzögerung von einer Sekunde zwischen den Ausgaben. Dadurch wird die Simulation visuell besser nachvollziehbar, und Sie können die Veränderungen im Spielfeld oder in den ausgegebenen Daten leichter verfolgen.

*Bislang sind das großartige Antworten.*

## 15 Frage

In der Schleife führen Sie dann als Letztes einen Simulationsschritt aus:

```
spielfeld = schritt (app, spielfeld);
```

Was muss die als nächstes zu schreibende Methode schritt leisten?

## 15.1 Antwort

Die Methode `schritt`, die Sie als nächstes schreiben müssen, soll einen einzelnen Simulationsschritt für Ihr Spielfeld ausführen. Abhängig vom Kontext Ihrer Anwendung (z.B. Game of Life, ein partikelbasiertes Simulationsspiel oder ein anderes simulationsbasiertes Modell) kann diese Methode unterschiedliche spezifische Aufgaben erfüllen, aber im Allgemeinen muss sie folgende Funktionen leisten:

### 15.1.1 1. Zustandsaktualisierung:

- **Berechnung des neuen Zustands:** Basierend auf dem aktuellen Zustand des `spielfeld` und ggf. weiteren Parametern oder Regeln der Simulation berechnet die Methode den nächsten Zustand für jedes Element (z.B. Zelle, Partikel) des Spielfelds. Dies kann die Anwendung komplexer Regeln beinhalten, wie z.B. die Conway's Game of Life-Regeln, physikalische Gesetze in Partikelsimulationen oder benutzerdefinierte Regeln für Ihr spezifisches Simulationsszenario.

### 15.1.2 2. Regelimplementation:

- **Anwendung von Regeln:** Die Methode muss die für Ihre Simulation spezifischen Regeln implementieren. Dies kann beinhalten, den Zustand einer Zelle basierend auf der Anzahl ihrer Nachbarn zu ändern, Partikel aufgrund von Kollisionen oder Kräften zu bewegen oder andere logikbasierte Updates durchzuführen.

### 15.1.3 3. Randbedingungen:

- **Berücksichtigung von Randbedingungen:** Die Methode muss geeignet auf Randbedingungen reagieren. Dies kann periodische Randbedingungen, feste Ränder oder andere Arten von Randverhalten umfassen, die definieren, wie Elemente am Rand des Spielfelds behandelt werden.

### 15.1.4 4. Rückgabe des aktualisierten Spielfelds:

- **Rückgabe des neuen Spielfelds:** Nach der Berechnung des neuen Zustands aller Elemente des Spielfelds gibt die Methode das aktualisierte Spielfeld zurück, sodass es in der nächsten Iteration der Simulationsschleife verwendet werden kann.

### 15.1.5 Zusätzliche Überlegungen:

- **Performance:** Die Methode sollte effizient gestaltet sein, um eine flüssige Simulation zu gewährleisten, besonders wenn das `spielfeld` groß ist oder die Berechnungen komplex sind.
- **Modularität:** Idealerweise sollte die Methode so modular und wiederverwendbar wie möglich gestaltet sein, um Anpassungen und Erweiterungen der Simulation zu erleichtern.

Insgesamt ist die `schritt`-Methode zentral für die Durchführung der Simulation, da sie die Logik für die Fortführung der Simulation von einem Zustand zum nächsten definiert.

*Wow! Einfach wow!*

## 16 Frage

Erstellen Sie ein neues Unterprogramm mit den Namen `schritt`, dem Sie die Variablen `app` und `spielfeld` übergeben und das die Variable `spielfeld_neu` zurückliefert. Im Unterprogramm werden Sie im Folgenden einen Schritt des Game of Life durchführen, werden also für jede Zelle des aktuellen Spielfeldes untersuchen, ob sie geboren wird, überlebt oder stirbt und daraus dann das neue Spielfeld erzeugen. Dazu ermitteln Sie als erstes die Anzahl der Zeilen und der Spalten des aktuellen Spielfeldes und weisen sie den Variablen `n_zeilen` und `n_spalten` zu.

Wie viele Zeilen und Spalten kann das Spielfeld maximal haben?

### 16.1 Antwort

Die maximale Anzahl von Zeilen und Spalten, die das Spielfeld in MATLAB haben kann, hängt von mehreren Faktoren ab, einschließlich der Version von MATLAB, die Sie verwenden, der verfügbaren Systemressourcen (insbesondere dem verfügbaren Arbeitsspeicher) und den Einschränkungen des Betriebssystems. MATLAB selbst setzt theoretisch keine harte Grenze für die Größe von Matrizen, außer dass sie innerhalb des verfügbaren Adressraums des Systems passen müssen. In der Praxis sind Sie jedoch durch die Menge des verfügbaren Arbeitsspeichers und die maximale Arraygröße, die MATLAB verarbeiten kann, begrenzt.

Für 64-Bit-Systeme, auf denen die meisten modernen MATLAB-Versionen laufen, ist der adressierbare Speicher theoretisch sehr groß (bis zu  $2^{64}$  Bytes), was bedeutet, dass die praktische Grenze weitgehend durch den physischen und virtuellen Speicher bestimmt wird, der Ihrem MATLAB-Prozess zur Verfügung steht.

Ein typisches modernes System könnte beispielsweise 8 GB, 16 GB oder mehr RAM haben, und MATLAB könnte in der Lage sein, einen Großteil davon für eine einzelne Matrix zu nutzen, solange auch genügend Swap-Speicher (virtueller Speicher) zur Verfügung steht. Die tatsächliche Grenze für eine Matrixgröße könnte also Milliarden von Elementen betragen. Allerdings würde das Arbeiten mit Matrizen dieser Größe sehr ineffizient in Bezug auf Rechenzeit und Speicherzugriffe sein.

Für das Game of Life, bei dem jedes Zellelement in der Matrix als `uint8` (ein Byte pro Zelle) gespeichert werden kann, bedeutet dies, dass das Spielfeld theoretisch bis zu mehrere Milliarden Zellen enthalten könnte. Die praktische Obergrenze wäre jedoch viel niedriger, da auch Rechenzeit und Effizienz bei der Ausführung der Simulation berücksichtigt werden müssen.

Wenn Sie beispielsweise eine Matrix mit `spielfeld = uint8(zeros(n_zeilen, n_spalten));` initialisieren, wo `n_zeilen` und `n_spalten` die Anzahl der Zeilen und Spalten darstellen, wäre die praktische Größe des Spielfeldes so zu wählen, dass sie gut mit Ihrer Hardware und den Anforderungen Ihrer spezifischen Anwendung harmoniert.

In einer realen Anwendung des Game of Life wäre es üblich, mit Spielfeldgrößen von einigen hundert bis zu einigen tausend Zeilen und Spalten zu arbeiten, um eine angemessene Performance und Benutzererfahrung zu gewährleisten.

*Hier hat sie leider vergessen, dass wir im Feld für die Breite ein Maximum von 1000 gesetzt hatten.*

## 17 Frage

Als nächstes kopieren Sie das aktuelle Spielfeld in das neue Spielfeld mit dem Namen `spielfeld_neu`.

Etwas schwierigere Frage: Wozu ist die Kopieraktion notwendig? Warum können Sie nicht einfach im aktuellen Spielfeld die Zellen entstehen, leben und sterben lassen?

## 17.1 Antwort

Die Kopieraktion, bei der das aktuelle Spielfeld in ein neues Spielfeld (namens `spielfeld_neu`) kopiert wird, bevor die Regeln des Game of Life angewendet werden, ist aus mehreren Gründen notwendig und wichtig für die korrekte Implementierung der Simulation. Hier sind die Hauptgründe:

### 17.1.1 Unabhängigkeit der Berechnungen

- **Parallele Aktualisierung:** Im Game of Life muss der Zustand jeder Zelle basierend auf den Zuständen ihrer Nachbarn im vorherigen Schritt bestimmt werden. Um dies korrekt zu tun, müssen alle Zellen “gleichzeitig” aktualisiert werden. Würde man Änderungen direkt im aktuellen Spielfeld vornehmen, würden nachfolgende Berechnungen für andere Zellen möglicherweise auf bereits aktualisierten (und somit veränderten) Zuständen basieren, was die Regeln des Spiels verletzt.

### 17.1.2 Konsistenz der Daten

- **Erhaltung des Originalzustands:** Durch das Kopieren des aktuellen Zustands in ein neues Array und das Durchführen aller Berechnungen auf diesem neuen Array stellt man sicher, dass der Originalzustand während der gesamten Berechnung unverändert bleibt. Dies ermöglicht es, dass die Regeln für jede Zelle unter denselben Bedingungen angewendet werden, basierend auf dem Zustand des Spielfelds am Anfang des aktuellen Schritts.

### 17.1.3 Rückgängigmachung von Änderungen

- **Einfaches Zurücksetzen:** Sollte während der Berechnung der neue Zustand des Spielfelds aus irgendeinem Grund zurückgesetzt werden müssen (z.B. aufgrund eines Fehlers oder weil eine bestimmte Bedingung nicht erfüllt ist), ist dies viel einfacher zu handhaben, wenn das Originalspielfeld noch unverändert in einer separaten Variablen vorhanden ist.

### 17.1.4 Effizienz und Code-Struktur

- **Einfachere und sicherere Code-Struktur:** Das Arbeiten mit einem separaten neuen Spielfeld für die Berechnungen kann den Code einfacher und sicherer machen, da man sich keine Gedanken darüber machen muss, Zustandsänderungen innerhalb desselben Arrays zu verwalten, was zu komplexeren und fehleranfälligeren Strukturen führen könnte.

Zusammenfassend ist das Kopieren des Spielfelds in ein neues Spielfeld und das Arbeiten mit diesem neuen Spielfeld für die Berechnung des nächsten Schritts eine grundlegende Technik, um die logische Integrität der Simulation zu wahren und die Implementierung zu vereinfachen.

*Phantastische Antworten!*

## 18 Frage

Ab jetzt analysieren Sie also das aktuelle Spielfeld, ändern aber das neue Spielfeld. Starten Sie eine Doppelschleife über “fast alle” Zeilen und Spalten. Beginnen Sie dabei jede Schleife mit dem Index zwei und lassen Sie beide Schleifen nur bis zum vorletzten Element laufen. Auf diese Weise verbleibt ein fester (schwarzer) Rand um das Spielfeld, dessen Pixel niemals gesetzt werden können.

Wie oft läuft die Doppelschleife dann also insgesamt durch?

### 18.1 Antwort

Um die Anzahl der Durchläufe der Doppelschleife zu berechnen, die jeweils bei 2 beginnen und bis zum vorletzten Element jeder Dimension (Zeile und Spalte) des Spielfelds laufen, muss man die Größe des Spielfelds kennen. Angenommen, das Spielfeld hat `n_zeilen` Zeilen und `n_spalten` Spalten. Da die Schleifen bei 2 beginnen und bis zum vorletzten Element laufen, werden die äußersten Reihen und Spalten übersprungen. Die effektive Größe des Bereichs, über den die Schleifen laufen, ist also `(n_zeilen - 2)` für die Zeilen und `(n_spalten - 2)` für die Spalten.

Die Gesamtzahl der Durchläufe der Doppelschleife ist dann das Produkt dieser beiden Zahlen:

$$(n\_zeilen - 2) * (n\_spalten - 2)$$

Das bedeutet, wenn das Spielfeld zum Beispiel 10 Zeilen und 10 Spalten hätte, dann wäre die Berechnung:

$$(10 - 2) * (10 - 2) = 8 * 8 = 64$$

In diesem Fall würde die Doppelschleife also insgesamt 64 Mal durchlaufen. Jeder Durchlauf entspricht einer Zelle im Inneren des Spielfelds, die analysiert und potenziell aktualisiert wird, ohne den festen Rand zu berühren.

*Genau so.*

## 19 Frage

In der Doppelschleife bestimmen Sie für jedes Pixel, wie viele lebende Nachbarn er besitzt, indem Sie einfach die Spielfeldwerte seiner Nachbarn aufaddieren.

Welche Werte müssen lebende bzw. tote Nachbarn haben, damit diese Methode funktioniert?

### 19.1 Antwort

Damit die Methode des einfachen Aufaddierens der Spielfeldwerte zur Bestimmung der Anzahl lebender Nachbarn funktioniert, müssen lebende und tote Nachbarn auf dem Spielfeld durch Werte repräsentiert werden, die eine solche Addition sinnvoll machen. Im Kontext des Game of Life und ähnlicher zellulärer Automaten ist es üblich, folgende Konventionen zu verwenden:

- **Lebende Zellen** werden durch den Wert **1** repräsentiert.
- **Tote Zellen** werden durch den Wert **0** repräsentiert.

Diese Konvention ermöglicht es, die Anzahl der lebenden Nachbarn einer Zelle direkt zu berechnen, indem man einfach die Werte der acht umgebenden Zellen aufaddiert (für jede Zelle, die in einer 2D-Gitterumgebung betrachtet wird). Da tote Zellen den Wert 0 haben, tragen sie nicht zur Summe

bei. Lebende Zellen hingegen, die durch den Wert 1 dargestellt werden, erhöhen die Summe um 1 für jeden lebenden Nachbarn.

Diese einfache und direkte Methode ermöglicht eine effiziente Berechnung der für die Spielregeln des Game of Life notwendigen Anzahl lebender Nachbarn, ohne dass komplizierte Abfragen oder bedingte Anweisungen erforderlich sind.

*Extrem hilfreich!*

## 20 Frage

Als letztes fragen Sie jetzt ab, ob ein Pixel im aktuellen Spielfeld leer ist und drei Nachbarn hat, um in diesem Fall das entsprechende Pixel im neuen Spielfeld zu setzen. Des Weiteren löschen Sie ein Pixel, wenn es weniger als zwei oder mehr als drei Nachbarn hat.

Warum müssen Sie den Fall, dass ein gesetztes Pixel genau zwei oder drei gesetzte Nachbarn hat, nicht gesondert betrachten?

Warum können Sie die Variable `app` in der Parameterliste des Unterprogrammes schritt durch eine Tilde (`~`) ersetzen?

### 20.1 Antwort

#### 20.1.1 Behandlung bestimmter Fälle nach den Regeln des Game of Life

Im Game of Life gelten die folgenden grundlegenden Regeln für die Evolution einer Zelle (Pixel) von einem Schritt zum nächsten:

1. **Geburt:** Eine tote Zelle (Pixel) mit genau drei lebenden Nachbarn wird in der nächsten Generation lebendig.
2. **Tod durch Einsamkeit:** Eine lebende Zelle mit weniger als zwei lebenden Nachbarn stirbt in der nächsten Generation.
3. **Überleben:** Eine lebende Zelle mit zwei oder drei lebenden Nachbarn bleibt in der nächsten Generation lebendig.
4. **Tod durch Überbevölkerung:** Eine lebende Zelle mit mehr als drei lebenden Nachbarn stirbt in der nächsten Generation.

Die Notwendigkeit, bestimmte Fälle gesondert zu betrachten, ergibt sich aus diesen Regeln:

- Der Fall, dass ein **gesetztes Pixel genau zwei oder drei gesetzte Nachbarn** hat, muss nicht gesondert betrachtet werden, weil diese Zellen laut Regel 3 einfach **weiterleben**. Wenn Sie das aktuelle Spielfeld in das neue Spielfeld kopieren, bevor Sie Änderungen vornehmen, bleiben diese Zellen bereits im gewünschten Zustand (lebendig), sofern sie nicht explizit geändert werden. Das heißt, es ist keine Aktion erforderlich, um diese Zellen im neuen Spielfeld zu “setzen”, da sie durch das Kopieren des Spielfeldes bereits korrekt gesetzt sind.
- Der Fall, dass eine Zelle “geboren” wird oder “stirbt”, sind die Szenarien, die explizit behandelt werden müssen, da sie eine Zustandsänderung erfordern: von tot zu lebendig (für die Geburt) oder von lebendig zu tot (für den Tod).

### 20.1.2 Ersetzen von `app` durch `~` in der Parameterliste

In MATLAB ermöglicht das Ersetzen eines Funktionsparameters durch eine Tilde (`~`) in der Parameterliste einer Funktion, diesen Parameter zu ignorieren. Dies ist nützlich, wenn die Funktion einen Parameter erwartet, der in der spezifischen Implementierung der Funktion jedoch nicht benötigt wird. Indem Sie `app` durch `~` ersetzen, signalisieren Sie, dass Ihr Unterprogramm `schritt` zwar einen Platzhalter für diesen Parameter hat (möglicherweise, um konsistent mit einer erwarteten Schnittstelle zu sein), diesen aber nicht verwendet. Dies kann die Klarheit verbessern und vermeiden, dass ungenutzte Variablen Speicherplatz im Funktionskontext belegen oder Verwirrung darüber stiften, wie die Funktion arbeitet.

*Einfach nur gut!*

## 21 Frage

Im Folgenden möchten Sie das Callback-Unterprogramm für die Datei-Schaltfläche schreiben, in der das externe Unterprogramm `conway_ini.mlx` zum Lesen einer Bilddatei aufgerufen und das Bild als Anfangskonfiguration dargestellt wird. Dazu initialisieren Sie im Callback-Unterprogramm das gesamte Spielfeld (die Größe kommt natürlich wieder aus dem Eingabefeld) mit `UINT8`-Nullen, rufen das externe Unterprogramm `conway_ini` auf und weisen dessen Ergebnis der Variable `bild` zu. Bestimmen Sie die Anzahl der Zeilen und Spalten von `bild` und kopieren Sie das Bild in die linke obere Ecke des Spielfeldes.

Wie könnten Sie erreichen, dass ein Bild, das kleiner als das Spielfeld ist, trotzdem in der Mitte des Spielfeldes landet?

### 21.1 Antwort

Um ein Bild, das kleiner als das Spielfeld ist, in der Mitte des Spielfeldes zu platzieren, müssen Sie zunächst die Größe des Bildes (`bild`) und die Größe des Spielfeldes (`spielfeld`) bestimmen. Anschließend berechnen Sie die Startindizes für Zeilen und Spalten im Spielfeld, sodass das Bild zentriert ist. Hier ist ein allgemeiner Ansatz, wie Sie dies in MATLAB erreichen können:

#### 21.1.1 Schritt 1: Bestimmen der Größen

Angenommen, `bild` und `spielfeld` sind bereits initialisiert, und `bild` wurde dem Ergebnis des Aufrufs von `conway_ini` zugewiesen. Die Anzahl der Zeilen und Spalten von `bild` kann mit der `size`-Funktion ermittelt werden:

```
[n_zeilen_bild, n_spalten_bild] = size(bild);
```

Und die Größen des Spielfeldes sind durch die Werte im Eingabefeld oder eine andere Methode der Initialisierung bekannt:

```
[n_zeilen_spielfeld, n_spalten_spielfeld] = deal(breite, breite); % Wenn das Spielfeld quadra
```

#### 21.1.2 Schritt 2: Berechnung der Startindizes

Um das Bild zentriert zu platzieren, müssen Sie die Startindizes für das Einfügen des Bildes in das Spielfeld berechnen. Dies kann durch Subtraktion der Bildgröße von der Spielfeldgröße und Teilung durch 2 erreicht werden (unter Verwendung von `floor` oder `ceil` für ganzzahlige Indizes):

```
start_zeile = floor((n_zeilen_spiel_feld - n_zeilen_bild) / 2) + 1;
start_spalte = floor((n_spalten_spiel_feld - n_spalten_bild) / 2) + 1;
```

### 21.1.3 Schritt 3: Platzierung des Bildes

Jetzt können Sie das Bild in das Spielfeld einfügen, indem Sie die berechneten Startindizes verwenden:

```
end_zeile = start_zeile + n_zeilen_bild - 1;
end_spalte = start_spalte + n_spalten_bild - 1;

spiel_feld_neu(start_zeile:end_zeile, start_spalte:end_spalte) = bild;
```

Dieser Codeausschnitt platziert das Bild `bild` zentriert im `spiel_feld_neu`, indem er es an den berechneten Startpositionen einfügt und dabei sicherstellt, dass das Bild genau in der Mitte des Spielfeldes positioniert wird, sofern die Größen von Bild und Spielfeld dies zulassen.

Es ist wichtig zu beachten, dass Sie sicherstellen müssen, dass das Bild vollständig innerhalb der Grenzen des Spielfeldes liegt. Wenn die berechneten Start- oder Endindizes außerhalb der Grenzen des Spielfeldes liegen würden, müssen Sie geeignete Anpassungen vornehmen, um Fehler zu vermeiden.

*Ob der `deal`-Befehl hier wirklich sinnvoll ist ...?*

## 22 Frage

Abschließend setzen Sie wieder - wie im Zufall-Callback-Unterprogramm - `app.game_over` auf `false` und starten die Simulationsschleife.

Warum muss die `game_over`-Variable eigentlich jedes Mal wieder neu initialisiert werden?

### 22.1 Antwort

Die `game_over`-Variable wird jedes Mal neu initialisiert (auf `false` gesetzt), bevor die Simulationsschleife gestartet wird, um sicherzustellen, dass die Simulation unter den richtigen Bedingungen beginnt. Hier sind die Hauptgründe für diese Praxis:

#### 22.1.1 Rücksetzen des Simulationszustands

- Die Simulation oder das Spiel kann mehrmals gestartet und gestoppt werden, während die App läuft. Das Zurücksetzen von `game_over` auf `false` stellt sicher, dass vorherige Beendigungen der Simulation keinen Einfluss auf den neuen Start haben. Ohne dieses Zurücksetzen würde die Schleife sofort enden oder gar nicht erst starten, wenn `game_over` noch von einem vorherigen Lauf auf `true` gesetzt wäre.

#### 22.1.2 Kontrolle über die Programmausführung

- Das Setzen von `game_over` auf `false` gibt explizit an, dass die Bedingungen für das Beenden der Simulation nicht erfüllt sind, und die Simulationsschleife kann wie vorgesehen beginnen. Dies ist wichtig für die Kontrolle der Programmausführung und um sicherzustellen, dass die Simulationsschleife unter kontrollierten und vorhersehbaren Bedingungen startet.

### 22.1.3 Benutzerinteraktion ermöglichen

- In vielen Fällen wird die `game_over`-Variable durch Benutzerinteraktion beeinflusst, beispielsweise durch das Klicken auf eine Start- oder Stopp-Schaltfläche. Das erneute Initialisieren ermöglicht es dem Benutzer, die Simulation nach einem Stopp neu zu starten, ohne dass die App neu geladen oder anderweitig zurückgesetzt werden muss.

### 22.1.4 Klarheit und Wartbarkeit des Codes

- Das explizite Setzen von Steuervariablen wie `game_over` auf ihren Ausgangswert vor Beginn eines Prozesses (wie der Simulationsschleife) trägt zur Klarheit des Codes bei. Entwickler und andere Personen, die den Code später lesen oder warten, können leicht verstehen, dass die Ausführung der Schleife von diesem Steuermechanismus abhängt.

Zusammengefasst dient das Zurücksetzen der `game_over`-Variable dazu, die korrekte Ausführung der Simulationsschleife zu gewährleisten und bietet eine klare Trennung zwischen aufeinanderfolgenden Ausführungen der Simulation. Es sorgt für eine robuste und benutzerfreundliche Anwendung, die mehrfach gestartet und gestoppt werden kann, ohne dass unerwartetes Verhalten auftritt.

*Yupp.*

## 23 Frage

Die App ist jetzt fertig und im Folgenden müssen Sie (nur) noch das Unterprogramm `conway_ini.mlx` schreiben, das aufgerufen wird, wenn Sie die Datei-Schaltfläche anklicken. Im Unterprogramm lesen Sie mit dem Befehl `imread` die Bilddatei

`conway_glider_gun.png`

ein, die Sie natürlich vom Server herunterladen und weisen die entsprechende Matrix der Variablen `bild` zu.

Ist `bild` ein Farb-, Graustufen- oder Schwarz-Weiß-Bild? Woran erkennen Sie dies?

### 23.1 Antwort

Wenn Sie ein Bild mit dem Befehl `imread` in MATLAB einlesen, hängt der Typ der resultierenden Matrix (`bild`) von den Farbinformationen des Bildes ab. Die Matrix kann entweder 2D (für Graustufen- oder Schwarz-Weiß-Bilder) oder 3D (für Farbbilder) sein. Die Dimension der Matrix gibt Ihnen einen Hinweis auf den Bildtyp:

- **Farbbild:** Ein Farbbild resultiert in einer 3D-Matrix. Die ersten beiden Dimensionen repräsentieren die Höhe und Breite des Bildes, während die dritte Dimension die Farbkanäle (üblicherweise Rot, Grün und Blau) repräsentiert. Eine 3D-Matrix würde somit bedeuten, dass `bild` ein Farbbild ist. Zum Beispiel hätte ein 100x100 Pixel großes RGB-Farbbild die Dimensionen 100x100x3.
- **Graustufenbild:** Ein Graustufenbild resultiert in einer 2D-Matrix, bei der jeder Eintrag in der Matrix einen Grauwert (zwischen 0 und 255 für ein 8-Bit-Bild) für einen Pixel darstellt. Das Fehlen einer dritten Dimension für Farbkanäle zeigt an, dass es sich um ein Graustufenbild handelt.

- **Schwarz-Weiß-Bild (Binärbild):** Auch Schwarz-Weiß-Bilder werden oft als 2D-Matrizen gespeichert, ähnlich wie Graustufenbilder. Der Unterschied liegt in den Pixelwerten: Ein Schwarz-Weiß-Bild hat typischerweise nur zwei Werte, 0 (für Schwarz) und 1 (für Weiß) oder alternativ 0 und 255, abhängig davon, wie die Binärdaten gespeichert sind. Auch wenn es technisch eine 2D-Matrix wie ein Graustufenbild ist, kann die Tatsache, dass nur zwei Werte vorkommen, darauf hinweisen, dass es sich um ein Schwarz-Weiß-Bild handelt.

Um genau zu bestimmen, ob `bild` ein Farb-, Graustufen- oder Schwarz-Weiß-Bild ist, könnten Sie die Dimension der Matrix prüfen und die Werteverteilung innerhalb der Matrix analysieren:

```
% Prüfen der Dimensionen
dims = ndims(bild); % Gibt 2 für Graustufen/Schwarz-Weiß, 3 für Farb zurück

% Für 2D-Matrizen könnte man zusätzlich die Werteverteilung prüfen
if dims == 2
    uniqueValues = unique(bild);
    if length(uniqueValues) == 2
        % Wahrscheinlich ein Schwarz-Weiß-Bild
    else
        % Wahrscheinlich ein Graustufenbild
    end
end
```

Für `conway_glider_gun.png`, ohne das Bild direkt gesehen zu haben, basiert die Annahme auf dem Kontext (Conway's Game of Life), dass es wahrscheinlich ein Schwarz-Weiß- oder Graustufenbild ist, da diese für die Darstellung solcher Simulationen üblich sind. Die exakte Bestimmung erfordert jedoch eine Untersuchung der Matrix, wie oben beschrieben.

*Großartig.*

## 24 Frage

Wandeln Sie das Bild erst in Graustufen um, verkleinern Sie es auf  $11 \times 38$  Pixel und wandeln es dann in ein Schwarz-Weiß-Bild um.

Was ist aus den weißen Zellen nach der Verkleinerung geworden? Warum ist das so?

### 24.1 Antwort

Um ein Bild in Graustufen umzuwandeln, es dann auf eine Größe von  $11 \times 38$  Pixel zu verkleinern und schließlich in ein Schwarz-Weiß-Bild umzuwandeln, durchläuft man mehrere Bildverarbeitungsschritte. Jeder dieser Schritte hat spezifische Auswirkungen auf das Bild, insbesondere auf die Darstellung der ursprünglich weißen Zellen. Hier ist, was im Allgemeinen passiert:

#### 24.1.1 1. Umwandlung in Graustufen

Die Umwandlung eines Farb- oder Graustufenbildes in Graustufen (falls das Bild nicht bereits in Graustufen vorliegt) führt dazu, dass die Farbinformationen entfernt werden. Die Helligkeitswerte der ursprünglichen Farben werden in entsprechende Graustufen umgerechnet. Weiße Bereiche bleiben in der Regel weiß, da Weiß alle Farben bei maximaler Helligkeit kombiniert.

### 24.1.2 2. Verkleinerung des Bildes

Die Verkleinerung des Bildes auf  $11 \times 38$  Pixel führt zu einem Informationsverlust, insbesondere wenn das ursprüngliche Bild erheblich größer war. Dieser Schritt wird üblicherweise durch eine Operation wie das Downsampling oder durch Anwendung von Algorithmen zur Bildinterpolation durchgeführt. Dabei werden mehrere benachbarte Pixel zusammengefasst, um den Wert eines neuen, kleineren Pixels zu bilden.

- **Auswirkungen auf weiße Zellen:** Wenn weiße Zellen (Pixel) in der Nähe von oder neben dunkleren Zellen liegen, kann ihre Helligkeit in den Prozess der Mittelwertbildung oder Interpolation einfließen, was dazu führen kann, dass die resultierenden Pixel in der verkleinerten Version des Bildes nicht mehr rein weiß sind. Stattdessen können sie verschiedene Grautöne annehmen, abhängig von der Helligkeit der umliegenden Zellen im ursprünglichen Bild.

### 24.1.3 3. Umwandlung in ein Schwarz-Weiß-Bild

Die endgültige Umwandlung in ein Schwarz-Weiß-Bild erfordert eine Entscheidungsschwelle (Threshold), bei der Graustufenpixel in entweder schwarz oder weiß umgewandelt werden, basierend darauf, ob sie heller oder dunkler als die Schwelle sind.

- **Verbleib der weißen Zellen:** Die zuvor weißen Zellen, die durch die Verkleinerung und die daraus resultierende Mittelwertbildung oder Interpolation in verschiedene Grautöne umgewandelt wurden, können je nach gewählter Schwelle entweder zu schwarz oder zu weiß werden. Wenn die Schwelle so gewählt wird, dass sie nahe am helleren Ende des Spektrums liegt, könnten viele dieser ursprünglich weißen, jetzt aber grauen Zellen als schwarze Zellen im Schwarz-Weiß-Bild erscheinen. Dies liegt daran, dass ihre Helligkeit durch die Vermischung mit dunkleren Pixeln während der Verkleinerung reduziert wurde.

Zusammenfassend kann die Verkleinerung eines Bildes dazu führen, dass ursprünglich weiße Zellen aufgrund der Interpolation mit benachbarten dunkleren Zellen nicht mehr rein weiß sind. Ihre Darstellung im endgültigen Schwarz-Weiß-Bild hängt dann von der gewählten Schwelle für die Umwandlung ab.

*Ja, aber ein bisschen lang.*

## 25 Frage

Verwenden Sie den Befehl

```
bild = ~bild;
```

Was macht der Befehl? Warum ist dies notwendig?

### 25.1 Antwort

Der Befehl `bild = ~bild;` in MATLAB führt eine logische Negation (NOT-Operation) auf jedes Element der Matrix `bild` aus. Dies bedeutet, dass der Befehl die Werte der Matrix invertiert:

- Werte, die vor der Operation 0 (was als `false` interpretiert wird) waren, werden zu 1 (`true`).
- Werte, die 1 (oder irgendein anderer von null verschiedener Wert, der als `true` interpretiert wird) waren, werden zu 0 (`false`).

Diese Operation wird oft bei binären oder logischen Matrizen angewendet, wo 1 und 0 spezifische Zustände oder Bedingungen repräsentieren, z.B. bei Schwarz-Weiß-Bildern, wo 1 möglicherweise weiße Pixel und 0 schwarze Pixel darstellt (oder umgekehrt, abhängig von der Konvention).

### 25.1.1 Warum ist dies notwendig?

Die Notwendigkeit, die Werte mit `~bild` zu invertieren, kann aus mehreren Gründen entstehen, insbesondere in der Bildverarbeitung oder in logischen Operationen:

1. **Anpassung an Erwartungen oder Konventionen:** Wenn Ihre Verarbeitungslogik oder die nachfolgenden Operationen erwarten, dass bestimmte Bedingungen oder Objekte im Bild als `true` (oder 1) repräsentiert werden und der aktuelle Zustand des Bildes diesen Erwartungen entgegengesetzt ist, hilft die Invertierung dabei, das Bild entsprechend anzupassen.
2. **Visuelle Darstellung:** In manchen Fällen möchten Sie vielleicht, dass weiße Pixel schwarz dargestellt werden und umgekehrt, um einen besseren Kontrast oder eine klarere Visualisierung bestimmter Aspekte des Bildes zu erreichen.
3. **Logische Operationen und Analyse:** Bei der Durchführung bestimmter logischer oder analytischer Operationen kann es nützlich sein, die Zustände zu invertieren, um die Identifizierung oder Zählung von Objekten, Mustern oder Bereichen zu erleichtern.

Ein Beispiel aus dem Kontext des Game of Life könnte sein, dass Sie die Konvention verwenden, bei der 1 tote Zellen und 0 lebende Zellen repräsentiert. Wenn Ihre Simulationslogik jedoch auf einer Konvention basiert, bei der 1 eine lebende Zelle bedeutet, könnte die Invertierung der Werte vor der Verarbeitung notwendig sein, um die Logik korrekt anzuwenden.

*Genau das!*

## 26 Frage

Um zwei Gleiterkanonen gegeneinander “kämpfen” zu lassen, spiegeln Sie das Bild:

```
bild_gespiegelt = fliplr(bild);
```

An welcher Achse haben Sie das Bild gerade gespiegelt?

### 26.1 Antwort

Der Befehl `fliplr(bild)` in MATLAB spiegelt die Matrix `bild` an der vertikalen Achse. Das bedeutet, dass die Reihenfolge der Spalten von links nach rechts umgekehrt wird. Wenn Sie also `bild_gespiegelt = fliplr(bild);` verwenden, erzeugen Sie eine gespiegelte Version des Originalbildes, bei der das, was vorher auf der linken Seite war, jetzt auf der rechten Seite ist, und umgekehrt. Es handelt sich um eine Spiegelung entlang der vertikalen Mitte des Bildes.

*Richtig.*

## 27 Frage

Abschließend fügen Sie dann die gespiegelte Gleiterkanone in das Gesamtbild ein:

```
bild(1 : 11, 63 : 100) = bild_gespiegelt;
```

und klicken die Datei-Schaltfläche der App an.

Beschreiben Sie, was passiert.

Was passiert, wenn Sie die gespiegelte Gleiterkanone ein Pixel weiter nach links schieben?

Was passiert, wenn Sie den Spaltenbereich der gespiegelten Gleiterkanone auf 461 : 498 setzen (und dabei natürlich die Breite der Darstellung auf 500)?

Gibt es dann einen stationären Endzustand?

## 27.1 Antwort

Die Beschreibung basiert auf der Annahme, dass Sie bereits eine Gleiterkanone in Ihrem Spielfeld haben, und nun eine gespiegelte Version dieser Gleiterkanone hinzufügen möchten, um zu sehen, wie sie interagieren.

### 27.1.1 Was passiert, nachdem Sie die gespiegelte Gleiterkanone hinzugefügt haben?

Wenn Sie `bild(1 : 11, 63 : 100) = bild_gespiegelt;` ausführen, wird die gespiegelte Gleiterkanone in den angegebenen Bereich des Spielfelds eingefügt. Dies positioniert die zweite Gleiterkanone so, dass sie ihre "Gleiter" (oder andere Strukturen, die sie erzeugt) in Richtung der ersten, ursprünglichen Gleiterkanone aussendet. Da beide Kanonen nun Gleiter oder andere Strukturen in Richtung der jeweils anderen aussenden, wird dies zu einer Interaktion zwischen den von den Kanonen erzeugten Strukturen führen, sobald die App gestartet wird.

### 27.1.2 Was passiert, wenn Sie die gespiegelte Gleiterkanone ein Pixel weiter nach links schieben?

Wenn Sie die gespiegelte Gleiterkanone ein Pixel weiter nach links verschieben, indem Sie den Bereich, in den sie eingefügt wird, leicht anpassen (z.B. `bild(1 : 11, 62 : 99) = bild_gespiegelt;`), ändern Sie die Ausgangspositionen der beiden sich entgegenkommenden Gleiterströme leicht. Dies kann zu einer Änderung der Art und Weise führen, wie die Gleiter und eventuelle andere Strukturen kollidieren oder interagieren. Kleinste Änderungen in den Startpositionen können im Game of Life zu dramatisch unterschiedlichen Ergebnissen führen, aufgrund des nicht-linearen und chaotischen Charakters der Dynamik des Spiels.

### 27.1.3 Was passiert, wenn Sie den Spaltenbereich der gespiegelten Gleiterkanone auf 461 : 498 setzen?

Indem Sie die gespiegelte Gleiterkanone an die Position 461 : 498 setzen (und die Breite des Spielfelds entsprechend auf mindestens 500 einstellen), platzieren Sie die beiden Kanonen weit voneinander entfernt am Spielfeld. Dies gibt den von den Kanonen erzeugten Strukturen mehr Raum, sich zu entwickeln, bevor sie möglicherweise aufeinandertreffen. Wenn die Gleiter oder andere Strukturen, die von den Kanonen ausgesendet werden, lange genug brauchen, um die andere Seite zu erreichen, oder wenn sie in einem hinreichend großen Spielfeld überhaupt nicht aufeinandertreffen, könnte es sein, dass die Kanonen ungestört weiter Gleiter aussenden.

### 27.1.4 Gibt es dann einen stationären Endzustand?

Ob es einen stationären Endzustand gibt, hängt von der spezifischen Interaktion der ausgesendeten Strukturen ab. In vielen Fällen führen Kollisionen zwischen den Strukturen zu komplexen

Dynamiken, die in stabilen Strukturen (stationären Zuständen), oszillierenden Mustern oder sogar in Chaos resultieren können. Bei weit voneinander entfernten Gleiterkanonen, die in gegenüberliegende Richtungen schießen, könnte es sein, dass keine direkte Interaktion stattfindet, solange die Ströme nicht auf Hindernisse oder die Grenzen des Spielfelds treffen. In einem großen genug Spielfeld könnten die Kanonen theoretisch unendlich lange Gleiter aussenden, ohne jemals einen stationären Zustand zu erreichen. Die spezifische Evolution und das mögliche Erreichen eines stationären Zustands hängen stark von den initialen Bedingungen und der Konfiguration des Spielfelds ab.

*Sehr salomonisch, aber durchaus wahr und hilfreich!*

## 28 Frage

Spielen Sie, experimentieren Sie.

Beschreiben Sie interessante Phänomene.

### 28.1 Antwort

Da ich nicht direkt mit der App interagieren oder die Ergebnisse von Experimenten in Echtzeit sehen kann, kann ich einige der faszinierenden Phänomene, die typischerweise im Game of Life und ähnlichen zellulären Automaten beobachtet werden, beschreiben. Das Game of Life ist bekannt für seine Fähigkeit, trotz einfacher Regeln eine erstaunliche Vielfalt an Verhaltensweisen zu erzeugen, einschließlich, aber nicht beschränkt auf:

#### 28.1.1 Stabile Strukturen

- **Blöcke, Bienenkörbe, Boote und Loafs:** Diese sind einfache Strukturen, die von Generation zu Generation unverändert bleiben. Ihre Stabilität macht sie zu den grundlegenden Bausteinen für komplexere Konstruktionen im Spiel.

#### 28.1.2 Oszillatoren

- **Blinker und Toads:** Diese Strukturen ändern ihre Form zyklisch und kehren nach einer bestimmten Anzahl von Generationen zu ihrem Ausgangszustand zurück. Oszillatoren können eine einfache periodische Bewegung wie der Blinker (Periode 2) zeigen oder komplexere Bewegungen durchlaufen.
- **Pulsars und Pentadecathlon:** Komplexere Oszillatoren, die über größere Bereiche agieren und längere Perioden haben. Sie zeigen, wie rhythmische Veränderungen über mehrere Zellen hinweg koordiniert werden können.

#### 28.1.3 Gleiter und Raumschiffe

- **Gleiter:** Kleine Konfigurationen, die sich diagonal über das Spielfeld bewegen. Gleiter sind bemerkenswert, weil sie die einfachste Form von sich selbst replizierenden Strukturen im Game of Life sind.
- **Leichte und schwere Raumschiffe (LWSS und HWSS):** Diese größeren Strukturen bewegen sich horizontal oder vertikal über das Spielfeld. Ihre Fähigkeit, sich fortzubewegen,

demonstriert, wie komplexe Verhaltensweisen aus den einfachen Regeln des Spiels entstehen können.

#### 28.1.4 Gleiterkanonen

- **Gosper's Gleiterkanone:** Eine Struktur, die periodisch Gleiter erzeugt und aussendet. Diese Konstruktion illustriert, wie das Spiel unendliches Wachstum und Komplexität aus einer statischen Anfangskonfiguration generieren kann.

#### 28.1.5 Komplexe Interaktionen und Reaktionen

- **Kollisionen zwischen Gleitern und anderen Strukturen:** Interessante Phänomene entstehen, wenn sich bewegende Strukturen kollidieren. Solche Kollisionen können zu völlig neuen Strukturen, zur Zerstörung oder Transformation der beteiligten Elemente führen.
- **Logische Tore und Rechenoperationen:** Durch geschickte Anordnung von Gleitern, Gleiterkanonen und anderen Strukturen können Spieler logische Schaltkreise und sogar einfache Computer innerhalb des Spiels konstruieren. Dies zeigt die Berechnungsfähigkeit des Game of Life und wie es als universeller Turing-Automat fungieren kann.

Diese Phänomene unterstreichen die unerwartete Komplexität und Schönheit, die aus den vier einfachen Regeln des Game of Life entstehen können. Sie zeigen, wie einfache lokale Interaktionen zu hochorganisiertem globalem Verhalten führen können, ein Konzept, das auch in vielen Bereichen der Wissenschaft und Mathematik von Bedeutung ist.

*Wunderschöne Antworten!*

# 11 3D-Grafik

April 11, 2024

## 1 Frage

Sie möchten im Folgenden ein sehr einfaches dreidimensionales Modell eines Flugzeugs erstellen und dieses dann im Raum bewegen und drehen. Erstellen Sie dazu ein Achsensystem mit den festen Grenzen  $-5 \dots 5$  in allen Achsen (händischen `DataAspectRatioMode` nicht vergessen!). Zusätzlich kehren Sie mit den Parametern `ZDir` und `YDir` die Richtung der entsprechenden beiden Achsen um.

Warum ist die Richtungsumkehr notwendig, wenn Sie das Koordinatensystem normgerecht entsprechend <https://m-server.fk5.hs-bremen.de/kos/kos.html> darstellen wollen?

### 1.1 Antwort

Um ein einfaches dreidimensionales Modell eines Flugzeugs in MATLAB zu erstellen und im Raum zu bewegen und zu drehen, einschließlich der Erstellung eines Achsensystems mit den festen Grenzen  $-5$  bis  $5$  für alle Achsen und der Umkehrung der Richtung der Z- und Y-Achsen, folgt hier ein MATLAB-Code-Beispiel:

```
% Achsensystem erstellen
figure;
axes('XLim', [-5 5], 'YLim', [-5 5], 'ZLim', [-5 5], ...
     'DataAspectRatio', [1 1 1], ...
     'DataAspectRatioMode', 'manual', ...
     'ZDir', 'reverse', 'YDir', 'reverse');
grid on;
view(3); % Dreidimensionale Ansicht

% Hier könnten weitere Befehle folgen, um das Flugzeugmodell zu erstellen
% und zu animieren, zum Beispiel mit patch- oder plot3-Befehlen.
```

Die Umkehrung der Richtung der Z- und Y-Achsen (`'ZDir', 'reverse', 'YDir', 'reverse'`) ist notwendig, um das Koordinatensystem normgerecht entsprechend den Konventionen darzustellen, die auf der verlinkten Seite zu finden sind. In vielen technischen und wissenschaftlichen Kontexten ist es üblich, dass die Z-Achse nach oben zeigt und die Y-Achse in die Tiefe geht, was von der Standardkonfiguration in MATLAB abweicht, wo die Z-Achse in die Tiefe zeigt und die Y-Achse nach oben. Die Umkehrung stellt sicher, dass das Koordinatensystem den gewohnten Konventionen entspricht, was vor allem in der Luft- und Raumfahrt wichtig ist, um Missverständnisse bei der Interpretation der Orientierung und Bewegung von Objekten zu vermeiden.

*Die Aussage "Standardkonfiguration in MATLAB abweicht, wo die Z-Achse in die Tiefe zeigt und*

die Y-Achse nach oben" ist falsch.

## 2 Frage

Benennen Sie die Achsen mit x-Achse, y-Achse und z-Achse und schalten Sie das Raster und eine perspektivische Projektion ein. Wählen Sie eine sinnvolle Blickrichtung mit

```
view(-128, 30)
```

und schalten Sie mit dem Befehl `rotate3d` die Möglichkeit ein, auch während der Simulation das Koordinatensystem mit der Maus zu drehen.

Was bedeuten die beiden Parameter des `view`-Befehls?

Warum klappt der übliche Befehl `view(3)` nicht?

### 2.1 Antwort

Um die Achsen zu benennen, das Raster und eine perspektivische Projektion einzuschalten, sowie eine sinnvolle Blickrichtung auszuwählen und die Möglichkeit zu aktivieren, das Koordinatensystem während der Simulation mit der Maus zu drehen, können Sie folgenden MATLAB-Code verwenden:

```
% Achsensystem erstellen
figure;
ax = axes('XLim', [-5 5], 'YLim', [-5 5], 'ZLim', [-5 5], ...
         'DataAspectRatio', [1 1 1], ...
         'DataAspectRatioMode', 'manual', ...
         'ZDir', 'reverse', 'YDir', 'reverse');
grid on; % Raster einschalten
view(-128, 30); % Blickrichtung einstellen
camproj('perspective'); % Perspektivische Projektion
rotate3d on; % Rotation mit der Maus ermöglichen

% Achsen benennen
xlabel('x-Achse');
ylabel('y-Achse');
zlabel('z-Achse');
```

Die beiden Parameter des `view`-Befehls bedeuten:

1. **Azimutwinkel (-128 Grad):** Der Azimutwinkel beschreibt die Rotation um die z-Achse. Ein negativer Wert bedeutet eine Drehung im Uhrzeigersinn, wenn man von der positiven Z-Achse aus nach unten schaut. In diesem Fall dreht -128 Grad die Ansicht so, dass sie leicht von der Seite betrachtet wird.
2. **Elevationswinkel (30 Grad):** Der Elevationswinkel bestimmt den Winkel über der x-y-Ebene. Ein positiver Wert hebt die Blickrichtung an, sodass man von einem erhöhten Standpunkt aus auf das Koordinatensystem schaut. 30 Grad sorgen für eine leichte Anhebung, was zu einer intuitiven dreidimensionalen Ansicht führt.

Der Befehl `view(3)` funktioniert in diesem Kontext nicht, da `view(3)` eine vordefinierte Ansicht ist, die eine isometrische Ansicht des Koordinatensystems darstellt, bei der die Achsen gleichmäßig

skaliert sind, um die dreidimensionale Struktur ohne Perspektivverzerrung darzustellen. In Szenarien, wo eine spezifische Ausrichtung oder eine perspektivische Ansicht gewünscht ist, muss `view` mit spezifischen Azimut- und Elevationswinkeln aufgerufen werden, um die gewünschte Ansicht zu erzielen. Die isometrische Ansicht (`view(3)`) bietet keinen spezifischen Blickwinkel oder eine perspektivische Verzerrung, die für bestimmte Visualisierungen oder zur Betonung der räumlichen Beziehungen innerhalb des Koordinatensystems nötig sein könnten.

*Bisschen holperige Erklärungen, aber passt.*

### 3 Frage

Im Folgenden erstellen Sie das 3D-Modell des Flugzeugs. Es soll ähnlich wie in

<https://m-server.fk5.hs-bremen.de/kos/kos.html>

aus zwei Dreiecken bestehen, deren Eckenmatrix folgendermaßen definiert ist:

```
ecken = [ ... -1 -1 0; ... -1 1 0; ... 2 0 0; ... -1 0 0; ... -1 0 -1; ... 0 0 0];
```

Welche Ecken gehören dabei zum “Rumpf”, welche zum Seitenleitwerk?

Welche Ecke definiert die vordere Spitze des Flugzeugs?

Welche Ecke definiert die rechte Flügelspitze?

Welche Ecke definiert die obere Spitze des Leitwerks?

Welche Ecke welches Flugzeugteils liegt genau im Ursprung?

#### 3.1 Antwort

Betrachten wir die gegebene Eckenmatrix für das 3D-Modell des Flugzeugs:

```
ecken = [ ...  
-1 -1 0; ... % Ecke 1  
-1 1 0; ... % Ecke 2  
2 0 0; ... % Ecke 3  
-1 0 0; ... % Ecke 4  
-1 0 -1; ... % Ecke 5  
0 0 0]; % Ecke 6
```

Die Ecken sind im Format `[x y z]` definiert, wobei `x`, `y` und `z` die Koordinaten im Raum darstellen.

- **Zum “Rumpf” gehören:** Die Ecken, die das Hauptdreieck des Flugzeugmodells bilden und den größten Teil des Flugzeugs ausmachen. Das wären Ecke 1, Ecke 2 und Ecke 3. Diese drei Ecken bilden ein Dreieck, das die Hauptform des Flugzeugs definiert, mit zwei Ecken hinten (links und rechts) und der vorderen Spitze.
- **Zum “Seitenleitwerk” gehören:** Die Ecken, die das kleinere Dreieck auf einer Seite des Rumpfes bilden. Das wären Ecke 4, Ecke 5 und Ecke 6. Diese bilden das vertikale Stabilisierungselement am Heck des Flugzeugs.
- **Die vordere Spitze des Flugzeugs** wird durch **Ecke 3** definiert, da diese am weitesten in positive X-Richtung zeigt, was typischerweise nach vorne gerichtet ist.

- **Die rechte Flügelspitze** des Flugzeugs kann durch **Ecke 2** definiert werden, wenn wir davon ausgehen, dass die positive Y-Richtung nach rechts zeigt. Diese Ecke hat einen positiven Y-Wert und befindet sich am hinteren Ende des Flugzeugs auf der rechten Seite.
- **Die obere Spitze des Leitwerks** wird durch **Ecke 5** definiert, da diese in negativer Z-Richtung zeigt, was nach oben zeigt, wenn das Flugzeug auf dem Boden steht.
- **Die Ecke, die genau im Ursprung liegt**, ist **Ecke 6**. Diese hat die Koordinaten  $[0\ 0\ 0]$ , was dem Ursprung des Koordinatensystems entspricht. Dieser Punkt könnte als Referenzpunkt für das Flugzeugmodell dienen, etwa als der Punkt, an dem das Seitenleitwerk auf den Rumpf trifft.

*Das hätte ich nicht erwartet!*

## 4 Frage

Definieren Sie in der Matrix `flaechen` die Eckenindizes der beiden Dreiecke.

Wie viele Zeilen und Spalten hat demzufolge `flaechen`? Warum eigentlich?

### 4.1 Antwort

Um die Eckenindizes der beiden Dreiecke zu definieren, die das 3D-Modell des Flugzeugs ausmachen, erstellen wir die Matrix `flaechen`. Diese Matrix ordnet jeder Fläche (in diesem Fall jedem der beiden Dreiecke) eine Reihe von Ecken zu. Da jedes Dreieck durch drei Ecken definiert wird, enthält jede Zeile der Matrix `flaechen` die Indizes von drei Ecken, die ein Dreieck bilden.

Für das Flugzeugmodell, das aus einem Rumpf und einem Seitenleitwerk besteht, können wir die Matrix `flaechen` wie folgt definieren:

- Für den **Rumpf** des Flugzeugs, der durch die Ecken 1, 2 und 3 gebildet wird.
- Für das **Seitenleitwerk** des Flugzeugs, das durch die Ecken 4, 5 und 6 gebildet wird.

```
flaechen = [ ...
1 2 3; ... % Dreieck des Rumpfs
4 5 6];    % Dreieck des Seitenleitwerks
```

Die Matrix `flaechen` hat demzufolge:

- **2 Zeilen**, da wir zwei Dreiecke (zwei Flächen) haben: eine Zeile für den Rumpf und eine Zeile für das Seitenleitwerk.
- **3 Spalten**, weil jedes Dreieck durch drei Ecken definiert ist und wir die Indizes dieser drei Ecken benötigen, um das Dreieck zu definieren.

Die Anzahl der Zeilen entspricht der Anzahl der Flächen (Dreiecke) im Modell, und die Anzahl der Spalten entspricht der Anzahl der Ecken, die benötigt werden, um eine Fläche in einem dreidimensionalen Raum zu definieren (für ein Dreieck sind das drei Ecken).

*Sehr richtig.*

## 5 Frage

Verwenden Sie den Befehl `patch`, um aus den gerade definierten Ecken- und Flächenmatrizen das 3D-Modell zu erzeugen. Nutzen Sie dabei den Parameter `FaceVertexCData`, um den Rumpf hellgrau und das Seitenleitwerk mittelgrau einzufärben.

Was passiert, wenn Sie dabei die `FaceColor` nicht auf `flat` setzen?

### 5.1 Antwort

Um das 3D-Modell des Flugzeugs mit den definierten Ecken- und Flächenmatrizen zu erzeugen und dabei unterschiedliche Farben für den Rumpf und das Seitenleitwerk zu verwenden, können Sie den MATLAB-Befehl `patch` verwenden. Der Parameter `FaceVertexCData` ermöglicht es, Farben für die einzelnen Vertices (Ecken) anzugeben, und durch Setzen von `FaceColor` auf `'flat'` werden die Farben der Flächen (Dreiecke) basierend auf den Farben der Vertices festgelegt. Hier ein Beispielcode:

```
% Definierte Eckenmatrix
ecken = [ ...
-1 -1 0; ... % Ecke 1
-1 1 0; ... % Ecke 2
 2 0 0; ... % Ecke 3
-1 0 0; ... % Ecke 4
-1 0 -1; ... % Ecke 5
 0 0 0]; % Ecke 6

% Definierte Flächenmatrix
flaechen = [ ...
1 2 3; ... % Dreieck des Rumpfs
4 5 6]; % Dreieck des Seitenleitwerks

% Farben definieren (hellgrau für den Rumpf, mittelgrau für das Seitenleitwerk)
farben = [0.8 0.8 0.8; 0.5 0.5 0.5];

% Modell erzeugen
figure;
patch('Vertices', ecken, 'Faces', flaechen, ...
      'FaceVertexCData', farben, 'FaceColor', 'flat');

% Achsensystem und Blickrichtung anpassen
axis equal;
view(3);
xlabel('X');
ylabel('Y');
zlabel('Z');
rotate3d on;
```

Wenn Sie den Parameter `FaceColor` nicht auf `'flat'` setzen, gibt es verschiedene Verhaltensweisen, je nachdem, welchen Wert Sie stattdessen verwenden:

- **'interp'**: Die Farben der Flächen werden basierend auf einer Interpolation der Vertex-Farben berechnet. Dies führt zu einem Gradienten innerhalb der Flächen, abhängig von den Farben der Ecken. In unserem Fall, wo wir `FaceVertexCData` verwenden, um spezifische Farben für die Flächen zu definieren, würde dies nicht das gewünschte Ergebnis liefern, da die Farben zwischen den Ecken interpoliert und nicht als einheitliche Flächenfarben dargestellt werden.
- **Nicht setzen oder auf einen anderen Wert setzen (z.B. 'none')**: Wenn Sie `FaceColor` nicht setzen oder auf `'none'` setzen, wird die Farbe der Flächen nicht basierend auf `FaceVertexCData` festgelegt. Bei `'none'` würden die Flächen beispielsweise transparent sein, sodass die Flächenfarben nicht sichtbar wären.

Um also sicherzustellen, dass jede Fläche einheitlich in der Farbe gefärbt wird, die durch die zugehörigen Einträge in `FaceVertexCData` definiert ist, ist es wichtig, `FaceColor` auf `'flat'` zu setzen.

*Passt alles.*

## 6 Frage

Jetzt beginnt die Simulation. Setzen Sie dazu die Variable `delta_t` auf 0.01.

Welche Bedeutung könnte eine Variable mit dem Namen `delta_t` wohl haben?

### 6.1 Antwort

Die Variable `delta_t` repräsentiert üblicherweise den Zeitunterschied oder die Zeitdifferenz zwischen aufeinanderfolgenden Schritten in einer Simulation oder Berechnung. Der Begriff stammt aus dem lateinischen Wort “delta” ( $\Delta$ ), das in der Mathematik und Physik oft als Symbol für eine Differenz oder Veränderung verwendet wird, und dem englischen Wort “time”, das Zeit bedeutet.

In zeitbasierten Simulationen oder numerischen Berechnungen, wie zum Beispiel der Bewegungssimulation eines Objekts oder der Lösung zeitabhängiger Differentialgleichungen, gibt `delta_t` an, wie viel “virtuelle Zeit” zwischen zwei Berechnungsschritten verstreicht. Ein kleinerer Wert für `delta_t` führt zu feineren, genaueren Simulationen, benötigt aber mehr Berechnungsschritte und somit mehr Rechenzeit, während ein größerer Wert für `delta_t` die Simulation beschleunigt, aber auf Kosten der Genauigkeit gehen kann.

In Ihrem Kontext bedeutet `delta_t = 0.01`, dass zwischen den einzelnen Schritten der Simulation eine virtuelle Zeit von 0,01 Zeiteinheiten verstreicht. Je nach Kontext könnten diese Zeiteinheiten Sekunden, Minuten oder eine andere Maßeinheit der Zeit sein. Dieser Wert wird typischerweise verwendet, um den zeitlichen Verlauf von Veränderungen in der Simulation präzise zu steuern.

*Sehr allgemeine aber hilfreiche Antwort.*

## 7 Frage

Als erstes wollen Sie eine Seitenwindlandung simulieren. Dabei fliegt das Flugzeug natürlich vorwärts, ein bisschen nach unten und ein bisschen nach rechts. Starten Sie eine Schleife, in der die Variable `t` mit der Abtastzeit `delta_t` von null bis fünf läuft.

Wie viele Einzelschritte werden also in der Simulation gerechnet werden?

## 7.1 Antwort

Um die Anzahl der Einzelschritte in der Simulation zu berechnen, die von

$$t = 0$$

bis

$$t = 5$$

mit einem Zeitintervall

$$\Delta t = 0.01$$

läuft, kann man die Gesamtzeit durch das Zeitintervall teilen und dann 1 hinzufügen, da sowohl der Start- als auch der Endpunkt in der Zählung eingeschlossen sind.

Formel:

$$\text{Anzahl der Schritte} = \frac{\text{Endzeit} - \text{Startzeit}}{\Delta t} + 1$$

Setzen wir die gegebenen Werte ein:

$$\text{Anzahl der Schritte} = \frac{5 - 0}{0.01} + 1 = \frac{5}{0.01} + 1 = 500 + 1 = 501$$

Also werden in der Simulation insgesamt 501 Einzelschritte gerechnet.

*Prima. Sie hat sogar das Gartenzaunproblem erkannt und beschrieben. Aber es fehlt der Matlab-Code. Dazu reicht eine gedankenleserische Aufforderung:*

## 7.2 Frage

Matlab?

### 7.2.1 Antwort

Um diese Berechnung und die Schleife in MATLAB zu implementieren, würde der Code wie folgt aussehen:

```
% Initialisierung der Variablen
delta_t = 0.01; % Zeitintervall
t_end = 5; % Endzeit
t_start = 0; % Startzeit

% Berechnung der Anzahl der Schritte
num_steps = (t_end - t_start) / delta_t + 1;

% Ausgabe der Anzahl der Schritte
disp(['Anzahl der Schritte: ', num2str(num_steps)]);

% Simulationsschleife
for t = t_start:delta_t:t_end
    % Hier kommen die Simulationsschritte hin
end
```

In diesem Code-Snippet wird zuerst `delta_t` auf 0.01 gesetzt, was das Zeitintervall zwischen jedem Schritt angibt. Die Variable `t_end` definiert den Endpunkt der Simulation bei 5 Sekunden, und `t_start` definiert den Startpunkt bei 0 Sekunden. Die Variable `num_steps` wird berechnet, indem die Gesamtzeit durch das Zeitintervall dividiert und 1 hinzugefügt wird, um den Startpunkt einzuschließen. Schließlich wird eine `for`-Schleife verwendet, um von `t_start` bis `t_end` mit Schritten von `delta_t` zu iterieren. In jedem Schritt der Schleife könnten Simulationsschritte ausgeführt werden, die hier mit einem Platzhalterkommentar markiert sind.

## 8 Frage

In der Schleife verwenden Sie den folgenden Befehl:

```
patch_flugzeug.Vertices = ecken + [t, t/2, t/2];
```

Erläutern Sie genau, was der Befehl bewirkt. Wo befindet sich der “Schwerpunkt” des Flugzeugs am Ende der Simulation?

### 8.1 Antwort

Der MATLAB-Befehl

```
patch_flugzeug.Vertices = ecken + [t, t/2, t/2];
```

innerhalb der Simulationsschleife aktualisiert die Position der Eckpunkte des Flugzeugs zu jedem Zeitpunkt ( $t$ ) der Simulation. Dabei wird die ursprüngliche Position der Eckpunkte (definiert in der Matrix `ecken`) durch das Hinzufügen eines Vektors  $[t, t/2, t/2]$  zu jedem Eckpunkt modifiziert. Dieser Vektor repräsentiert eine Transformation im dreidimensionalen Raum, die auf jede Ecke angewendet wird.

Der Vektor  $[t, t/2, t/2]$  bewirkt Folgendes für die Position jeder Ecke des Flugzeugs:

- **In der X-Richtung** wird der Wert ( $t$ ) hinzugefügt, was bedeutet, dass das Flugzeug mit jedem Schritt der Simulation linear in X-Richtung (vorwärts) bewegt wird.
- **In der Y-Richtung** wird der Wert ( $t/2$ ) hinzugefügt, was eine langsamere Bewegung in Y-Richtung (nach rechts) zur Folge hat.
- **In der Z-Richtung** wird ebenfalls ( $t/2$ ) hinzugefügt, was bedeutet, dass das Flugzeug auch in Z-Richtung (nach unten) bewegt wird, jedoch in der gleichen Rate wie in Y-Richtung.

Zusammengefasst sorgt dieser Befehl dafür, dass das Flugzeug im Laufe der Zeit vorwärts, nach rechts und nach unten fliegt. Die Bewegung nach rechts und unten erfolgt dabei halb so schnell wie die Vorwärtsbewegung.

Am **Ende der Simulation** bei ( $t = 5$ ):

- Die X-Koordinate jedes Eckpunkts hat sich um (5) Einheiten erhöht.
- Die Y-Koordinate jedes Eckpunkts hat sich um ( $5/2 = 2.5$ ) Einheiten erhöht.
- Die Z-Koordinate jedes Eckpunkts hat sich ebenfalls um ( $5/2 = 2.5$ ) Einheiten erhöht.

Der “Schwerpunkt” des Flugzeugs, betrachtet man ihn als den durchschnittlichen Punkt aller Eckpunkte, befindet sich also am Ende der Simulation 5 Einheiten vorwärts, 2.5 Einheiten rechts und 2.5 Einheiten unterhalb der ursprünglichen Position.

*Großartig!*

## 9 Frage

Um die Simulation etwa in Echtzeit ablaufen zu lassen, verwenden Sie in der Schleife den Befehl `pause` mit dem Parameter `delta_t`.

Wie arbeitet diese “Echtzeitanpassung”?

Warum kann sie aber niemals exakt funktionieren? Ist die Simulation schneller oder langsamer als die Echtzeit? Spendieren Sie einmal ein `tic` vor und ein `toc` nach der Schleife.

Welches alternative Verfahren haben Sie schon kennengelernt, um Befehle zu festgelegten Zeiten auszuführen?

Was würde geschehen, wenn Sie auf den `pause`-Befehl verzichten und stattdessen den Befehl `drawnow` verwenden würden?

### 9.1 Antwort

#### 9.1.1 Wie arbeitet diese “Echtzeitanpassung”?

Die Verwendung des `pause(delta_t)` Befehls in einer Schleife soll die Ausführung der Simulation annähernd in Echtzeit ablaufen lassen. Der Befehl `pause(delta_t)` unterbricht die Ausführung des MATLAB-Codes für die Dauer von `delta_t` Sekunden. Wenn `delta_t` der realen Zeit zwischen den Simulationsschritten entspricht, sollte die Simulation in etwa in Echtzeit ablaufen.

#### 9.1.2 Warum kann sie aber niemals exakt funktionieren?

Es gibt mehrere Gründe, warum die Simulation trotz der Pause nicht exakt in Echtzeit ablaufen kann:

- **Verarbeitungszeit:** Die Berechnungen und Anweisungen in jedem Schleifendurchlauf benötigen Zeit. Diese Verarbeitungszeit wird von der Pause nicht berücksichtigt, was bedeutet, dass jeder Zyklus tatsächlich länger dauert als `delta_t`.
- **Scheduling und Timing-Genauigkeit:** Betriebssysteme garantieren in der Regel nicht die exakte Timing-Genauigkeit für Pausen oder Wartezeiten, besonders in einer Multitasking-Umgebung. Die tatsächliche Wartezeit kann etwas variieren.

#### 9.1.3 Ist die Simulation schneller oder langsamer als die Echtzeit?

Aufgrund der zusätzlichen Verarbeitungszeit für die Befehle in jedem Schleifendurchlauf und möglichen Verzögerungen durch das Betriebssystem wird die Simulation tendenziell langsamer als die Echtzeit ablaufen.

#### 9.1.4 Tic und Toc

Durch den Einsatz von `tic` vor und `toc` nach der Schleife können Sie die tatsächliche Laufzeit der Schleife messen. Zum Beispiel:

```
tic; % Startzeit messen
for t = t_start:delta_t:t_end
    % Simulationsschritte
    pause(delta_t); % Echtzeitanpassung
```

```
end
toc; % Endzeit messen und Differenz anzeigen
```

### 9.1.5 Alternatives Verfahren

Ein alternatives Verfahren zur Ausführung von Befehlen zu festgelegten Zeiten ist die Verwendung eines Timers. MATLAB-Timer-Objekte können so konfiguriert werden, dass sie Funktionen oder Befehle in regelmäßigen Intervallen ausführen. Dies bietet mehr Flexibilität und kann in manchen Fällen genauer sein als eine einfache Pause.

### 9.1.6 Verwendung von `drawnow` statt `pause`

Der Befehl `drawnow` zwingt MATLAB, alle anstehenden Grafikoperationen sofort auszuführen, ohne den Code zu unterbrechen. Wenn Sie `pause(delta_t)` durch `drawnow` ersetzen, wird die Simulation ohne Verzögerungen durchlaufen, was die Ausführung beschleunigt. `drawnow` ist besonders nützlich, um sicherzustellen, dass die Benutzeroberfläche reaktionsfähig bleibt oder um die Aktualisierung von Grafiken in Echtzeit zu ermöglichen, aber es bietet keine Möglichkeit, die Ausführung zu verlangsamen, um eine Echtzeit-Simulation zu erreichen.

*Sehr schön.*

## 10 Frage

Im Folgenden möchten Sie das Flugzeug nacheinander gieren, nicken und rollen lassen. Dazu verwenden Sie das Unterprogramm `rotieren_3d` vom Server.

Erklären Sie detailliert, wie das Unterprogramm funktioniert.

### 10.1 Antwort

Das Unterprogramm `rotieren_3d` in MATLAB führt eine dreidimensionale Rotation eines Objekts oder eines Koordinatensystems um die drei Hauptachsen (x, y, z) durch. Die Funktion nimmt drei Winkel als Eingabeparameter: `ph` für die Rotation um die x-Achse (Rollen), `th` für die Rotation um die y-Achse (Nicken) und `ps` für die Rotation um die z-Achse (Gieren). Das Ergebnis ist eine Rotationsmatrix, die die kombinierte Rotation darstellt.

Hier ist, wie das Unterprogramm funktioniert:

#### 1. Erzeugung der Rotationsmatrizen:

- `m_ps` ist die Rotationsmatrix für die Rotation um die z-Achse (Gieren, Psi `ps`). Sie verändert die x- und y-Koordinaten, während die z-Koordinate unverändert bleibt.
- `m_th` ist die Rotationsmatrix für die Rotation um die y-Achse (Nicken, Theta `th`). Sie verändert die x- und z-Koordinaten, während die y-Koordinate unverändert bleibt.
- `m_ph` ist die Rotationsmatrix für die Rotation um die x-Achse (Rollen, Phi `ph`). Sie verändert die y- und z-Koordinaten, während die x-Koordinate unverändert bleibt.

#### 2. Matrixmultiplikation:

- Die finale Rotationsmatrix `matrix` wird durch Multiplikation der drei individuellen Rotationsmatrizen in der Reihenfolge `m_ps * m_th * m_ph` berechnet. Diese Reihenfolge entspricht einer Rotation um die z-Achse, gefolgt von einer Rotation um die y-Achse und schließlich einer Rotation um die x-Achse.

- Die Multiplikation von Matrizen ist nicht kommutativ, was bedeutet, dass die Reihenfolge der Multiplikation das Ergebnis beeinflusst. Die spezifische Reihenfolge bestimmt, wie die Rotationen kombiniert werden.

### 3. Rotation im dreidimensionalen Raum:

- Die resultierende Matrix `matrix` kann verwendet werden, um die Koordinaten eines Punktes oder eines Objekts im Raum zu transformieren, was einer sequenziellen Rotation um die z-Achse, y-Achse und x-Achse entspricht.
- Jede Rotationsmatrix dreht Punkte im Raum um den Ursprung (0,0,0) entlang der jeweiligen Achse.

Zusammenfassend erlaubt `rotieren_3d` die Kombination von drei Rotationen um die Hauptachsen in einer einzigen Transformation. Diese Funktion ist nützlich, um die Orientierung eines Objekts im Raum zu ändern, beispielsweise in Simulationen oder bei der visuellen Darstellung dreidimensionaler Objekte.

*Gute Erklärungen. Sogar die Drehreihenfolge ist richtig. Und sie kennt die Langnamen der Winkel.*

## 11 Frage

Zum Gieren starten Sie eine Schleife für den Winkel `ps` von 0 bis 1 in Schritten von `delta_t`. In der Schleife verwenden Sie den Befehl

```
patch_flugzeug.Vertices = (rotieren_3d (0, 0, ps) * ecken')';
```

Was macht der Befehl?

Wird dabei die Matrix `ecken` verändert?

Wozu sind insbesondere die Hochkommata notwendig? Erinnern Sie sich mal daran, welche Regeln bei einer Matrizenmultiplikation gelten.

### 11.1 Antwort

Der angegebene MATLAB-Befehl

```
patch_flugzeug.Vertices = (rotieren_3d (0, 0, ps) * ecken')';
```

führt die Rotation des Flugzeugs um die z-Achse (Gieren) durch und aktualisiert die Position seiner Eckpunkte entsprechend dem aktuellen Rotationswinkel `ps`. Hier ist die detaillierte Erklärung der Komponenten dieses Befehls und was genau er bewirkt:

1. **`rotieren_3d (0, 0, ps)`:** Dies ruft die Funktion `rotieren_3d` auf, um eine Rotationsmatrix zu erzeugen, die eine Rotation um die z-Achse repräsentiert. Die Winkel für Rollen (`ph`) und Nicken (`th`) sind hierbei auf 0 gesetzt, da nur um die z-Achse (Gieren) rotiert wird. Der Winkel `ps` gibt den Rotationsgrad um die z-Achse an.
2. **`ecken'`:** Das Hochkomma (`'`) ist der Transpositionsoperator in MATLAB. Er wandelt die Matrix `ecken` von einer Matrix der Größe ( $m \times n$ ) in eine transponierte Matrix der Größe ( $n \times m$ ) um. Da `ecken` eine Matrix der Eckpunkte des Flugzeugs ist, deren jede Zeile einen Punkt (Ecke) im 3D-Raum darstellt, wechselt die Transposition die Zeilen und Spalten. Dies ist notwendig, weil die Rotationsmatrix auf die Spaltenvektoren der Eckpunkte angewendet werden muss.

3. **Matrizenmultiplikation (\*):** Die Rotationsmatrix, die von `rotieren_3d` zurückgegeben wird, wird mit den transponierten Eckpunkten multipliziert. Diese Operation dreht die Punkte um die z-Achse. Die Matrizenmultiplikation erfordert, dass die Anzahl der Spalten in der ersten Matrix der Anzahl der Zeilen in der zweiten Matrix entspricht, daher ist die Transposition der Eckpunkte notwendig.
4. **Erneute Transposition ('):** Nach der Multiplikation wird das Ergebnis erneut transponiert ('). Dies konvertiert die Matrix zurück in das ursprüngliche Format, in dem jede Zeile einen Punkt im Raum darstellt. Diese Umwandlung ist notwendig, um die aktualisierten Eckpunkte dem `Vertices`-Feld von `patch_flugzeug` zuzuweisen, welches erwartet, dass jede Zeile einen 3D-Punkt repräsentiert.
5. **Aktualisierung der Eckpunkte (`patch_flugzeug.Vertices = ...`):** Schließlich werden die rotierten und zurücktransponierten Eckpunkte verwendet, um die `Vertices`-Eigenschaft des `patch`-Objekts `patch_flugzeug` zu aktualisieren. Dies ändert die visuelle Darstellung des Flugzeugs in MATLAB entsprechend der durchgeführten Rotation.

Die Matrix `ecken` wird in diesem Prozess nicht direkt verändert. Die Operationen arbeiten auf Kopien oder abgeleiteten Matrizen. Die ursprüngliche `ecken`-Matrix bleibt unverändert, was bedeutet, dass die ursprünglichen Koordinaten der Eckpunkte erhalten bleiben und für zukünftige Transformationen oder Berechnungen verfügbar sind. Die Transpositionsoperationen sind erforderlich, um die Regeln der Matrizenmultiplikation zu erfüllen und die korrekte Anwendung der Rotation auf die Eckpunkte zu gewährleisten.

*Hätte ich nicht besser gekonnt.*

## 12 Frage

Führen Sie anschließend die Nick- und Rollschleifen durch. Achten Sie dabei darauf, dass die vorher durchgeführten Drehungen erhalten bleiben; am Ende soll das Flugzeug also gleichzeitig einen Gier-, Nick- und Rollwinkel von jeweils 1 besitzen.

### 12.1 Antwort

Um das Flugzeug nacheinander gieren, nicken und rollen zu lassen und dabei sicherzustellen, dass die vorher durchgeführten Drehungen erhalten bleiben, müssen Sie die Rotationswinkel akkumulieren. Das bedeutet, dass Sie die Schleifen so durchlaufen, dass jede neue Rotation auf der vorherigen aufbaut. Am Ende soll das Flugzeug einen Gier-, Nick- und Rollwinkel von jeweils 1 haben. Hier ist ein Beispiel, wie dies in MATLAB implementiert werden kann:

```
% Initialwerte
delta_t = 0.01; % Schrittweite
ps_final = 1; % Endwert für Gieren
th_final = 1; % Endwert für Nicken
ph_final = 1; % Endwert für Rollen

% Gieren
for ps = 0:delta_t:ps_final
    % Berechnen der neuen Position der Eckpunkte
    patch_flugzeug.Vertices = (rotieren_3d(0, 0, ps) * ecken')';
```

```

    pause(delta_t);
end

% Nicken, Gierwinkel bleibt erhalten
for th = 0:delta_t:th_final
    % Berechnen der neuen Position der Eckpunkte
    patch_flugzeug.Vertices = (rotieren_3d(0, th, ps_final) * ecken')';
    pause(delta_t);
end

% Rollen, Gier- und Nickwinkel bleiben erhalten
for ph = 0:delta_t:ph_final
    % Berechnen der neuen Position der Eckpunkte
    patch_flugzeug.Vertices = (rotieren_3d(ph, th_final, ps_final) * ecken')';
    pause(delta_t);
end

```

### 12.1.1 Erklärung

- Im ersten Schritt (*ps*-Schleife) wird das Flugzeug um die z-Achse (Gieren) gedreht, bis es den gewünschten Gierwinkel *ps\_final* erreicht.
- Im zweiten Schritt (*th*-Schleife) wird das Flugzeug um die y-Achse (Nicken) gedreht, wobei der Gierwinkel *ps\_final* aus dem ersten Schritt erhalten bleibt. Jetzt wird der Nickwinkel *th\_final* angewendet.
- Im dritten Schritt (*ph*-Schleife) wird das Flugzeug um die x-Achse (Rollen) gedreht, wobei sowohl der Gierwinkel *ps\_final* als auch der Nickwinkel *th\_final* aus den vorherigen Schritten erhalten bleiben. Schließlich wird der Rollwinkel *ph\_final* angewendet.

### 12.1.2 Wichtig

- In jedem Schritt wird die Funktion `rotieren_3d` mit den aktuellen Winkeln für Gieren, Nicken und Rollen aufgerufen, um die entsprechende Rotationsmatrix zu erhalten. Diese Matrix wird dann verwendet, um die Eckpunkte des Flugzeugs zu rotieren.
- Die Funktion `pause(delta_t)` wird verwendet, um die Simulation in etwa in Echtzeit ablaufen zu lassen.
- Nach Abschluss aller Schritte besitzt das Flugzeug die geforderten Rotationen um alle drei Achsen, wobei die vorherigen Rotationen jeweils erhalten bleiben.

*Perfekt*

## 13 Frage

Sie möchten das Flugzeug jetzt einen Looping fliegen lassen. Dazu muss es in der Schleife gleichzeitig nicken und sich auf einer Kreisbahn bewegen. Der Kreisradius soll dabei 4 betragen. Legen Sie los.

Wie erreichen Sie, dass im ersten Befehl in der Schleife die Rotation durchgeführt wird und im zweiten Befehl zusätzlich die Translation?

## 13.1 Antwort

Um das Flugzeug einen Looping fliegen zu lassen, bei dem es gleichzeitig nickt und sich auf einer Kreisbahn mit einem Radius von 4 bewegt, können Sie zwei Schritte in jeder Iteration der Schleife durchführen:

1. **Rotation:** Nutzen Sie die Rotationsmatrix aus `rotieren_3d`, um die Ecken des Flugzeugs um die gewünschten Winkel zu rotieren, was dem Nicken entspricht.
2. **Translation:** Verschieben Sie das Flugzeug entlang einer Kreisbahn, indem Sie seine Position aktualisieren.

Hier ist, wie Sie dies in MATLAB umsetzen können:

```
% Parameter für die Kreisbahn
radius = 4;
% Winkel für den kompletten Kreis
kreisWinkel = 0:delta_t:2*pi;

for th = kreisWinkel
    % Rotation um die y-Achse (Nicken)
    neueEcken = (rotieren_3d(0, th, 0) * ecken)';

    % Berechnung der Kreisposition
    x = radius * cos(th);
    z = radius * sin(th) - radius; % Zentriert den Kreis unterhalb des Ursprungs

    % Translation entlang einer Kreisbahn
    patch_flugzeug.Vertices = neueEcken + [x, 0, z];

    drawnow; % Aktualisiert die Grafik
    pause(delta_t); % Pause, um die Simulation in etwa in Echtzeit zu halten
end
```

### 13.1.1 Erläuterung:

- **Rotationsmatrix:** Die Funktion `rotieren_3d(0, th, 0)` generiert eine Rotationsmatrix für die Rotation um die y-Achse, basierend auf dem aktuellen Winkel `th`. Das bedeutet, dass das Flugzeug während des Loopings nickt.
- **Translation auf einer Kreisbahn:** Durch die Berechnung von `x` und `z` Koordinaten auf Basis des aktuellen Winkels `th` und des vorgegebenen Radius von 4, können Sie das Flugzeug entlang einer vertikalen Kreisbahn bewegen. Die Z-Koordinate wird um `radius` verringert, um den Kreismittelpunkt unter den Ursprung zu setzen, was bewirkt, dass das Flugzeug einen vollständigen Looping fliegt, der mit dem Boden beginnt und endet.
- **Aktualisierung der Eckpunkte:** `patch_flugzeug.Vertices = neueEcken + [x, 0, z];` verschiebt das Flugzeug nach der Rotation zu seiner neuen Position auf der Kreisbahn. Hier wird die Translation hinzugefügt, nachdem die Eckpunkte rotiert wurden.

Durch diese Kombination von Rotation und Translation in jedem Schritt der Schleife wird das Flugzeug in der Simulation einen Looping fliegen, der sowohl das Nicken als auch die Bewegung auf einer Kreisbahn beinhaltet.

Nicht ganz:

## 13.2 Frage

Überleg noch mal, wo du die trigonometrischen Funktionen einsetzen musst. Das Looping startet in  $(0|0)$  als unterem Kreispunkt.

### 13.2.1 Antwort

Um sicherzustellen, dass das Looping korrekt startet, mit dem unteren Kreispunkt bei  $(0|0)$  und unter Verwendung der richtigen trigonometrischen Funktionen für die Bewegung auf der Kreisbahn, sollte die Berechnung der Position so angepasst werden, dass die y- und z-Koordinaten entsprechend geändert werden, während die x-Koordinate für die Vorwärtsbewegung des Flugzeugs genutzt wird. Für ein vertikales Looping mit einem Startpunkt am unteren Kreispunkt und unter der Annahme, dass das Flugzeug in der yz-Ebene looped, könnten die Anpassungen wie folgt aussehen:

```
% Kreisradius
radius = 4;
% Schrittweite für die Schleife, entspricht dem Nicken
delta_t = 0.01;

% Gesamtdauer des Loopings, die einen vollständigen Kreis (2*pi) darstellt
kreisWinkel = 0:delta_t:2*pi;

for th = kreisWinkel
    % Rotation um die y-Achse (Nicken) - hier mit th als Nickwinkel
    neueEcken = (rotieren_3d(0, th, 0) * ecken')';

    % Berechnung der Kreisbahn für die Translation
    y = radius * sin(th); % Änderung in der y-Richtung (vertikale Achse)
    z = -radius * (1 - cos(th)); % Änderung in der z-Richtung (Start im unteren Kreispunkt)

    % Aktualisierung der Position des Flugzeugs um die berechnete Translation
    patch_flugzeug.Vertices = neueEcken + [0, y, z];

    drawnow; % Sofortige Aktualisierung der Grafik
    pause(delta_t); % Kurze Pause, um die Simulation realistisch zu halten
end
```

### 13.2.2 Korrektur in der trigonometrischen Berechnung:

In diesem Ansatz:

- **Sinus ( $\sin(\text{th})$ )** wird verwendet, um die Verschiebung entlang der y-Achse (Höhe) zu berechnen. Dies erzeugt eine sinusförmige Bewegung, die das Flugzeug auf und ab entlang der vertikalen Achse bewegt.
- **Cosinus ( $\cos(\text{th})$ )** wird verwendet, um die Verschiebung entlang der z-Achse zu berechnen. Der Ausdruck  $-\text{radius} * (1 - \cos(\text{th}))$  bewirkt, dass das Flugzeug bei  $(\text{th} = 0)$  am untersten Punkt des Kreises startet und dann aufsteigt. Durch  $1 - \cos(\text{th})$  wird

sichergestellt, dass dieser Wert bei ( $th = 0$ ) gleich 0 ist und bei ( $th = \pi$ ) (dem höchsten Punkt des Loopings) den maximalen Wert erreicht.

Diese Anpassung ermöglicht ein korrektes Looping, bei dem das Flugzeug vom untersten Punkt des Kreises startet, aufsteigt, den obersten Punkt erreicht und dann zurück zum Ausgangspunkt kehrt, alles während es kontinuierlich um die y-Achse nickt.

*Immer noch nicht ganz:*

### 13.3 Frage

Beachte, dass die x-Achse nach vorne, die y-Achse nach rechts und die z-Achse nach unten zeigt.

#### 13.3.1 Antwort

Angesichts der Orientierung, dass die x-Achse nach vorne, die y-Achse nach rechts und die z-Achse nach unten zeigt, und dem Ziel, das Flugzeug einen Looping fliegen zu lassen, der im Punkt (0|0) startet und dabei gleichzeitig nickt, müssen wir die Bewegung entsprechend anpassen. Für ein vertikales Looping entlang der xz-Ebene, wobei die x-Achse die Vorwärtsbewegung und die z-Achse die Höhenänderung darstellt, sieht die korrigierte Implementierung so aus:

```
% Parameter für die Kreisbahn und das Looping
radius = 4;
delta_t = 0.01; % Schrittweite

% Berechnung der Winkel für ein vollständiges Looping
kreisWinkel = 0:delta_t:2*pi;

for th = kreisWinkel
    % Rotation um die y-Achse (Nicken) entsprechend dem Winkel th
    neueEcken = (rotieren_3d(0, th, 0) * ecken)';

    % Berechnung der Position auf der Kreisbahn
    x = radius * sin(th); % Vorwärtsbewegung entlang der x-Achse
    z = -radius * (1 - cos(th)); % Bewegung in der Höhe entlang der z-Achse

    % Aktualisierung der Eckpunkte des Flugzeugs
    patch_flugzeug.Vertices = neueEcken + [x, 0, z];

    drawnow; % Aktualisiert die Grafik
    pause(delta_t); % Hält die Simulation in Echtzeit an
end
```

#### 13.3.2 Korrektur für die Achsenorientierung:

- **x-Achse (Vorwärtsbewegung):** Die Verwendung von  $\sin(th)$  für die x-Koordinate simuliert die Vorwärtsbewegung des Flugzeugs entlang der x-Achse. Das Flugzeug startet am unteren Punkt des Loopings und bewegt sich vorwärts, während es aufsteigt.
- **y-Achse (Seitenbewegung):** Da das Looping in dieser Situation keine seitliche Bewegung beinhaltet, bleibt die y-Koordinate unverändert (0).

- **z-Achse (Höhe):** Die Verwendung von  $-\text{radius} * (1 - \cos(\text{th}))$  für die z-Koordinate passt die Höhenposition des Flugzeugs an. Das Minuszeichen vor dem Ausdruck stellt sicher, dass das Flugzeug nach oben in die positive z-Richtung steigt, da die z-Achse nach unten zeigt.

Diese Änderungen berücksichtigen die spezifische Orientierung der Achsen und ermöglichen es dem Flugzeug, während des Loopings kontinuierlich zu nicken, während es gleichzeitig einen korrekten vertikalen Looping vollführt.

*Und plötzlich geht's!! Die Tatsache, dass ich ungeschickterweise gefordert hatte, dass der untere Punkt bei 0|0 statt sinnvollerweise bei 0|4 liegt, kann man ihr ja nun wirklich nicht anlasten.*

# 12 Sortierung

April 11, 2024

## 1 Frage

Sie möchten im Folgenden die Bogosort-Methode verwenden, um die vertauschten Buchstaben eines gesuchten Wortes solange zufällig zu tauschen, bis das gesuchte Wort herauskommt. Speichern Sie dazu das gesuchte Wort 'milch' in der Variable `gesucht` ab. Bestimmen Sie seine Länge und speichern Sie sie in `n_gesucht` ab.

Wie viele unterschiedliche Möglichkeiten gibt es, die Buchstaben des gesuchten Wortes anzuordnen? Wie können Sie diese Anzahl in Matlab berechnen?

### 1.1 Antwort

Um die Anzahl unterschiedlicher Möglichkeiten zu bestimmen, die Buchstaben des gesuchten Wortes 'milch' anzuordnen, kann man die Fakultät der Anzahl der Buchstaben berechnen. Das gesuchte Wort 'milch' hat 5 Buchstaben, daher gibt es  $5!$  ( $5$  Fakultät) Möglichkeiten, die Buchstaben anzuordnen.

Die Berechnung von  $5!$  in Matlab kann durch die Verwendung der `factorial`-Funktion erfolgen, die direkt die Fakultät einer gegebenen Zahl berechnet. Die Syntax für die Berechnung der Fakultät von 5 in Matlab wäre also:

```
n_gesucht = factorial(5);
```

Dieser Code berechnet die Anzahl unterschiedlicher Möglichkeiten, die Buchstaben des Wortes 'milch' anzuordnen, und speichert das Ergebnis in der Variable `n_gesucht`.

Die gesamte Matlab-Implementierung, einschließlich der Initialisierung des gesuchten Wortes und der Berechnung seiner Länge, würde wie folgt aussehen:

```
gesucht = 'milch'; % Das gesuchte Wort  
n_gesucht = factorial(length(gesucht)); % Berechnung der Anzahl der Anordnungsmöglichkeiten
```

Hierbei wird `length(gesucht)` verwendet, um die Länge des gesuchten Wortes zu bestimmen, und `factorial()` berechnet die Fakultät dieser Länge, was der Anzahl der Anordnungsmöglichkeiten entspricht.

*Yupp.*

## 2 Frage

Verwenden Sie den Befehl `randperm`, um sich einen zufälligen Indexvektor von der Länge des gesuchten Wortes erzeugen zu lassen und verwenden Sie ihn als Indexvektor des gesuchten Wortes,

um sich einen Startwert des gesuchten Wortes mit vertauschten Buchstaben zu beschaffen. Speichern Sie den Startwert in der Variablen `versuch` ab.

Erläutern Sie, was `randperm` macht und wie Sie sein Ergebnis als Indexvektor verwenden können.

## 2.1 Antwort

Der Befehl `randperm(n)` in Matlab erzeugt einen zufälligen Permutationsvektor der Länge `n`, der Zahlen von 1 bis `n` in zufälliger Reihenfolge enthält. Dieser Vektor kann als Indexvektor verwendet werden, um Elemente eines anderen Vektors oder Arrays in eine zufällige Reihenfolge zu bringen.

Um den Befehl `randperm` zu verwenden, um die Buchstaben des gesuchten Wortes ‘milch’ zufällig zu vertauschen und einen Startwert mit vertauschten Buchstaben zu erhalten, kann folgender Code verwendet werden:

```
gesucht = 'milch'; % Das gesuchte Wort
n = length(gesucht); % Länge des gesuchten Wortes
zufaelligerIndex = randperm(n); % Erzeugt einen zufälligen Indexvektor der Länge n
versuch = gesucht(zufaelligerIndex); % Verwendet den zufälligen Indexvektor, um die Buchstabe
```

In diesem Code:

- `gesucht` ist das ursprüngliche Wort, dessen Buchstaben vertauscht werden sollen.
- `n` ist die Länge des Wortes ‘milch’, also 5.
- `randperm(n)` erzeugt einen Vektor mit einer zufälligen Permutation von Zahlen von 1 bis `n`. Beispielsweise könnte eine solche Permutation `[3, 1, 4, 5, 2]` sein.
- `gesucht(zufaelligerIndex)` verwendet diesen Permutationsvektor als Indexvektor für das Wort ‘milch’. Das bedeutet, dass jeder Buchstabe des Wortes entsprechend der Reihenfolge der Zahlen im Permutationsvektor neu angeordnet wird. Wenn der Permutationsvektor beispielsweise `[3, 1, 4, 5, 2]` ist, wird das Wort zu ‘ichlm’ umgestellt.
- Das Ergebnis dieser Umstellung wird in der Variable `versuch` gespeichert, die somit einen Startwert des gesuchten Wortes mit vertauschten Buchstaben enthält.

Die Verwendung des zufälligen Indexvektors als Indexvektor für das Wort ermöglicht es uns, eine zufällige Anordnung der Buchstaben des Wortes zu erzeugen, was beim Bogosort-Algorithmus als Startpunkt für den Versuch, das Wort durch zufällige Permutationen zu sortieren, dient.

*Tolle Erklärung!*

## 3 Frage

Starten Sie eine `while`-Schleife, die abbricht, wenn der aktuelle `versuch` gleich dem gesuchten Wort ist.

Warum ist es sinnvoll, dabei den Befehl `strcmp` und nicht den Vergleichsoperator `==` zu verwenden?

### 3.1 Antwort

Um eine `while`-Schleife in Matlab zu starten, die so lange läuft, bis der aktuelle `versuch` gleich dem gesuchten Wort `gesucht` ist, können Sie den Befehl `strcmp` für den Vergleich verwenden. Hier ist ein Beispiel, wie das implementiert werden könnte:

```

gesucht = 'milch'; % Das gesuchte Wort
n = length(gesucht); % Länge des gesuchten Wortes

% Initialisierung des versuch-Vektors mit einer zufälligen Permutation
zufaelligerIndex = randperm(n);
versuch = gesucht(zufaelligerIndex);

% Zähler für die Anzahl der Versuche
anzahlVersuche = 1;

% Schleife, die so lange läuft, bis versuch gleich gesucht ist
while ~strcmp(versuch, gesucht)
    zufaelligerIndex = randperm(n);
    versuch = gesucht(zufaelligerIndex);
    anzahlVersuche = anzahlVersuche + 1; % Zähler erhöhen
end

disp(['Gefunden nach ', num2str(anzahlVersuche), ' Versuchen: ', versuch]);

```

In diesem Beispiel:

- `strcmp(versuch, gesucht)` vergleicht die beiden Strings `versuch` und `gesucht`. Die Schleife läuft so lange, bis `strcmp true` zurückgibt, was bedeutet, dass beide Strings identisch sind.
- Jeder Durchlauf der Schleife generiert eine neue zufällige Permutation der Buchstaben des gesuchten Wortes und erhöht den Zähler `anzahlVersuche` um eins.

Die Verwendung von `strcmp` anstelle des Vergleichsoperators `==` ist hier aus mehreren Gründen sinnvoll:

1. **Kompatibilität mit Strings:** Der Vergleichsoperator `==` ist für die Element-für-Element-Vergleiche von Arrays gedacht und funktioniert nicht zuverlässig beim Vergleich von Strings. `strcmp` hingegen ist speziell dafür ausgelegt, zwei Strings zu vergleichen und gibt ein logisches `true` oder `false` zurück, je nachdem, ob die Strings identisch sind oder nicht.
2. **Vermeidung von Dimensionalitätsproblemen:** Wenn zwei Strings unterschiedliche Längen haben, führt der Versuch eines Vergleichs mit `==` zu einem Fehler wegen unübereinstimmender Dimensionen. `strcmp` hingegen kann ohne Probleme Strings unterschiedlicher Längen vergleichen und gibt einfach `false` zurück, wenn die Strings nicht identisch sind.
3. **Klarheit und Lesbarkeit:** Die Verwendung von `strcmp` macht den Code klarer und expliziter in seiner Absicht, was die Lesbarkeit und Wartbarkeit des Codes verbessert.

*Super.*

## 4 Frage

In der Schleife müssen Sie jetzt nur noch den `randperm`-Befehl verwenden, um aus dem aktuellen `versuch` einen weiteren `versuch` zu erzeugen.

Warum ist es sinnvoller, in der Schleife den `disp`-Befehl zu verwenden, um sich den aktuellen Versuch anzeigen zu lassen als das Semikolon hinter der Zuweisung wegzulassen?

## 4.1 Antwort

In der Schleife den `disp`-Befehl zu verwenden, um sich den aktuellen Versuch anzeigen zu lassen, anstatt das Semikolon hinter der Zuweisung wegzulassen, hat mehrere Vorteile:

1. **Kontrollierte Ausgabe:** Mit dem `disp`-Befehl haben Sie die volle Kontrolle darüber, was genau und wie es ausgegeben wird. Sie können zusätzlichen Text hinzufügen, um die Ausgabe informativer zu gestalten, wie z.B. die Anzahl der bisherigen Versuche. Wenn Sie das Semikolon weglassen, erhalten Sie lediglich die rohe Ausgabe des Variableninhalts, ohne zusätzlichen Kontext.
2. **Klarheit und Lesbarkeit:** Der gezielte Einsatz von `disp` für Ausgaben macht den Code klarer und signalisiert explizit, dass eine Ausgabe an dieser Stelle beabsichtigt ist. Das Weglassen des Semikolons könnte hingegen als Versehen interpretiert werden, insbesondere in umfangreicheren Skripten oder Funktionen, wo nicht auf Anhieb klar ist, ob jede Ausgabe beabsichtigt ist.
3. **Vermeidung von unerwünschten Ausgaben:** In manchen Fällen möchten Sie möglicherweise die Ausgabe bestimmter Berechnungen oder Zuweisungen unterdrücken, die vor oder nach dem aktuellen Versuch in der Schleife stattfinden. Das konsequente Verwenden von Semikolons am Ende jeder Zeile, außer bei gezielten `disp`-Aufrufen, hilft dabei, nur die gewünschten Informationen auszugeben.
4. **Performance:** Während die Performance bei der Anzeige von ein paar Strings kaum beeinflusst wird, ist es in der Praxis eine gute Gewohnheit, Ausgaben auf das Notwendige zu beschränken. Unnötige Ausgaben können insbesondere in Schleifen, die sehr oft durchlaufen werden, die Ausführungszeit des Codes erhöhen.

Zusammenfassend verbessert die Verwendung von `disp` für Ausgaben in einer Schleife die Kontrolle, Klarheit und Effizienz des Codes.

Hier ein Beispiel, wie man den `disp`-Befehl nutzen könnte, um den aktuellen Versuch in jeder Iteration der Schleife anzuzeigen:

```
while ~strcmp(versuch, gesucht)
    zufaelligerIndex = randperm(n);
    versuch = gesucht(zufaelligerIndex);
    anzahlVersuche = anzahlVersuche + 1; % Zähler erhöhen
    disp(['Versuch Nr. ', num2str(anzahlVersuche), ': ', versuch]); % Aktuellen Versuch anzeigen
end
```

In diesem Beispiel wird bei jedem Durchlauf der Schleife eine klare und informative Ausgabe erzeugt, die den Benutzer über den Fortschritt des Algorithmus informiert.

*Ja, aber mein Hauptargument wäre, dass `disp` nur eine Zeile benötigt.*

## 5 Frage

Sie möchten am Ende der Suche wissen, wie viele Versuche der Algorithmus gebraucht hat.

Wie können Sie in einer `while`-Schleife mitzählen? Warum brauchen Sie dabei eine Initialisierung?

Wie viele Versuche kann der Algorithmus maximal benötigen?

Wie viele Versuche braucht er im Mittel, wenn Sie ihn ein paar Mal durchlaufen lassen?

## 5.1 Antwort

Um in einer `while`-Schleife mitzuzählen, wie viele Versuche der Algorithmus benötigt, können Sie eine Zählervariable verwenden, die bei jedem Durchlauf der Schleife inkrementiert wird. Die Initialisierung dieser Zählervariable ist notwendig, um einen Ausgangswert festzulegen, von dem aus gezählt wird. Typischerweise wird diese Zählervariable vor dem Eintritt in die Schleife auf einen Startwert, oft 0 oder 1, gesetzt, abhängig davon, wie Sie zählen möchten (beginnend bei 0 oder bei 1).

### 5.1.1 Initialisierung

Die Initialisierung ist wichtig, weil:

- Sie definiert den Anfangszustand der Zählervariablen.
- Ohne Initialisierung wäre die Variable zu Beginn der Schleife undefiniert, was zu einem Fehler führen könnte, wenn versucht wird, sie zu inkrementieren.
- Sie ermöglicht die korrekte Zählung von der ersten Iteration an.

### 5.1.2 Beispielcode

```
gesucht = 'milch'; % Das gesuchte Wort
n = length(gesucht);
anzahlVersuche = 0; % Initialisierung des Zählers

while true
    zufaelligerIndex = randperm(n);
    versuch = gesucht(zufaelligerIndex);
    anzahlVersuche = anzahlVersuche + 1; % Zähler inkrementieren
    if strcmp(versuch, gesucht)
        break; % Schleife beenden, wenn das Wort gefunden wurde
    end
end

disp(['Anzahl der Versuche: ', num2str(anzahlVersuche)]);
```

### 5.1.3 Maximale Anzahl an Versuchen

Theoretisch gibt es für den Bogosort-Algorithmus keine feste obere Grenze für die Anzahl der Versuche, da es theoretisch möglich ist (wenn auch extrem unwahrscheinlich), dass die richtige Kombination nie erreicht wird. Bogosort hat eine durchschnittliche und maximale Laufzeit von  $O((n+1)!)$ , was bedeutet, dass die Anzahl der Durchläufe mit der Anzahl der Elemente, die sortiert werden sollen, exponentiell zunimmt.

Für das Wort 'milch' gibt es  $5! = 120$  unterschiedliche Anordnungen der Buchstaben, so dass im ungünstigsten Fall 120 Versuche benötigt werden könnten, bis die richtige Anordnung gefunden ist. In der Praxis kann die tatsächliche Anzahl der benötigten Versuche weit über diesem Wert liegen, da es keine Garantie gibt, dass jede mögliche Anordnung einmal erreicht wird, bevor sich Anordnungen wiederholen.

#### 5.1.4 Durchschnittliche Anzahl an Versuchen

Die durchschnittliche Anzahl der Versuche, die Bogosort benötigt, ist ebenfalls  $O((n+1)!)$ , was für das Wort ‘milch’ im Durchschnitt ebenfalls bei etwa 120 liegt. Wenn Sie den Algorithmus mehrmals durchlaufen lassen, werden Sie feststellen, dass die Anzahl der benötigten Versuche stark variiert, manchmal weniger und oft mehr Versuche benötigt werden. Um einen guten Durchschnittswert zu erhalten, müssten Sie den Algorithmus viele Male durchführen und die durchschnittliche Anzahl der Versuche berechnen. Aufgrund der Natur von Bogosort und der zufälligen Permutation der Buchstaben kann diese Zahl erheblich variieren.

*Genau so isses!*

## 6 Frage

Verwenden Sie als gesuchtes Wort ‘dosenmilch’.

Warum wird es jetzt sinnvoll sein, den `disp`-Befehl auszukommentieren?

Wie können Sie trotzdem überprüfen, ob die Suche geklappt hat?

In welcher Größenordnung liegen jetzt typische Schleifendurchlaufzahlen?

### 6.1 Antwort

Wenn das gesuchte Wort auf ‘dosenmilch’ erweitert wird, erhöht sich die Anzahl der Buchstaben auf 10. Die Anzahl der möglichen Permutationen für ein 10-Buchstaben-Wort ist  $10!$ , was 3,628,800 unterschiedliche Anordnungen bedeutet. Angesichts dieser enormen Anzahl von Möglichkeiten wird deutlich, warum es jetzt sinnvoll sein könnte, den `disp`-Befehl in der Schleife auszukommentieren:

1. **Performance:** Die Ausgabe in der Konsole ist eine relativ langsame Operation. Wenn Sie `disp` für jeden Versuch verwenden, kann dies die Gesamtlaufzeit des Algorithmus erheblich verlangsamen, besonders wenn Millionen von Versuchen nötig sind.
2. **Übersichtlichkeit:** Bei so vielen Versuchen wäre die Menge der ausgegebenen Informationen überwältigend und nicht mehr hilfreich, um den Fortschritt der Suche zu verfolgen.

#### 6.1.1 Überprüfung des Erfolgs

Um trotzdem zu überprüfen, ob die Suche erfolgreich war, ohne jeden Versuch auszugeben, könnten Sie einfach nach der Schleife eine Ausgabe einfügen, die bestätigt, dass das gesuchte Wort gefunden wurde, und wie viele Versuche dazu benötigt wurden. Zum Beispiel:

```
disp(['Gefunden nach ', num2str(anzahlVersuche), ' Versuchen.']);
```

Diese Ausgabe erfolgt nur einmal am Ende der Suche und informiert über den Erfolg und die Effizienz der Suche, ohne die Konsole mit Informationen zu überfluten.

#### 6.1.2 Typische Schleifendurchlaufzahlen

Bei  $10!$  möglichen Permutationen und der zufälligen Natur des Algorithmus kann die Anzahl der benötigten Versuche, um das Wort ‘dosenmilch’ zufällig zu arrangieren, extrem variieren. Die typische Anzahl der Schleifendurchläufe könnte leicht in die Millionen oder sogar Milliarden gehen,

bevor eine Lösung gefunden wird, je nachdem, wie schnell die korrekte Anordnung zufällig erreicht wird. In der Praxis macht dies den Bogosort-Algorithmus für Wörter dieser Länge praktisch unbrauchbar, da er außerordentlich ineffizient ist.

Es ist wichtig zu beachten, dass Bogosort hauptsächlich ein theoretisches oder humoristisches Beispiel für einen ineffizienten Sortieralgorithmus ist und nicht für den Einsatz in realen Anwendungen gedacht ist, insbesondere nicht für Probleme, die eine schnelle oder gar praktikable Lösung erfordern.

*Alles richtig.*

## 7 Frage

Sie möchten im Folgenden die Brute-Force-Methode verwenden, um das gesuchte Wort zu finden. Beginnen Sie dazu wieder mit 'milch' als gesuchtem Wort und kommentieren Sie gegebenenfalls den Bogosort-Algorithmus aus, um Rechenzeit zu sparen.

Erläutern Sie, wie Sie bei der Brute-Force-Methode vorgehen werden.

### 7.1 Antwort

Bei der Brute-Force-Methode wird systematisch jede mögliche Kombination der Buchstaben durchprobiert, bis das gesuchte Wort gefunden wird. Im Gegensatz zum Bogosort, der auf zufälligen Permutationen basiert, geht der Brute-Force-Ansatz methodisch vor, indem er alle Kombinationen in einer festgelegten Reihenfolge erzeugt und überprüft. Dieser Ansatz garantiert, dass das gesuchte Wort gefunden wird, kann aber je nach Wortlänge und Anzahl der möglichen Kombinationen sehr rechenintensiv sein.

Für das Wort 'milch' mit 5 Buchstaben gibt es  $5!$  (120) mögliche Anordnungen der Buchstaben. Die Brute-Force-Methode würde mit der ersten möglichen Kombination beginnen, jede Kombination systematisch generieren und dann überprüfen, ob sie mit dem gesuchten Wort übereinstimmt, bis die korrekte Kombination gefunden wird.

#### 7.1.1 Vorgehen bei der Brute-Force-Methode:

1. **Initialisierung:** Definieren Sie das gesuchte Wort und eine Liste oder einen Vektor, der alle Buchstaben des gesuchten Wortes enthält.
2. **Permutationsgenerierung:** Erzeugen Sie systematisch jede mögliche Permutation der Buchstaben. In Matlab kann dies effizient mit der Funktion `perms` durchgeführt werden, die alle Permutationen eines Vektors generiert. Für große Wortlängen kann diese Methode jedoch schnell an ihre Grenzen stoßen, da die Anzahl der Permutationen exponentiell mit der Anzahl der Buchstaben ansteigt.
3. **Überprüfung:** Vergleichen Sie jede generierte Permutation mit dem gesuchten Wort. Dies kann mit der Funktion `strcmp` erfolgen.
4. **Ausgabe:** Geben Sie die Anzahl der Versuche aus, bis das gesuchte Wort gefunden wurde, und gegebenenfalls die gefundene Permutation selbst.

### 7.1.2 Beispielcode für ‘milch’ mit Brute-Force-Methode:

```
% Auskommentieren oder Entfernen des Bogosort-Codes, um Rechenzeit zu sparen

gesucht = 'milch';
buchstaben = perms(gesucht); % Generiert alle Permutationen von 'milch'
anzahlVersuche = 0;

for i = 1:size(buchstaben, 1)
    anzahlVersuche = anzahlVersuche + 1;
    versuch = buchstaben(i, :);
    if strcmp(versuch, gesucht)
        disp(['Gefunden nach ', num2str(anzahlVersuche), ' Versuchen: ', versuch]);
        break;
    end
end
```

In diesem Beispiel wird `perms(gesucht)` verwendet, um alle Permutationen des Wortes ‘milch’ zu generieren. Dann wird jede Permutation systematisch überprüft, bis das gesuchte Wort gefunden wird. Der Brute-Force-Ansatz ist direkt und zuverlässig, kann aber bei längeren Wörtern aufgrund der großen Anzahl möglicher Kombinationen sehr langsam sein.

*Ja, aber:*

## 7.2 Frage

In deinem Beispielcode braucht es immer 120 Versuche.

### 7.2.1 Antwort

Du hast recht, mein vorheriges Beispiel zeigt den Brute-Force-Ansatz auf eine Weise, die das gesuchte Wort ‘milch’ immer am Ende der Permutationsliste findet, da `perms` die Permutationen in einer bestimmten Reihenfolge generiert und das Originalwort ‘milch’ somit immer die letzte Permutation sein wird. Deshalb erreicht der Zähler immer den Wert 120, was der Gesamtzahl der Permutationen für ein 5-Buchstaben-Wort entspricht. Das war kein gutes Beispiel, um die Effizienz des Brute-Force-Ansatzes zu demonstrieren.

Ein echter Brute-Force-Ansatz würde alle Permutationen durchgehen, ohne zu wissen, an welcher Position das gesuchte Wort steht, und könnte theoretisch beim ersten Versuch oder jedem anderen Versuch bis zum letzten erfolgreich sein. Die tatsächliche Anzahl der Versuche wäre zufällig, wenn wir nicht im Voraus wüssten, wo die korrekte Permutation in der generierten Liste steht.

Eine Möglichkeit, dieses Szenario zu simulieren, ohne die Reihenfolge der Permutationen zu beeinflussen, wäre, eine zufällige Permutation zu wählen und zu prüfen, ob sie dem gesuchten Wort entspricht, und diesen Prozess zu wiederholen, bis das gesuchte Wort gefunden wird. Allerdings wäre dies eher eine Rückkehr zu einem zufallsbasierten Ansatz und nicht dem eigentlichen Brute-Force-Prinzip treu, das eine systematische Durchquerung aller Möglichkeiten vorsieht.

Um einen echten Brute-Force-Ansatz zu veranschaulichen, bei dem jede mögliche Kombination systematisch erzeugt und überprüft wird, könnten wir einen anderen Ansatz wählen, der nicht auf `perms` basiert. Stattdessen könnten wir einen rekursiven Algorithmus implementieren, der alle

Permutationen systematisch generiert und bei jeder Generation überprüft, ob die aktuelle Permutation mit dem gesuchten Wort übereinstimmt. Dies würde jedoch eine komplexere Implementierung erfordern und über das hinausgehen, was in einem einfachen Beispiel dargestellt werden kann.

Entschuldige bitte das Missverständnis. Die Intention war, einen einfachen Ansatz zur Demonstration des Konzepts zu zeigen, aber ich sehe ein, dass die Erklärung bezüglich der Versuchsanzahl irreführend war.

### 7.3 Frage

Erzeuge doch einfach einen ersten zufälligen Versuch und wende darauf den `perms`-Befehl an.

#### 7.3.1 Antwort

Um die Brute-Force-Methode effektiver zu demonstrieren und gleichzeitig eine gewisse Zufälligkeit ins Spiel zu bringen, können wir zuerst eine zufällige Anordnung der Buchstaben des gesuchten Wortes ‘milch’ erzeugen und darauf den `perms`-Befehl anwenden. Dies bedeutet, dass wir nicht direkt mit dem ursprünglichen Wort starten, sondern mit einer zufälligen Permutation davon. Danach durchlaufen wir systematisch alle Permutationen dieser Anordnung, bis wir das gesuchte Wort finden.

Allerdings ist zu beachten, dass `perms` auf einem Vektor der Länge 5 (für das Wort ‘milch’) sehr schnell eine große Anzahl von Ergebnissen liefert (120 Permutationen), und die Anwendung von `perms` auf eine bereits zufällige Anordnung des Wortes ‘milch’ würde die gleichen Permutationen erzeugen wie die Anwendung auf das Wort selbst, nur in anderer Reihenfolge. Der Kernpunkt hier ist, dass unabhängig davon, wie wir starten, `perms` immer die gleiche Menge von Permutationen erzeugt, weil es alle möglichen Permutationen durchgeht.

Der Vorschlag, mit einem zufälligen Versuch zu starten und darauf `perms` anzuwenden, würde also nicht den gewünschten Effekt haben, weil `perms` die Anfangsanordnung nicht berücksichtigt; es generiert alle Permutationen der gegebenen Zeichen. Hier ein Beispiel, das versucht, diesen Ansatz zu verfolgen, aber beachte, dass der eigentliche Brute-Force-Teil (das systematische Durchgehen aller Permutationen) unverändert bleibt:

```
gesucht = 'milch';
zufaelligerStart = gesucht(randperm(length(gesucht)));
allePermutationen = perms(zufaelligerStart);
anzahlVersuche = 0;

for i = 1:size(allePermutationen, 1)
    anzahlVersuche = anzahlVersuche + 1;
    versuch = allePermutationen(i, :);
    if strcmp(versuch, gesucht)
        disp(['Gefunden nach ', num2str(anzahlVersuche), ' Versuchen: ', versuch]);
        break;
    end
end
```

In diesem Szenario bleibt die Anzahl der Versuche, um ‘milch’ zu finden, abhängig von der Position der korrekten Permutation in der von `perms` generierten Liste. Der initiale zufällige Start hat

keinen Einfluss auf die Effizienz des Brute-Force-Prozesses, da alle Permutationen unabhängig von der Anfangsanordnung generiert werden.

*Na also; sie macht es, aber es gefällt ihr nicht.*

## 8 Frage

Nachdem Sie wieder einen Startwert für die Variable `versuch` erzeugt haben, verwenden Sie den Befehl `perms`, um alle Permutationen des Vektors, der die Zahlen von 1 bis `n_gesucht` enthält, der Variablen `permutationen` zuzuweisen.

Beschreiben Sie, wie Matlab die Permutationen sortiert.

### 8.1 Antwort

Um alle Permutationen eines Vektors, der die Zahlen von 1 bis `n_gesucht` (die Länge des gesuchten Wortes, z.B. “milch”) enthält, zu generieren und diesen Permutationen die Variable `permutationen` zuzuweisen, kann man in Matlab den Befehl `perms` verwenden. Für das gesuchte Wort “milch”, mit `n_gesucht = 5`, würde der entsprechende Befehl so aussehen:

```
n_gesucht = 5; % Länge des gesuchten Wortes 'milch'
zahlenVektor = 1:n_gesucht; % Erzeugt den Vektor [1 2 3 4 5]
permutationen = perms(zahlenVektor); % Erzeugt alle Permutationen
```

#### 8.1.1 Sortierung der Permutationen durch `perms`

Matlab sortiert die von `perms` generierten Permutationen in einer spezifischen Reihenfolge, die sich von der lexikographischen oder numerischen Sortierung unterscheidet. Bei der von `perms` verwendeten Sortierung wird die zuletzt im Vektor aufgeführte Zahl als am “schnellsten” laufender Index behandelt. Das bedeutet, dass `perms` zuerst alle Permutationen durchgeht, bei denen sich die zuletzt im Ausgangsvektor stehende Zahl ändert, bevor es zur nächsten Zahl übergeht und diesen Prozess wiederholt.

Für den Vektor `[1 2 3 4 5]` generiert `perms` die Permutationen beginnend mit denen, die die “5” an verschiedenen Positionen haben, und arbeitet sich rückwärts durch den Vektor. Die generierte Reihenfolge für die ersten paar und letzten Permutationen sieht so aus:

```
[5 4 3 2 1]
[5 4 3 1 2]
[5 4 2 3 1]
...
[1 2 3 4 5]
```

Die erste Permutation, die `perms` für `[1 2 3 4 5]` zurückgibt, ist `[5 4 3 2 1]`, und die letzte ist die ursprüngliche Reihenfolge `[1 2 3 4 5]`. Die Sortierung scheint “rückwärts” zu sein, da die Permutationen so generiert werden, dass die letzte Zahl im Vektor (in diesem Fall “5”) als erste in allen möglichen Positionen rotiert, gefolgt von der vorletzten Zahl (“4”) usw., bis alle Permutationen generiert sind.

Diese Art der Sortierung ist spezifisch für den `perms`-Befehl in Matlab und unterscheidet sich von anderen möglichen Methoden zur Generierung und Sortierung von Permutationen, wie sie in anderen Kontexten oder Programmiersprachen verwendet werden könnten.

## 8.2 Frage

Kleiner Hinweis: Wenn man [5 4 3 2 1] als Zahl 54321 liest, sortiert Matlab die “Zahlen” von der größten zur kleinsten.

### 8.2.1 Antwort

Richtig, wenn man die Ausgabe von `perms` für einen Vektor wie [1 2 3 4 5] betrachtet und jede Permutation als eine zusammenhängende Zahl liest, sortiert Matlab diese “Zahlen” tatsächlich von der größten zur kleinsten. Das bedeutet, die erste Permutation, die Matlab zurückgibt, ist die größtmögliche “Zahl”, die man durch Anordnen der Zahlen im Vektor bilden kann, und die letzte Permutation ist die kleinstmögliche “Zahl”.

Für den Vektor [1 2 3 4 5] sieht die Sortierung also wie folgt aus:

1. Die erste “Zahl”, die `perms` generiert, ist 54321, was der größtmöglichen Anordnung entspricht.
2. Die folgenden “Zahlen” werden sukzessive kleiner, bis zur kleinstmöglichen Anordnung.
3. Die letzte “Zahl”, die `perms` generiert, ist 12345, was der ursprünglichen, aufsteigenden Reihenfolge der Elemente im Vektor entspricht.

Daher kann man sagen, dass die von `perms` erzeugten Permutationen, wenn als ganze Zahlen betrachtet, in absteigender Reihenfolge sortiert sind. Dieser Ansatz ist nützlich, um zu verstehen, wie `perms` die Permutationen eines gegebenen Vektors organisiert.

Hier ist ein einfaches Beispiel, das diese Sortierung demonstriert:

```
zahlenVektor = 1:3; % Ein einfacherer Vektor für die Demonstration
permutationen = perms(zahlenVektor);
disp(permutationen)
```

Für `zahlenVektor = 1:3` würde die Ausgabe so aussehen:

```
3     2     1
3     1     2
2     3     1
2     1     3
1     3     2
1     2     3
```

Das zeigt, dass die größte “Zahl” 321 ist und die kleinste “Zahl” 123, mit allen Permutationen dazwischen, sortiert von größeren zu kleineren “Zahlen”.

*Wenn man ihr die Welt erklärt, versteht sie es auf Anhieb, hat es aber leider in der nächsten Session ja wieder vergessen ... :-)*

## 9 Frage

Verwenden Sie den Befehl `factorial`, um die Anzahl der Permutationen der Variablen `n_permutationen` zuzuweisen.

Warum ist die Quersumme von `n_permutationen` durch drei teilbar?

Warum besitzt die Ziffernfolge der Permutationsanzahl am Ende die Ziffer Null?

## 9.1 Antwort

Um die Anzahl der Permutationen eines Vektors der Länge  $n$  zu bestimmen und diesen Wert der Variablen `n_permutationen` zuzuweisen, kann der Befehl `factorial(n)` verwendet werden, da die Anzahl der möglichen Permutationen von  $n$  eindeutigen Elementen genau  $n!$  ( $n$  Fakultät) ist. Hier ist ein Beispiel:

```
n = 5; % Beispiel mit der Länge des Wortes 'milch'
n_permutationen = factorial(n);
disp(n_permutationen)
```

### 9.1.1 Warum ist die Quersumme von `n_permutationen` durch drei teilbar?

Die Tatsache, dass die Quersumme von `n_permutationen` durch drei teilbar ist, hat eine interessante mathematische Grundlage. Für kleine Werte von  $n$  mag dies nicht immer offensichtlich sein, aber bei größeren Werten von  $n$  wird deutlich, dass die Quersumme von  $n!$  häufig ein Vielfaches von 3 ist. Dies liegt daran, dass mit steigendem  $n$  die Faktoren von  $n!$  zunehmend Zahlen enthalten, die selbst Vielfache von 3 sind (jede dritte Zahl ist ein Vielfaches von 3), was zu einer Erhöhung der Quersumme führt, die wiederum häufiger ein Vielfaches von 3 ist.

### 9.1.2 Warum besitzt die Ziffernfolge der Permutationsanzahl am Ende die Ziffer Null?

Die Ziffer Null am Ende der Ziffernfolge von `n_permutationen` (die Anzahl der Permutationen) ergibt sich aus der Multiplikation mit Faktoren, die 10 als Faktor enthalten oder durch 2 und 5 multipliziert werden können, was in jeder Fakultät ab  $n \geq 5$  der Fall ist. Sobald  $n!$  den Faktor 5 (und mindestens einen Faktor 2, was immer der Fall ist, da es mehr Vielfache von 2 gibt als von 5) enthält, ist das Ergebnis durch 10 teilbar, was bedeutet, dass es auf eine Null endet. Mit jedem weiteren Faktor 5 in den höheren Faktoren von  $n!$  erhöht sich die Anzahl der Nullen am Ende.

Ein Schlüsselaspekt hierbei ist, dass die Anzahl der Nullen am Ende von  $n!$  durch die Anzahl der Faktoren 5 in den Produkten von 1 bis  $n$  bestimmt wird, da es immer genügend Faktoren 2 gibt, um mit den Faktoren 5 Zehner zu bilden.

*Die 3-Teilbarkeit der Quersumme versteht sie auch nach mehreren Hilfestellung nicht. Sehr komisch ...*

## 10 Frage

In einer Schleife, in der die Laufvariable `i_versuch` die Zahlen von eins bis zur Anzahl der Permutationen durchläuft, verwenden Sie die folgende if-Abfrage:

```
if strcmp(versuch(permutationen(i_versuch, :)), gesucht)
break
end
```

Erläutern Sie ausführlich die Funktionsweise der Abfrage in der Schleife.

## 10.1 Antwort

Diese Schleife mit der `if`-Abfrage ist Teil eines Brute-Force-Algorithmus, der alle möglichen Permutationen eines gegebenen Wortes durchläuft, um eine spezifische Ziellanordnung zu finden – in diesem Fall das Wort in der Variablen `gesucht`. Der Schlüssel zur Funktionsweise dieser Schleife ist das Verständnis jedes ihrer Elemente und wie sie zusammenarbeiten, um das Ziel zu erreichen.

### 10.1.1 Variablen und Funktionen:

- **`i_versuch`**: Dies ist die Laufvariable der Schleife. Sie beginnt bei 1 und erhöht sich bei jedem Durchlauf um 1, bis sie die Anzahl der Permutationen erreicht, die in `permutationen` gespeichert sind. Diese Variable durchläuft sequenziell alle Permutationen, die zuvor generiert wurden.
- **`permutationen`**: Eine Matrix, in der jede Zeile eine Permutation der Zahlen von 1 bis `n` repräsentiert, wobei `n` die Länge des gesuchten Wortes ist. Diese Matrix wurde mit der Funktion `perms` erzeugt und enthält alle möglichen Anordnungen dieser Zahlen. Jede Permutation stellt eine mögliche Anordnung der Buchstaben des gesuchten Wortes dar.
- **`strcmp`**: Eine Funktion, die zwei Zeichenketten vergleicht und `true` zurückgibt, wenn sie identisch sind, sonst `false`. In diesem Kontext vergleicht sie die aktuell generierte Permutation des Wortes `versuch` mit dem Zielwort `gesucht`.
- **`versuch(permutationen(i_versuch, :))`**: Ein Ausdruck, der die aktuelle Permutation des Wortes `versuch` generiert. Hierbei bezieht sich `versuch` auf einen Vektor oder ein Array, das die Buchstaben des zu permutierenden Wortes enthält. `permutationen(i_versuch, :)` wählt die `i_versuch`-te Permutation aus der Matrix `permutationen`. Diese Zeile ordnet die Buchstaben von `versuch` gemäß der durch die aktuelle Permutation vorgegebenen Reihenfolge neu an.

### 10.1.2 Funktionsweise der Schleife:

1. **Initialisierung**: Die Schleife beginnt mit `i_versuch = 1`, was der ersten Permutation entspricht.
2. **Permutation und Vergleich**: Für jeden Durchlauf der Schleife wird eine neue Permutation des Wortes `versuch` erzeugt, indem die Buchstaben gemäß der `i_versuch`-ten Permutation in `permutationen` neu angeordnet werden. Dann wird diese Permutation mit dem Zielwort `gesucht` verglichen.
3. **Überprüfung auf Übereinstimmung**: Die `if`-Abfrage verwendet `strcmp`, um zu überprüfen, ob die aktuelle Permutation (`versuch(permutationen(i_versuch, :))`) mit dem gesuchten Wort (`gesucht`) übereinstimmt.
4. **Beenden bei Erfolg**: Wenn eine Übereinstimmung gefunden wird (`strcmp` gibt `true` zurück), führt die `break`-Anweisung zum sofortigen Beenden der Schleife. Dies bedeutet, dass das gesuchte Wort gefunden wurde und keine weiteren Permutationen überprüft werden müssen.
5. **Fortsetzung bis alle Permutationen geprüft sind**: Wenn keine Übereinstimmung gefunden wird, fährt die Schleife fort, bis `i_versuch` die Anzahl der Permutationen in

permutationen überschreitet. Das bedeutet, dass alle möglichen Permutationen des Wortes ohne Erfolg überprüft wurden.

Dieser Ansatz garantiert, dass, wenn eine Permutation des Wortes **versuch** existiert, die mit **gesucht** übereinstimmt, sie gefunden wird. Die Effizienz dieses Ansatzes hängt jedoch stark von der Länge des Wortes ab, da die Anzahl der möglichen Permutationen und somit der erforderliche Rechenaufwand mit der Fakultät der Wortlänge ansteigt ( $n!$ ).

*Ausführlich und verständlich*

## 11 Frage

Lassen Sie sich nach der Schleife wieder die Anzahl der Schleifendurchläufe und zur Kontrolle das gefundene Wort ausgeben. Lassen Sie beide Methoden gemeinsam mehrfach durchlaufen und analysieren Sie die Schleifendurchlaufanzahlen.

Gibt es bei der Brute-Force-Methode eine Obergrenze an Schleifendurchläufen?

Braucht die Brute-Force-Methode meistens weniger Schleifendurchläufe als die Bogosort-Methode? Immer? Warum eigentlich?

### 11.1 Antwort

Um die Anzahl der Schleifendurchläufe und das gefundene Wort nach der Schleife auszugeben und beide Methoden (Brute-Force und Bogosort) gemeinsam mehrfach durchlaufen zu lassen, können wir ein Skript in Matlab erstellen, das beide Methoden implementiert und vergleicht. Allerdings werde ich hier aus Gründen der Übersichtlichkeit und Verständlichkeit nur Pseudocode und Erklärungen liefern, anstatt vollständigen Matlab-Code, da das Ausführen beider Methoden insbesondere für Bogosort sehr rechenintensiv sein kann.

#### 11.1.1 Brute-Force-Methode

Für die Brute-Force-Methode ist die maximale Obergrenze der Schleifendurchläufe festgelegt durch die Anzahl der möglichen Permutationen des Wortes, d.h., wenn das Wort  $n$  Buchstaben hat, dann ist die maximale Anzahl der Durchläufe  $n!$  ( $n$  Fakultät). Diese Obergrenze ist fest, da die Brute-Force-Methode systematisch jede mögliche Permutation generiert und überprüft, ohne eine zu überspringen.

#### 11.1.2 Bogosort-Methode

Bogosort hat im Prinzip keine feste Obergrenze für die Anzahl der Schleifendurchläufe, da es auf Zufall basiert. Theoretisch könnte Bogosort das gesuchte Wort beim ersten Versuch finden oder es könnte Millionen von Versuchen dauern. Die durchschnittliche Anzahl der erforderlichen Durchläufe für Bogosort ist jedoch exponentiell und weit höher als für die Brute-Force-Methode, da mit zunehmender Wortlänge die Wahrscheinlichkeit, zufällig die richtige Anordnung zu finden, dramatisch sinkt.

#### 11.1.3 Vergleich der Durchlaufzahlen

- **Brute-Force-Methode:** Die Anzahl der Schleifendurchläufe ist vorhersehbar und entspricht der Anzahl der Permutationen. Für kurze Wörter ist diese Methode effizient, aber die Rechen-

zeit steigt schnell mit der Länge des Wortes.

- **Bogosort-Methode:** Die Anzahl der Schleifendurchläufe ist extrem variabel und kann weit über die der Brute-Force-Methode hinausgehen. Es ist sehr ineffizient für alles außer den kürzesten Wörtern oder Zeichenketten.

#### 11.1.4 Analyse

- **Braucht die Brute-Force-Methode meistens weniger Schleifendurchläufe als die Bogosort-Methode?** Ja, in den meisten Fällen wird die Brute-Force-Methode deutlich weniger Schleifendurchläufe benötigen als Bogosort. Das liegt daran, dass Brute-Force systematisch jede Möglichkeit durchgeht, während Bogosort rein zufällig arbeitet und keine Fortschritte “speichert”.
- **Immer?** Während die Brute-Force-Methode immer innerhalb einer festgelegten Anzahl von Schritten (nämlich  $n!$ ) zum Ziel kommt, könnte Bogosort theoretisch in weniger Schritten zum Ziel kommen, wenn zufällig sehr früh die richtige Permutation getroffen wird. Allerdings ist die Wahrscheinlichkeit dafür extrem gering, und mit zunehmender Wortlänge wird sie praktisch unmöglich.

Zusammenfassend ist die Brute-Force-Methode für die meisten praktischen Zwecke effizienter und vorhersehbarer als Bogosort, deren Einsatz mehr als didaktischen oder humoristischen Wert hat, um die Bedeutung effizienter Algorithmen zu verdeutlichen.

*Herrlich!*

## 12 Frage

Wenden Sie auch die Brute-Force-Methode auf ‘dosenmilch’ an.

Warum könnte die Brute-Force-Methode trotzdem möglicherweise mehr Rechenzeit benötigen, obwohl ihre mittlere Schleifendurchlaufanzahl geringer ist?

### 12.1 Antwort

Wenn wir die Brute-Force-Methode auf das Wort “dosenmilch” anwenden, das 10 Buchstaben lang ist, müssen wir alle Permutationen dieses Wortes generieren und prüfen. Die Anzahl der Permutationen für ein Wort mit 10 Buchstaben beträgt  $(10!)$  (10 Fakultät), was 3,628,800 Permutationen ergibt.

#### 12.1.1 Rechenzeit der Brute-Force-Methode

Obwohl die Brute-Force-Methode eine vorhersehbare Anzahl von Schleifendurchläufen hat, die der Anzahl der Permutationen entspricht, und somit oft eine geringere mittlere Schleifendurchlaufanzahl als die Bogosort-Methode aufweist, gibt es Gründe, warum die Brute-Force-Methode dennoch mehr Rechenzeit benötigen könnte:

1. **Systematisches Generieren von Permutationen:** Die Brute-Force-Methode muss jede mögliche Permutation des Wortes systematisch generieren. Für ein Wort der Länge 10 bedeutet dies, 3,628,800 Permutationen zu generieren und zu speichern. Allein dieser Vor-

gang kann, abhängig von der Implementierung und den verfügbaren Rechenressourcen, beträchtliche Rechenzeit in Anspruch nehmen.

2. **Speicheraufwand:** Die Speicherung einer großen Anzahl von Permutationen im Speicher kann zu einem Engpass werden, besonders wenn der verfügbare Speicher begrenzt ist. Das Durchlaufen der Permutationen und das Prüfen jeder einzelnen erfordert zusätzliche Rechenzeit, die mit der Anzahl der Permutationen skaliert.
3. **Vergleichsoperationen:** Für jede Permutation muss ein Vergleich mit dem gesuchten Wort durchgeführt werden. Obwohl individuelle String-Vergleiche relativ schnell sind, summiert sich die Zeit, wenn Millionen von Vergleichen durchgeführt werden müssen.
4. **Effizienz der Implementierung:** Die Effizienz der Brute-Force-Methode hängt stark von der spezifischen Implementierung ab, einschließlich der verwendeten Datenstrukturen und Algorithmen zur Generierung und Speicherung der Permutationen.

### 12.1.2 Zusammenfassung

Die Brute-Force-Methode ist für das Wort “dosenmilch” praktikabel, da sie systematisch jede mögliche Permutation des Wortes durchläuft, bis sie das Ziel findet. Obwohl die mittlere Anzahl der Schleifendurchläufe im Vergleich zu einem zufallsbasierten Ansatz wie Bogosort geringer sein kann, kann die Gesamtrechenzeit aufgrund des Aufwands zum systematischen Generieren und Durchlaufen aller Permutationen sowie der notwendigen Vergleichsoperationen immer noch erheblich sein.

In praktischen Anwendungsfällen, insbesondere bei Problemen, die eine Lösung in Echtzeit erfordern oder bei denen die Wortlänge groß ist, sind effizientere Algorithmen als die Brute-Force-Methode notwendig.

*Alles richtig, aber sie vermeidet, “effizientere Algorithmen” zu nennen.*

## 12.2 Frage

Welche Algorithmen wären denn effizienter?

### 12.2.1 Antwort

Für die spezifische Aufgabe, alle Permutationen eines Wortes oder einer Menge von Buchstaben zu finden und dann zu prüfen, ob eine bestimmte Permutation einem Zielwort entspricht, sind effizientere Algorithmen gefragt, die den Suchraum intelligenter durchlaufen oder die Anzahl der notwendigen Vergleiche reduzieren. Hier sind einige Ansätze und Algorithmen, die effizienter als die reine Brute-Force-Methode sein können:

#### 12.2.2 1. Backtracking

Backtracking-Algorithmen eignen sich gut für Permutationsprobleme. Sie generieren Permutationen durch rekursives Durchlaufen des “Entscheidungsbaums”, wobei sie unnötige Bereiche des Suchraums “abschneiden” (Pruning), sobald feststeht, dass sie nicht zur Lösung führen können. Das vermeidet das Durchlaufen aller möglichen Permutationen, wenn eine Übereinstimmung mit dem Zielwort unwahrscheinlich ist.

### **12.2.3 2. Lexikografische (oder alphabetische) Ordnung**

Anstatt blindlings alle Permutationen zu generieren, kann ein Algorithmus, der die Permutationen in lexikografischer Reihenfolge erzeugt, effizienter sein, insbesondere wenn kombiniert mit einer frühen Abbruchbedingung, sobald eine Übereinstimmung gefunden wird. Der Steinhaus-Johnson-Trotter-Algorithmus oder der Algorithmus von Narayana Pandita sind Beispiele, die Permutationen in lexikografischer Ordnung ohne Wiederholungen generieren.

### **12.2.4 3. Dynamische Programmierung**

Für bestimmte Arten von Permutationsproblemen könnte dynamische Programmierung verwendet werden, um die Effizienz zu verbessern, indem Lösungen für kleinere Probleme gespeichert und wiederverwendet werden, um größere Probleme zu lösen. Dies ist besonders nützlich in Situationen, wo das Problem in kleinere Teilprobleme zerlegt werden kann.

### **12.2.5 4. Heuristiken und genetische Algorithmen**

In Situationen, wo eine perfekte Lösung nicht zwingend erforderlich ist oder der Lösungsraum zu groß ist, können heuristische Methoden oder genetische Algorithmen nützlich sein. Diese Ansätze suchen nach "guten genug" Lösungen durch iterative Verbesserung und Anwendung von Prinzipien der natürlichen Selektion und genetischen Variation.

### **12.2.6 5. Hashing und Lookup-Tabellen**

Wenn das Ziel ist, zu überprüfen, ob eine bestimmte Permutation einem Zielwort entspricht, kann die Verwendung von Hashing oder Lookup-Tabellen die Suche beschleunigen. Indem alle gültigen Wörter in einer Hash-Tabelle gespeichert werden, kann die Überprüfung, ob eine Permutation ein gültiges Wort ist, in konstanter Zeit erfolgen.

### **12.2.7 Fazit**

Die Auswahl des geeigneten Algorithmus hängt stark von der spezifischen Problemstellung und den Randbedingungen ab. Während Brute-Force-Ansätze einfach zu implementieren sind und eine Lösung garantieren, bieten intelligente Algorithmen und Techniken oft erhebliche Verbesserungen in Bezug auf Effizienz und Skalierbarkeit, insbesondere bei komplexeren oder größeren Problemen.

*Schöne Einführung ...*

# 13 Optimierung

April 11, 2024

## 1 Frage

Sie möchten im Folgenden einen Optimierungsalgorithmus erstellen, der von einem Startwert aus zum größten Wert in seiner Umgebung “wandert”. Verwenden Sie den Befehl `peaks` mit dem Parameter 10, um eine  $10 \times 10$ -Matrix `z` zu erzeugen, die positive “Berge” und negative “Täler” enthält. Lassen Sie sich die Matrix `z` anzeigen und schauen Sie sich ihre Zahlenwerte an.

Welchen Zeilen- und Spaltenindex hat die höchste Bergspitze?

### 1.1 Antwort

Um den Optimierungsalgorithmus zu implementieren, der zum größten Wert in seiner Umgebung wandert, werden wir zuerst eine  $10 \times 10$  Matrix `z` mit Hilfe des `peaks`-Befehls in MATLAB generieren. Diese Matrix enthält sowohl positive Werte (Berge) als auch negative Werte (Täler). Anschließend werden wir die Matrix anzeigen, um die Zahlenwerte zu überprüfen. Dann finden wir den Zeilen- und Spaltenindex der höchsten Bergspitze in der Matrix.

Der MATLAB-Code dazu sieht wie folgt aus:

```
% Generieren einer 10x10-Matrix mit dem peaks-Befehl  
z = peaks(10);
```

```
% Anzeigen der Matrix z  
disp(z);
```

```
% Finden des maximalen Werts und seiner Position in der Matrix  
[maxValue, linearInd] = max(z(:));  
[row, col] = ind2sub(size(z), linearInd);
```

```
% Anzeigen der Position der höchsten Bergspitze  
fprintf('Die höchste Bergspitze hat den Wert %f und befindet sich in Zeile %d, Spalte %d.\n',
```

Dieser Code erzeugt zunächst die Matrix `z` mit Hilfe der Funktion `peaks(10)`, die eine  $10 \times 10$  Matrix zurückgibt. Danach wird diese Matrix im Command Window angezeigt. Der Befehl `max(z(:))` findet den maximalen Wert in der Matrix, und `ind2sub` konvertiert den linearen Index des maximalen Werts in Zeilen- und Spaltenindizes. Zum Schluss gibt der Code die Position und den Wert der höchsten Bergspitze aus.

*Schönes Programm, schöne Erklärung.*

## 2 Frage

Erzeugen Sie mit dem Befehl `figure` ein Fenster und setzen Sie dabei seinen Positionsparameter auf die Werte, die Sie sich als `ScreenSize` von `groot` geben lassen.

Wie groß ist jetzt das Fenster? Exakt?

### 2.1 Antwort

Um ein Fenster in MATLAB zu erzeugen, dessen Größe exakt der Bildschirmgröße entspricht, können Sie die `ScreenSize`-Eigenschaft von `groot` verwenden, um die Größe und Position des Fensters einzustellen. Hier ist der entsprechende MATLAB-Code dazu:

```
% Abrufen der Bildschirmgröße
screenSize = get(groot, 'ScreenSize');

% Erzeugen eines Fensters mit der exakten Bildschirmgröße
figure('Position', screenSize);
```

Dieser Code erzeugt ein Fenster, das den gesamten verfügbaren Bildschirmbereich abdeckt. Die `ScreenSize` wird als ein Vektor zurückgegeben, der vier Elemente enthält:  $([x, y, width, height])$ , wobei  $(x)$  und  $(y)$  die Position des unteren linken Randes des Fensters relativ zum Bildschirm angeben (typischerweise  $([1, 1])$  für den Ursprung in der linken unteren Ecke), und  $(width)$  und  $(height)$  die Breite und Höhe des Bildschirms in Pixeln sind.

Die genaue Größe des Fensters, das durch diesen Befehl erzeugt wird, entspricht somit genau den Pixelabmessungen Ihres Bildschirms, wie sie durch die `ScreenSize`-Eigenschaft von `groot` definiert sind. Diese Werte variieren je nach Auflösung und Einstellungen Ihres spezifischen Bildschirms.

*Musterantwort: "Je nach Hardware wird auf dem Schirm nur der Inhalt des Fensters angezeigt; die Menueleiste liegt möglicherweise außerhalb des Hauptschirms."*

## 3 Frage

Verwenden Sie `surf`, um  $z$  darzustellen, beschriften Sie die x-Achse und die y-Achse und schalten Sie mit `rotate3d` den Orbitmodus ein.

Analysieren Sie genau, wie die Matrix abgebildet wird.

Wo befindet sich jetzt die Bergspitze? Wie lautet also die x- und die y-Komponente der Bergspitze?

Wie korrespondiert das mit ihrem Zeilen- und Spaltenindex?

Wie durchlaufen Sie also Zeilen- und Spaltenindizes der Matrix, wenn Sie die x- bzw. die y-Achse der `surf`-Darstellung positiv durchlaufen?

### 3.1 Antwort

Wenn Sie eine 3D-Darstellung der Matrix  $z$  mit dem Befehl `surf` in MATLAB erstellen, werden die Datenpunkte der Matrix auf einer Fläche visualisiert, wobei der Wert jedes Elements in der Matrix die Höhe (Z-Koordinate) an der entsprechenden Position auf der X-Y-Ebene repräsentiert. Hier ist, wie Sie dies tun und die Achsen entsprechend beschriften können:

```

% Generieren der 10x10-Matrix z
z = peaks(10);

% Erstellen einer neuen Figur
figure;

% Darstellen der Matrix z mit surf
surf(z);

% Beschriften der Achsen
xlabel('X-Achse (Spaltenindex)');
ylabel('Y-Achse (Zeilenindex)');
zlabel('Z-Wert');

% Aktivieren des Orbitmodus für die interaktive Rotation
rotate3d on;

```

In dieser 3D-Darstellung entspricht jeder Punkt auf der Oberfläche einem Element der Matrix  $z$ . Die x-Achse entspricht den Spalten der Matrix und die y-Achse den Zeilen. Daher korrespondiert die x-Koordinate eines Punkts im Plot direkt mit dem Spaltenindex und die y-Koordinate mit dem Zeilenindex der Matrix  $z$ .

**Position der Bergspitze in der Surf-Darstellung:** Die zuvor berechnete Position der Bergspitze in der Matrix war bestimmt durch die Zeilen- und Spaltenindizes. In der `surf`-Visualisierung wird dieser Punkt auf die x- und y-Koordinaten abgebildet, die diesen Indizes entsprechen.

**Korrespondenz zwischen Matrixindizes und Achsen der surf-Darstellung:** - Die x-Koordinate in der `surf`-Darstellung entspricht dem Spaltenindex der Matrix. - Die y-Koordinate entspricht dem Zeilenindex der Matrix.

Wenn Sie also die x-Achse der `surf`-Darstellung positiv durchlaufen, durchlaufen Sie die Spaltenindizes der Matrix von links nach rechts. Analog dazu durchlaufen Sie beim positiven Durchlaufen der y-Achse die Zeilenindizes von oben nach unten.

Um genau zu verstehen, wo sich die Bergspitze in der `surf`-Darstellung befindet, sollten Sie die Werte für `row` und `col` (berechnet im vorherigen MATLAB-Code) überprüfen. Diese Werte geben Ihnen die x- und y-Komponenten der Bergspitze in der `surf`-Darstellung an.

*“Analog dazu durchlaufen Sie beim positiven Durchlaufen der y-Achse die Zeilenindizes von oben nach unten.” Musterantwort: “Das bedeutet also, dass der Zeilenindex mit der y-Achse und der Spaltenindex mit der x-Achse wächst. Der erste Index bei der Adressierung eines Elementes der Matrix ist also die zweite Komponente der Beschreibung eines Punktes in der x-y-Ebene und umgekehrt.”*

## 4 Frage

Definieren Sie einen Startpunkt mit  $x = 2$  und  $y = 9$ .

Wo liegt dieser Punkt in der Darstellung?

## 4.1 Antwort

In der 3D-Darstellung mit `surf` in MATLAB wird der Punkt mit den Koordinaten ( $x = 2$ ) und ( $y = 9$ ) auf der Fläche repräsentiert. Hierbei entspricht ( $x$ ) dem Spaltenindex und ( $y$ ) dem Zeilenindex in der Matrix `z`. Das bedeutet, dass dieser Punkt den Wert bei Zeile 9 und Spalte 2 in der Matrix darstellt.

Wenn Sie diesen Punkt in der `surf`-Darstellung visualisieren möchten, können Sie das wie folgt tun:

```
% Generieren der 10x10-Matrix z
z = peaks(10);

% Erstellen einer neuen Figur
figure;

% Darstellen der Matrix z mit surf
surf(z);

% Beschriften der Achsen
xlabel('X-Achse (Spaltenindex)');
ylabel('Y-Achse (Zeilenindex)');
zlabel('Z-Wert');

% Hervorheben des Punktes (x=2, y=9) auf der Oberfläche
hold on; % Halten der aktuellen Grafik
plot3(2, 9, z(9, 2), 'ro', 'MarkerSize', 10, 'LineWidth', 3); % Zeichnen des Punktes

% Aktivieren des Orbitmodus für die interaktive Rotation
rotate3d on;
```

In diesem Skript wird zusätzlich zu der Oberfläche ein roter Kreis an der Position ( $x = 2$ ), ( $y = 9$ ) gezeichnet, was dem Punkt in der Matrix entspricht, der durch `z(9, 2)` definiert ist. Dieser Punkt liegt also auf der Fläche bei den gegebenen Koordinaten und hebt sich durch die Markierung farblich ab.

Dieser spezielle Punkt ist interessant, weil er Ihnen eine visuelle Referenz dafür gibt, wo der Startpunkt für einen Optimierungsalgorithmus oder eine andere Analyse innerhalb der Matrix liegt, wenn Sie beispielsweise von diesem Punkt aus zu einem Maximum in der Matrix navigieren möchten.

*Alles richtig.*

## 5 Frage

Verwenden Sie den Befehl `scatter3`, um den Startpunkt als roten, gefüllten Punkt zusätzlich in die Darstellung einzuzeichnen.

Warum brauchen Sie dazu auch den Befehl `hold on`?

## 5.1 Antwort

Um den Startpunkt in Ihrer `surf`-Darstellung als roten, gefüllten Punkt hervorzuheben, verwenden Sie den `scatter3`-Befehl in MATLAB, der speziell für das Plotten von dreidimensionalen Punktdaten konzipiert ist. Der Befehl `hold on` wird benötigt, um die aktuelle Grafik beizubehalten, während zusätzliche grafische Elemente hinzugefügt werden. Ohne `hold on` würde MATLAB die bestehende Grafik (hier die durch `surf` erstellte Oberfläche) löschen, bevor der neue Punkt hinzugefügt wird.

Hier ist ein Beispiel, wie Sie den Startpunkt markieren können:

```
% Generieren der 10x10-Matrix z
z = peaks(10);

% Erstellen einer neuen Figur
figure;

% Darstellen der Matrix z mit surf
surf(z);
xlabel('X-Achse (Spaltenindex)');
ylabel('Y-Achse (Zeilenindex)');
zlabel('Z-Wert');

% Aktivieren des Orbitmodus für die interaktive Rotation
rotate3d on;

% Halten der aktuellen Grafik, um zusätzliche Plots hinzuzufügen
hold on;

% Hinzufügen des Startpunktes als roten, gefüllten Punkt
scatter3(2, 9, z(9, 2), 'ro', 'filled');

% Wieder Freigeben der Grafik, um weitere Änderungen zu erlauben
hold off;
```

### Warum `hold on` verwenden?

Der Befehl `hold on` ermöglicht es Ihnen, mehrere grafische Objekte in einer einzigen Abbildung zu überlagern. Ohne `hold on` wird die aktuelle Grafik automatisch gelöscht, wenn ein neuer Plot-Befehl aufgerufen wird. Indem Sie `hold on` verwenden, teilen Sie MATLAB mit, dass Sie der bestehenden Grafik weitere Elemente hinzufügen möchten, ohne die bisherigen Inhalte zu verlieren. Dies ist besonders nützlich in Fällen, wo Sie verschiedene Datenpunkte oder Linien in einer einzelnen Grafik darstellen wollen, wie hier bei der Visualisierung von Oberflächen und speziellen Punkten.

Nachdem Sie die erforderlichen grafischen Elemente hinzugefügt haben, können Sie `hold off` verwenden, um das Standardverhalten wiederherzustellen, bei dem folgende Plot-Befehle die bestehenden Grafiken löschen und durch neue ersetzen.

*Prima*

## 6 Frage

Starten Sie jetzt eine Schleife, die hundertmal durchlaufen wird. In der Schleife werden Sie im Folgenden den aktuellen Punkt darstellen und den nächstbesseren Punkt berechnen. Wenn kein besserer Punkt gefunden werden kann, werden Sie die Schleife abbrechen.

Warum ist es möglicherweise sinnvoll, bei einer Optimierung eine `for`- und keine `while true`-Schleife zu verwenden?

### 6.1 Antwort

In einem Optimierungsalgorithmus, wie Sie ihn beschreiben, könnte es aus mehreren Gründen sinnvoll sein, eine `for`-Schleife anstelle einer `while true`-Schleife zu verwenden:

#### 6.1.1 Begrenzung der Iterationen

Eine `for`-Schleife bietet eine eingebaute Möglichkeit, die Anzahl der Iterationen zu begrenzen. Dies ist besonders nützlich, um zu verhindern, dass die Schleife unendlich läuft, falls keine Konvergenz erreicht wird oder das Abbruchkriterium aufgrund eines Fehlers im Code nicht erfüllt wird. Bei Optimierungsaufgaben, wo potenziell kein optimaler Punkt gefunden wird oder dieser schwer zu erreichen ist, begrenzt eine `for`-Schleife effektiv die Anzahl der Versuche und verhindert so eine unendliche Ausführung, die Ressourcen verschwendet.

#### 6.1.2 Vorhersehbarkeit und Kontrolle

Die `for`-Schleife bietet eine klar definierte Struktur: Sie wissen im Voraus, wie oft der Block maximal ausgeführt wird. Dies erleichtert die Kontrolle über den Algorithmus und macht das Verhalten des Codes vorhersehbarer. Bei der Verwendung von `while true` ist dies weniger offensichtlich, da die Abbruchbedingung innerhalb des Schleifenkörpers erfüllt werden muss, um die Ausführung zu stoppen.

#### 6.1.3 Effizienz und Ressourcenmanagement

Die Beschränkung auf eine maximale Anzahl von Iterationen durch eine `for`-Schleife hilft, den Ressourcenverbrauch (z.B. CPU-Zeit, Speicher) zu kontrollieren und zu verwalten. Dies ist besonders wichtig in Umgebungen, wo die Rechenleistung oder Speicher begrenzt sind, oder bei Anwendungen, die in Echtzeit oder auf eingebetteten Systemen laufen.

#### 6.1.4 Code-Klarheit

`for`-Schleifen machen den Code oft klarer und leichter zu verstehen, besonders wenn die Iteration eine feste, vordefinierte Anzahl von Durchläufen hat. Bei einer `while true`-Schleife müssen Sie die Abbruchbedingungen innerhalb der Schleife genau überprüfen, was zu mehr Komplexität führen kann.

Hier ist ein Beispiel, wie Sie Ihren Optimierungsalgorithmus mit einer `for`-Schleife in MATLAB implementieren könnten:

```
% Matrix z erstellen  
z = peaks(10);
```

```

% Startpunkt definieren
x = 2;
y = 9;

% Maximale Anzahl von Iterationen
maxIter = 100;

% Optimierungsschleife
for k = 1:maxIter
    current_value = z(y, x);

    % Bestimmen des nächsten Punktes (vereinfachte Logik)
    % Hier sollte Ihre Logik zur Bestimmung des nächsten Punktes sein
    % Zum Beispiel: Überprüfen benachbarter Punkte

    % Beispiel für einen zufälligen nächsten Punkt
    next_x = x + randi([-1, 1]);
    next_y = y + randi([-1, 1]);
    next_value = z(next_y, next_x);

    % Prüfen, ob der neue Punkt besser ist
    if next_value <= current_value
        % Keine Verbesserung gefunden, Schleife verlassen
        fprintf('Keine Verbesserung nach %d Iterationen gefunden.\n', k);
        break;
    else
        % Aktualisieren des Punktes
        x = next_x;
        y = next_y;
    end
end
end

```

Dieses Beispiel ist stark vereinfacht und setzt voraus, dass die neue Position innerhalb der gültigen Matrixgrenzen liegt. Die Logik zur Bestimmung des nächsten Punktes müsste entsprechend Ihrer spezifischen Anforderungen entwickelt werden.

*Passt.*

## 7 Frage

In der Schleife stellen Sie als erstes den aktuellen Punkt mit dem gerade ausgetesteten `scatter3`-Befehl dar. Um mitverfolgen zu können, wie der Optimierer sich gemütlich den Berg hoch schlängelt, spendieren Sie dann bei jedem Schleifendurchlauf eine Pause von 100ms.

In welcher Einheit wird der Parameter des `pause`-Befehls angegeben?

## 7.1 Antwort

Der Parameter des `pause`-Befehls in MATLAB wird in Sekunden angegeben. Wenn Sie eine Pause von 100 Millisekunden einlegen möchten, müssen Sie den Parameter entsprechend als (0.1) Sekunden angeben, da (100) Millisekunden (0.1) Sekunden entsprechen.

Hier ist ein Beispiel, wie Sie den `pause`-Befehl innerhalb Ihrer Schleife verwenden könnten:

```
% Matrix z erstellen
z = peaks(10);

% Startpunkt definieren
x = 2;
y = 9;

% Maximale Anzahl von Iterationen
maxIter = 100;

% Erstellen einer neuen Figur für die Visualisierung
figure;
hold on;
xlabel('X-Achse (Spaltenindex)');
ylabel('Y-Achse (Zeilenindex)');
zlabel('Z-Wert');
rotate3d on;

% Optimierungsschleife
for k = 1:maxIter
    current_value = z(y, x);

    % Darstellen des aktuellen Punktes
    scatter3(x, y, current_value, 'ro', 'filled');

    % Beispiel für einen zufälligen nächsten Punkt
    next_x = x + randi([-1, 1], 1);
    next_y = y + randi([-1, 1], 1);

    % Begrenzung auf gültige Matrixgrenzen
    next_x = max(1, min(10, next_x));
    next_y = max(1, min(10, next_y));

    next_value = z(next_y, next_x);

    % Pause für 100ms
    pause(0.1);

    % Prüfen, ob der neue Punkt besser ist
    if next_value <= current_value
        % Keine Verbesserung gefunden, Schleife verlassen
        break;
    end
end
```

```

        fprintf('Keine Verbesserung nach %d Iterationen gefunden.\n', k);
        break;
    else
        % Aktualisieren des Punktes
        x = next_x;
        y = next_y;
    end
end

% Entfernen des Haltebefehls
hold off;

```

Dieses Skript visualisiert, wie der Optimierer durch die Landschaft navigiert, indem es jeden neuen Punkt rot und gefüllt darstellt. Die `pause` von 0.1 Sekunden zwischen den Iterationen ermöglicht es, die Bewegung des Optimierers visuell zu verfolgen.

*Ja, aber das zufällige Auswählen der "Nachbarn" ist ja nicht so richtig zielführend.*

## 8 Frage

Jetzt möchten Sie alle Nachbarn des aktuellen Punktes analysieren. Definieren Sie dazu die  $3 \times 3$ -Untermatrix von  $z$  um den aktuellen Punkt und weisen Sie sie der Variablen  $g$  zu.

Wie viele Nachbarn untersuchen Sie also?

Erwischen Sie bei dieser quadratischen Untermatrix also auch die diagonalen Nachbarn des aktuellen Punktes?

### 8.1 Antwort

Wenn Sie eine  $(3 \times 3)$ -Untermatrix um den aktuellen Punkt  $((x, y))$  in der Matrix  $(z)$  definieren, untersuchen Sie insgesamt neun Positionen: den aktuellen Punkt selbst sowie die acht umliegenden Punkte. Diese acht umliegenden Punkte sind sowohl die direkten Nachbarn (oben, unten, links, rechts) als auch die diagonalen Nachbarn. Also ja, bei dieser quadratischen Untermatrix erfassen Sie auch die diagonalen Nachbarn des aktuellen Punktes.

Die Variable  $(g)$ , die diese  $(3 \times 3)$ -Untermatrix speichert, würde wie folgt definiert, wobei darauf geachtet werden muss, dass die Grenzen der Matrix  $(z)$  nicht überschritten werden (d.h.,  $(x)$  und  $(y)$  sollten nicht am Rand der Matrix sein):

```

% Berechnung der Grenzen für die 3x3-Untermatrix
x_min = max(1, x-1);
x_max = min(size(z, 2), x+1);
y_min = max(1, y-1);
y_max = min(size(z, 1), y+1);

% Zuweisen der 3x3-Untermatrix zu g
g = z(y_min:y_max, x_min:x_max);

```

### 8.1.1 Analyse der (3 × 3)-Untermatrix:

- **Zentrum:** Der aktuelle Punkt ( ( x, y ) ).
- **Horizontale und vertikale Nachbarn:** Direkt benachbart oben ((y-1)), unten ((y+1)), links ((x-1)), und rechts ((x+1)).
- **Diagonale Nachbarn:** Oben links ((x-1, y-1)), oben rechts ((x+1, y-1)), unten links ((x-1, y+1)), und unten rechts ((x+1, y+1)).

Die Nutzung einer (3 × 3)-Matrix erlaubt es, eine vollständige lokale Nachbarschaftsanalyse durchzuführen, was besonders in Optimierungs- und Bildverarbeitungsaufgaben nützlich sein kann, wo lokale Minima oder Maxima von Interesse sind. Jede Position in dieser Untermatrix gibt Ihnen Informationen über den lokalen Gradienten oder die Topologie der Daten, was für Schritte zur Optimierung oder für Filteroperationen in der Bildverarbeitung hilfreich sein kann.

*Verstanden*

## 9 Frage

Jetzt müssen Sie im Folgenden eigentlich nur noch die Koordinaten des größten Nachbarn finden. Dies ist allerdings leider nicht trivial, da es keinen Matlab-Befehl gibt, um direkt die Indizes des größten Elementes einer Matrix zu erhalten.

Was liefert der max-Befehl, wenn Sie ihn auf eine Matrix anwenden?

### 9.1 Antwort

Wenn Sie den max-Befehl in MATLAB auf eine Matrix anwenden, gibt er standardmäßig für jede Spalte der Matrix das größte Element zurück. Der Befehl behandelt die Matrix spaltenweise und liefert zwei Arten von Ausgaben:

1. **Die maximalen Werte jeder Spalte:** Dies ist das erste Ergebnis des Befehls max, wenn er auf eine Matrix angewendet wird.
2. **Die Indizes der maximalen Werte jeder Spalte:** Wenn Sie zwei Ausgabewerte angeben, gibt MATLAB auch die Indizes innerhalb jeder Spalte zurück, wo diese maximalen Werte gefunden wurden.

Hier ein Beispiel, wie Sie den max-Befehl auf eine Matrix anwenden können:

```
A = [1 2 3; 4 5 6; 7 8 9];  
[maxVals, rowIndices] = max(A);
```

In diesem Fall gibt maxVals die Werte [7 8 9] zurück, weil dies die maximalen Werte jeder Spalte sind, und rowIndices gibt [3 3 3] zurück, weil diese maximalen Werte alle in der dritten Zeile jeder Spalte auftreten.

#### 9.1.1 Um den Index des größten Elements in der gesamten Matrix zu finden

Um den Index des absolut größten Elements in der gesamten Matrix zu ermitteln, kann man eine Kombination aus max und der Funktion ind2sub verwenden, um den linearen Index in zweidimensionale Indizes umzuwandeln. Hier ist ein Beispiel:

```
% Angenommen, g ist die 3x3-Untermatrix
[maxValCol, rowIndex] = max(g); % Findet Maxima jeder Spalte und ihre Zeilenindizes
[maxVal, colIndex] = max(maxValCol); % Findet das maximale Element aus den Spaltenmaxima
rowIndex = rowIndex(colIndex); % Den Zeilenindex des maximalen Wertes aus dem vorherigen Sch
```

*% rowIndex und colIndex sind nun die Koordinaten des größten Werts innerhalb von g*

Dieser Ansatz hilft dabei, den größten Wert in der Matrix und seine genaue Position zu finden, was für Optimierungsaufgaben nützlich sein kann.

*Das ist sicherlich eine Möglichkeit.*

## 10 Frage

Der im Folgenden angewandte “Trick” besteht darin, die Matrix spaltenweise in einen Vektor zu überführen, dort den (einen) Index des Maximums zu finden und aus diesem Index rückwärts wieder die Indizes der Matrix zu ermitteln. Das Überführen einer Matrix in einen Vektor ist einfach:

```
gg = g(:)
```

Beschreiben Sie den Transformationsvorgang.

Welcher Index läuft zuerst?

Ist gg ein Zeilen- oder ein Spaltenvektor? Geschieht die Umwandlung also zeilen- oder spaltenweise?

Etwas schwierigere Frage: Was vermuten Sie, wie Matrizen im linearen (eindimensionalen) Adressraum eines Speichers abgelegt werden?

### 10.1 Antwort

Die Transformation einer Matrix in einen Vektor in MATLAB mittels der Operation `g(:)` konvertiert die Matrix `g` in einen Vektor `gg` durch das Aneinanderreihen ihrer Spalten. Dieser Vorgang wird häufig verwendet, um matrix-basierte Operationen auf Vektor-basierte Operationen zu reduzieren oder um bestimmte Indexoperationen zu vereinfachen.

#### 10.1.1 Transformationsvorgang

Wenn Sie `g(:)` auf eine Matrix anwenden, geschieht Folgendes:

1. **Matrix zu Vektor:** Die Elemente jeder Spalte der Matrix werden der Reihe nach unten genommen und aneinandergereiht, um einen Vektor zu bilden. Dies wird für die erste Spalte, dann die zweite Spalte, und so weiter durchgeführt, bis alle Spalten abgearbeitet sind.
2. **Reihenfolge der Indizes:** Der Index der Matrixelemente im resultierenden Vektor läuft so, dass die Elemente einer Spalte nacheinander kommen, bevor die Elemente der nächsten Spalte folgen.

#### 10.1.2 Art des Vektors

- **gg ist ein Spaltenvektor:** Durch die Operation `g(:)` wird ein Spaltenvektor erzeugt. Dies ist standardmäßig der Fall in MATLAB, wenn eine Matrix zu einem Vektor umgeformt wird.

### 10.1.3 Umwandlungsmethode

- **Umwandlung erfolgt spaltenweise:** Da MATLAB die Matrixelemente spaltenweise liest und in den Vektor schreibt, ist die Umwandlung spaltenweise. Dies bedeutet, dass zuerst alle Elemente der ersten Spalte in den Vektor aufgenommen werden, gefolgt von allen Elementen der zweiten Spalte und so weiter.

### 10.1.4 Speicherung von Matrizen im Speicher

- **Speicherungsweise in MATLAB (und vielen anderen Programmiersprachen):** Matrizen werden in einem eindimensionalen Adressraum eines Speichers typischerweise spaltenweise gespeichert. Das bedeutet, dass die erste Spalte der Matrix im Speicher direkt vor der zweiten Spalte liegt. Diese Anordnung wird als "Column-major order" bezeichnet und ist charakteristisch für MATLAB sowie Fortran. Im Gegensatz dazu verwendet eine Sprache wie C oder Python (NumPy) standardmäßig eine "Row-major order", bei der die Elemente zeilenweise im Speicher abgelegt werden.

Die Kenntnis dieser Details ist besonders wichtig für effiziente Speicherzugriffe und kann bei der Optimierung von Code für speicherintensive oder performance-kritische Anwendungen von Vorteil sein.

*Wieder was gelernt: "Im Gegensatz dazu verwendet eine Sprache wie C oder Python (NumPy) standardmäßig eine "Row-major order", bei der die Elemente zeilenweise im Speicher abgelegt werden."*

## 11 Frage

Verwenden Sie jetzt den `max`-Befehl, um den linearen Index des maximalen Elementes zu erhalten.

Warum ist es sinnvoll, als ersten Rückgabeparameter des `max`-Befehls die Tilde (`~`) zu verwenden?

### 11.1 Antwort

In MATLAB wird die Tilde (`~`) verwendet, um Ausgabeparameter einer Funktion zu ignorieren, die Sie nicht benötigen oder verwenden möchten. Wenn Sie den `max`-Befehl anwenden, um den linearen Index des maximalen Elements einer Matrix zu finden, sind Sie möglicherweise nur am Index interessiert und nicht am Wert des maximalen Elements selbst. In diesem Fall können Sie die Tilde verwenden, um den Wert zu ignorieren und nur den Index zu speichern.

Der `max`-Befehl gibt in seiner einfachsten Form den maximalen Wert zurück, den er findet. Wenn Sie den Befehl jedoch so erweitern, dass er auch den Index des maximalen Werts zurückgibt, sieht das in MATLAB folgendermaßen aus:

```
[maxValue, index] = max(gg);
```

Wenn Sie nur am Index interessiert sind und den maximalen Wert selbst nicht benötigen, können Sie die Tilde (`~`) verwenden, um diesen Wert zu ignorieren:

```
[~, index] = max(gg);
```

### 11.1.1 Warum ist das sinnvoll?

1. **Speichereffizienz:** Indem Sie den nicht benötigten Wert ignorieren, vermeiden Sie es, Speicherplatz für eine Variable zu belegen, die Sie nicht verwenden werden.
2. **Code-Klarheit und -Wartbarkeit:** Die Verwendung der Tilde macht deutlich, dass der entsprechende Rückgabewert in dem folgenden Code nicht verwendet wird. Dies verbessert die Lesbarkeit und Wartbarkeit des Codes, da andere Entwickler (oder Sie selbst zu einem späteren Zeitpunkt) sofort sehen können, dass dieser Wert irrelevant ist.
3. **Vermeidung von Programmierfehlern:** Manchmal kann das unbeabsichtigte Überschreiben von Variablen zu Fehlern führen. Durch das Ignorieren nicht benötigter Ausgaben mit `~` verringern Sie das Risiko, wichtige Daten versehentlich zu überschreiben.

Die Verwendung der Tilde in solchen Kontexten ist eine gute Praxis in MATLAB und wird häufig in Situationen angewendet, in denen Funktionen mehrere Rückgabewerte liefern, von denen nicht alle benötigt werden.

*Sehr hilfreich!*

## 12 Frage

Verwenden Sie den Befehl `ind2sub`, um aus dem linearen Index des maximalen Elementes wieder seinen Zeilen- und Spaltenindex (bezüglich der Matrix `g`) zu erhalten.

Warum können Sie `size(g)` als ersten Parameter von `ind2sub` verwenden?

Und nochmal: In welcher Reihenfolge gibt der `ind2sub`-Befehl die x- bzw. die y-Komponente des Maximums zurück?

Noch eine etwas schwierigere Frage: Alternativ könnten Sie die Indizes des Matrizenmaximums mit folgenden Befehlen ermitteln:

```
m = max (max (g)) [delta_y, delta_x] = find (g == m)
```

Erläutern Sie, wie diese Alternative funktioniert und warum sie (bei großen Matrizen) rechenintensiver ist.

### 12.1 Antwort

#### 12.1.1 Verwendung von `ind2sub` zur Indexumwandlung

Der Befehl `ind2sub` ist in MATLAB sehr nützlich, um lineare Indizes, die Sie zum Beispiel mit dem `max`-Befehl erhalten haben, in mehrdimensionale Indizes umzuwandeln. Hier ist, wie Sie `ind2sub` verwenden können:

```
[~, linearIndex] = max(g(:)); % Finden des linearen Index des maximalen Wertes  
[rowIndex, colIndex] = ind2sub(size(g), linearIndex); % Umwandeln des linearen Indexes in Zeilen- und Spaltenindex
```

#### 12.1.2 Warum `size(g)` als Parameter verwenden?

Der erste Parameter von `ind2sub` muss die Größe der Matrix angeben, von der der lineare Index abgeleitet wurde. `size(g)` gibt genau diese Information als Vektor zurück, der die Anzahl der Zeilen und Spalten von `g` enthält. Dies ist notwendig, damit `ind2sub` korrekt interpretieren kann,

wie der lineare Index auf die mehrdimensionale (Zeilen und Spalten) Struktur der Matrix abgebildet wird.

### 12.1.3 Reihenfolge der Rückgabe von `ind2sub`

Der `ind2sub`-Befehl gibt die Indizes in der Reihenfolge Zeilenindex (`rowIndex`) und dann Spaltenindex (`colIndex`) zurück. Das bedeutet, dass der Zeilenindex der y-Komponente und der Spaltenindex der x-Komponente des Maximums entspricht.

### 12.1.4 Alternative Methode mit `max(max(g))` und `find(g == m)`

Die Alternative, die Sie erwähnen, verwendet zwei Schritte: 1. `m = max(max(g))` findet den maximalen Wert in der Matrix `g`. Dieser Befehl wendet `max` einmal an, um die Maxima jeder Spalte zu finden, und dann noch einmal, um das Maximum dieser Maxima zu finden. 2. `[delta_y, delta_x] = find(g == m)` sucht die Positionen in der Matrix `g`, wo der Wert gleich `m` ist. Dies könnte mehrere Indizes zurückgeben, falls das Maximum mehrmals in der Matrix vorkommt.

### 12.1.5 Warum ist diese Methode möglicherweise rechenintensiver?

- **Mehrfache Durchläufe:** Die erste Verwendung von `max` verarbeitet die gesamte Matrix, und die zweite Anwendung von `max` muss über die Ergebnisse der ersten Operation laufen. Dies kann bei großen Matrizen rechenintensiv sein.
- **Suchoperation über die gesamte Matrix:** Der `find`-Befehl muss die gesamte Matrix durchlaufen, um alle Vorkommen des maximalen Wertes zu identifizieren. Dies kann besonders ineffizient sein, wenn der maximale Wert häufig vorkommt oder die Matrix sehr groß ist.

Die erste Methode mit `max(g(:))` und `ind2sub` ist in der Regel effizienter, da sie direkt den maximalen Wert findet und seinen Index bestimmt, ohne die Matrix mehrmals durchlaufen zu müssen. Die Verwendung von `ind2sub` konvertiert diesen Index effizient in die korrespondierenden Zeilen- und Spaltenindizes.

*Hat sie verstanden!*

## 13 Frage

Verwenden Sie jetzt eine `if`-Abfrage, um die Schleife mit einem `break`-Befehl zu beenden, wenn der aktuelle Wert schon das Maximum ist.

Wie groß sind dann die gerade berechneten Indizes?

### 13.1 Antwort

Um eine `if`-Abfrage zu implementieren, die die Schleife mit einem `break`-Befehl beendet, wenn der aktuelle Wert bereits das Maximum der betrachteten 3x3-Untermatrix ist, können Sie den folgenden Ansatz verwenden. Hierbei wird geprüft, ob der aktuelle Wert an der Position  $(x, y)$  (relativ zur Matrix  $(g)$ ) dem maximalen Wert in  $(g)$  entspricht. Wenn dies der Fall ist, führt die Schleife einen `break` durch und die Optimierung ist abgeschlossen.

### 13.1.1 Beispiel-Implementierung

Angenommen,  $(g)$  ist die  $3 \times 3$ -Untermatrix, die den aktuellen Punkt und seine Nachbarn umfasst:

```
% Bestimmen des maximalen Wertes und dessen linearen Index
[maxVal, maxIndex] = max(g(:));

% Bestimmen des Zeilen- und Spaltenindex des maximalen Wertes
[rowOfMax, colOfMax] = ind2sub(size(g), maxIndex);

% Position des aktuellen Punktes in g (mitten in der Matrix)
currentRow = 2;
currentCol = 2;
currentVal = g(currentRow, currentCol);

% Prüfen, ob der aktuelle Wert bereits das Maximum ist
if currentVal == maxVal
    % Da maxVal das Maximum der 3x3 Matrix ist und currentVal diesem entspricht,
    % befinden wir uns bereits am maximalen Punkt
    disp('Der aktuelle Punkt ist bereits das Maximum.');
```

```
    break; % Beenden der Schleife
end
```

### 13.1.2 Größe der berechneten Indizes

Wenn die `if`-Abfrage wahr ist (d.h., wenn der aktuelle Punkt bereits das Maximum ist), dann sind die berechneten Indizes (`rowOfMax`) und (`colOfMax`) gleich den Indizes des aktuellen Punktes in der Untermatrix ( $g$ ), also 2, 2. Das liegt daran, dass in einer  $3 \times 3$ -Matrix, die um einen zentralen Punkt herum aufgebaut ist, der Mittelpunkt (2, 2) der zentrale Punkt ist, wenn die Indizierung bei 1 beginnt.

### 13.1.3 Warum könnte das sinnvoll sein?

Die Überprüfung, ob der aktuelle Punkt bereits das Maximum ist, ist ein effektiver Weg, um unnötige Berechnungen zu vermeiden und die Schleife vorzeitig zu beenden, wenn keine weitere Verbesserung möglich ist. Dies ist besonders nützlich in Optimierungsszenarien, wo das Ziel ist, den höchsten Wert in einem gegebenen Bereich zu finden. Durch das sofortige Beenden der Schleife, sobald das Maximum erreicht ist, spart der Algorithmus Zeit und Ressourcen.

*Ja, so ausführlich wollte ich das gar nicht.*

## 14 Frage

Wenn dies nicht der Fall ist, verwenden Sie die gerade berechneten Indizes des “besseren” Nachbarn, um die  $x$ - und  $y$ -Koordinate des nächsten aktuellen Punktes zu berechnen.

Warum müssen Sie bei der Berechnung der neuen Koordinaten jeweils 2 abziehen?

## 14.1 Antwort

Wenn Sie die gerade berechneten Indizes des “besseren” Nachbarn verwenden, um die neuen (x) und (y) Koordinaten des aktuellen Punktes in der größeren Matrix (z) zu aktualisieren, müssen Sie jeweils 2 abziehen, um die korrekte Verschiebung zu berechnen. Dies ist notwendig, weil die Indizes (rowOfMax) und (colOfMax) relativ zur 3x3-Untermatrix (g) sind, nicht zur gesamten Matrix (z).

### 14.1.1 Kontext der Indexanpassung

Angenommen, der aktuelle Punkt befindet sich in der Matrix (z) an der Position (x, y). Die umgebende 3x3-Untermatrix (g) wurde so definiert, dass sie den Punkt (x, y) als Mittelpunkt hat. Das bedeutet:

- Der Mittelpunkt von (g) (also der Punkt (x, y) in (z)) entspricht den Indizes (2, 2) in (g) (wenn die Indizierung bei 1 beginnt).

### 14.1.2 Warum 2 abziehen?

Wenn Sie zum Beispiel feststellen, dass der beste benachbarte Punkt in (g) bei den Indizes (3, 3) liegt, sind diese Indizes relativ zu (g). Um die entsprechenden Koordinaten in der größeren Matrix (z) zu finden, müssen Sie berücksichtigen, dass der Mittelpunkt (g[2, 2]) den Koordinaten (x, y) in (z) entspricht. Das bedeutet:

- Der Index (3) in (g) entspricht dem Index (x+1) oder (y+1) in (z), weil es eine Position rechts bzw. unten vom Mittelpunkt in (g) ist.
- Um den Offset von (g[2, 2]) (dem Mittelpunkt) zu (g[3, 3]) (neuer Punkt) zu berechnen, subtrahieren Sie 2 von den Indizes, die Sie aus der (max) Berechnung erhalten, um die relative Verschiebung vom Mittelpunkt aus zu ermitteln (3 - 2 = 1).
- Die neuen Koordinaten in (z) werden dann (x + (rowOfMax - 2)) und (y + (colOfMax - 2)), wobei (rowOfMax) und (colOfMax) die aus der (max) Funktion erhaltenen Indizes sind.

### 14.1.3 Beispiel

```
% Angenommen, rowOfMax und colOfMax sind die Indizes des maximalen Wertes in g  
new_x = x + (colOfMax - 2);  
new_y = y + (rowOfMax - 2);
```

Diese Berechnung verschiebt effektiv die Koordinaten von der lokalen 3x3-Matrix (g) auf die umfassendere Matrix (z), indem sie die lokale Position des neuen Punktes relativ zum Mittelpunkt berücksichtigt. Dies ist ein entscheidender Schritt, um sicherzustellen, dass der Algorithmus korrekt im größeren Kontext der Matrix (z) navigiert.

*Gut erklärt!*

## 15 Frage

Nach der Schleife möchten Sie sich vielleicht noch die Koordinaten des gefundenen Maximums und die Anzahl der Suchschritte anzeigen lassen.

Verwenden Sie außerdem den Befehl text, um die Koordinaten des gefundenen Maximums etwas oberhalb des Maximums in das Diagramm zu schreiben.

Warum brauchen Sie dazu den `num2str`-Befehl?

## 15.1 Antwort

Um die Koordinaten des gefundenen Maximums und die Anzahl der durchgeführten Suchschritte nach dem Ende der Schleife anzuzeigen, ist es hilfreich, zusätzliche Informationen direkt in die grafische Darstellung einzufügen. Der `text`-Befehl in MATLAB ermöglicht es Ihnen, Text in einem Plot zu positionieren. Wenn Sie jedoch numerische Werte im Text darstellen wollen, müssen Sie diese zuerst in Zeichenketten (Strings) umwandeln, da der `text`-Befehl nur Zeichenketten als Eingabe für den darzustellenden Text akzeptiert. Hier kommt der `num2str`-Befehl ins Spiel.

### 15.1.1 Der `num2str`-Befehl

`num2str` steht für “number to string” und ist ein MATLAB-Befehl, der numerische Werte in Zeichenketten umwandelt. Dies ist notwendig, weil Sie numerische Ergebnisse (wie Koordinaten oder Zählvariablen) typischerweise nicht direkt in Funktionen verwenden können, die Text erfordern.

### 15.1.2 Beispielanwendung

Nehmen wir an, Sie haben die finale Position des Maximums in den Variablen `final_x` und `final_y` und die Anzahl der Suchschritte in `num_steps`. Hier ist, wie Sie diese Informationen grafisch darstellen könnten:

```
figure;
surf(z); % Angenommen z ist die Matrix, die Sie visualisieren
hold on;
% Markieren des Maximums
plot3(final_x, final_y, z(final_y, final_x), 'ro');

% Anzeigen der Koordinaten des Maximums im Plot
text(final_x, final_y, z(final_y, final_x) + 0.5, ['Max bei (', num2str(final_x), ',', num2str(
final_y), ')'], 'r');

% Anzeigen der Anzahl der Suchschritte
title(['Gefundenes Maximum nach ', num2str(num_steps), ' Schritten']);

hold off;
```

### 15.1.3 Warum `num2str` verwenden?

- **Kompatibilität:** Der `text`-Befehl erwartet einen String als Eingabe für den Textparameter. `num2str` konvertiert die numerischen Werte in Strings, sodass sie in den Text integriert werden können.
- **Flexibilität:** Mit `num2str` können Sie auch das Format der Zahlen anpassen, was für die Darstellung wissenschaftlicher Daten nützlich sein kann.
- **Benutzerfreundlichkeit:** Die Darstellung numerischer Daten als Text im Kontext einer grafischen Visualisierung verbessert die Lesbarkeit und Verständlichkeit der dargestellten Informationen.

Die Verwendung von `num2str` ist eine elegante Lösung, um numerische Daten nahtlos in textuelle Anmerkungen innerhalb grafischer Darstellungen in MATLAB zu integrieren.

Jo.

## 16 Frage

Nachdem Sie das Konzept an einer  $10 \times 10$ -Matrix verstanden haben, erzeugen Sie jetzt mit dem `peaks`-Befehl eine  $100 \times 100$ -Matrix.

Wie lauten dann die Koordinaten des Startwertes, wenn er wieder (aus Kamerasicht) in der linken Ecke liegen soll? Warum können Sie die Optimierung nicht mit `[1, 100]` starten lassen? Beschreiben Sie, auf welchem Weg der Optimierer jetzt das Maximum findet.

### 16.1 Antwort

Bei einer Vergrößerung der Matrix auf eine  $100 \times 100$ -Größe durch den `peaks`-Befehl in MATLAB wird die Komplexität der Oberfläche und damit auch die Herausforderung für den Optimierungsalgorithmus erhöht. Der `peaks`-Befehl erzeugt eine Funktion mit mehreren lokalen Maxima und Minima, wobei die Größe der Matrix die Auflösung und damit die Anzahl dieser Extrema beeinflusst.

#### 16.1.1 Koordinaten des Startwertes

Wenn der Startwert wieder aus der "Kamerasicht" in der linken unteren Ecke liegen soll, würden Sie typischerweise mit den Koordinaten `[1, 1]` starten. In MATLAB werden Indizes bei 1 begonnen, sodass die Koordinate `(1, 1)` tatsächlich die linke untere Ecke der Matrix darstellt.

#### 16.1.2 Warum nicht `[1, 100]` als Startwert?

Wenn Sie mit `[1, 100]` starten, setzen Sie den Startpunkt in die obere linke Ecke der Matrix (erste Zeile, letzte Spalte). Dies ist nicht ideal, da dies eine Randposition ist. Randpositionen haben den Nachteil, dass nicht alle umgebenden Nachbarn existieren, was die Richtungen einschränkt, in die der Optimierer bewegen kann. Außerdem tendieren bei der `peaks`-Funktion die Werte an den Rändern dazu, weniger signifikante Extrema zu sein (oftmals sind sie näher an den Minima als an den zentralen Maxima). Daher ist die Wahrscheinlichkeit geringer, dass von dieser Position aus ein globales Maximum effizient gefunden werden kann.

#### 16.1.3 Weg des Optimierers zum Maximum

Der Weg des Optimierers zum Maximum hängt stark von der Startposition und der Struktur der `peaks`-Matrix ab. Hier ist, wie der Prozess typischerweise abläuft:

1. **Initialisierung:** Der Algorithmus startet bei `[1, 1]`, analysiert die Werte der  $3 \times 3$ -Nachbarschaft um diesen Punkt.
2. **Lokale Suche:** In jedem Schritt sucht der Algorithmus innerhalb der lokalen  $3 \times 3$ -Nachbarschaft nach dem höchsten Wert.
3. **Bewegung:** Der Optimierer bewegt sich zum höchsten Wert in dieser Nachbarschaft.
4. **Iteration:** Dieser Prozess wiederholt sich, wobei in jedem Schritt die umgebenden Werte erneut evaluiert werden, bis keine Verbesserung mehr erreicht wird (d.h., der höchste Punkt in der lokalen Nachbarschaft ist der Punkt, auf dem der Optimierer bereits steht).

5. **Termination:** Die Suche endet, wenn der Optimierer einen Punkt erreicht, der keine höheren benachbarten Punkte hat, was bedeutet, dass ein lokales Maximum erreicht ist. Bei der **peaks**-Funktion könnte dies entweder ein globales oder ein lokales Maximum sein, abhängig davon, wo der Startpunkt lag und welche Pfade der Optimierer genommen hat.

Die Effizienz und die Fähigkeit, das globale Maximum zu finden (statt eines lokalen Maximums), hängen stark von der initialen Platzierung des Startpunkts ab. In größeren Matrizen mit vielen Extrema, wie bei einer 100 x 100 **peaks**-Matrix, kann dies besonders herausfordernd sein. In solchen Fällen könnten fortgeschrittenere Techniken wie simulierte Abkühlung (simulated annealing), genetische Algorithmen oder Gradientenverfahren erforderlich sein, um die Wahrscheinlichkeit zu erhöhen, das globale Maximum zu erreichen.

*Das war ja mal gar nix! Die linke Ecke liegt nicht bei [1, 1] sondern allenfalls bei [1, 100]. [1, 1] liegt aber ja auch auf dem Rand und sollte deshalb nicht genommen werden.*

## 17 Frage

Verwenden Sie als Startwert [2, 2].

Wo landet der Optimierer jetzt? Warum ist das so?

### 17.1 Antwort

Die **peaks**-Funktion in MATLAB erzeugt eine künstliche Landschaft mit mehreren lokalen Maxima und Minima. Die genaue Position, an der der Optimierer landet, hängt von den spezifischen Werten in der Nähe des Startpunkts ab. Wenn Sie als Startpunkt [2, 2] wählen, beginnen Sie nahe der Ecke, haben aber Raum für Bewegung in alle Richtungen, da dies nicht auf einem Rand liegt.

#### 17.1.1 Simulation des Prozesses

Um zu verstehen, wo der Optimierer mit einem Startpunkt bei [2, 2] in einer 100x100 **peaks**-Matrix landen könnte, können wir einen einfachen Gradientenanstieg-Algorithmus simulieren. Hier ist ein einfacher MATLAB-Code, um diesen Prozess zu visualisieren:

```
% Erzeugen der 100x100 peaks-Matrix
z = peaks(100);

% Startposition
x = 2;
y = 2;

% Maximale Anzahl von Iterationen
maxIter = 1000;

% Visualisierung
figure;
surf(z);
hold on;

% Optimierungsschleife
```

```

for k = 1:maxIter
    % Aktuellen Wert abrufen
    currentVal = z(y, x);

    % 3x3-Nachbarschaft um den aktuellen Punkt analysieren
    [localMax, ind] = max(z(max(y-1,1):min(y+1,100), max(x-1,1):min(x+1,100)), [], 'all', 'li');
    [dy, dx] = ind2sub([3, 3], ind);

    % Neue Position berechnen
    newY = y + dy - 2;
    newX = x + dx - 2;

    % Prüfen, ob der neue Punkt besser ist
    if z(newY, newX) <= currentVal
        fprintf('Der Optimierer landet bei (%d, %d) nach %d Schritten.\n', x, y, k);
        scatter3(x, y, z(y, x) + 0.5, 'ro', 'filled'); % Markieren des Endpunkts
        break;
    else
        % Aktualisieren der Position
        x = newX;
        y = newY;
        scatter3(x, y, z(y, x) + 0.5, 'yo'); % Markieren des Pfads
    end
end

hold off;

```

### 17.1.2 Warum der Optimierer dort landet

Der oben skizzierte Algorithmus führt einen einfachen Gradientenanstieg durch, der jeweils die lokal beste Option wählt. Da die `peaks`-Funktion jedoch mehrere lokale Maxima aufweist, kann der Optimierer in einem dieser lokalen Maxima “stecken bleiben”, bevor er das globale Maximum erreicht, wenn es keine direkte, aufsteigende Route vom aktuellen lokalen Maximum zum globalen Maximum gibt.

Wenn der Startpunkt  $[2, 2]$  ist, wird der Optimierer sich zu dem nächstgelegenen Punkt bewegen, der höher ist als der aktuelle. Dieser Prozess wiederholt sich, bis keine benachbarten Punkte mehr einen höheren Wert haben. Je nach der genauen Topologie der `peaks`-Funktion könnte dieser Punkt ein globales Maximum oder häufiger ein lokales Maximum sein.

### 17.1.3 Einfluss des Startpunkts

Der Startpunkt  $[2, 2]$  ist relativ nah am Rand der Matrix und könnte nahe an einem der tieferen Täler von `peaks` liegen, was den Weg zum höchsten Gipfel erschwert. Der Pfad des Optimierers und der Endpunkt hängen stark von den initialen und umgebenden Werten ab. Ohne Kenntnis der spezifischen Werte oder eine Visualisierung (wie im bereitgestellten Code) ist es schwer vorherzusagen, wo genau der Optimierer enden wird, insbesondere in einer komplexen Landschaft wie der von `peaks` erzeugten.

*Da hat sich durchaus den richtigen Riecher.*

*Spannend: Offensichtlich kann der INFO-Assistent kein Python ausführen.*

## 18 Frage

Probieren Sie den Startwert [99, 2].

Warum klappt das nicht?

### 18.1 Antwort

In MATLAB beginnt die Indizierung von Arrays und Matrizen bei 1, und wenn Sie eine Matrix **z** mit der Funktion **peaks** erzeugt haben, die zum Beispiel 100 x 100 groß ist, dann sind die gültigen Indizes für jede Dimension von 1 bis 100. Wenn Sie versuchen, mit dem Startwert [99, 2] in einer solchen Matrix zu starten, liegen Sie noch innerhalb der gültigen Bereichsgrenzen für die Indizes der Matrix, sofern die Matrix groß genug ist (zum Beispiel eine 100x100-Matrix).

Allerdings könnte es Probleme geben, wenn die Dimensionen der Matrix kleiner sind als der angegebene Index. Hier sind einige Gründe, warum es nicht klappen könnte, abhängig von der tatsächlichen Größe der **peaks**-Matrix:

1. **Matrixgröße:** Wenn Ihre **peaks**-Matrix kleiner als 100x100 ist, dann existiert der Index [99, 2] möglicherweise nicht, und ein Zugriffsversuch auf diesen Index würde einen Fehler verursachen.
2. **Randlage:** Selbst wenn die Matrix 100x100 groß ist, liegt der Punkt [99, 2] sehr nahe am unteren Rand der Matrix. Wenn Sie versuchen, eine 3x3-Untermatrix um diesen Punkt zu bilden, um die lokale Nachbarschaft zu untersuchen, würden Sie versuchen, auf Elemente zuzugreifen, die außerhalb der Grenzen der Matrix liegen (z.B. Zeile 100+1=101), was zu einem Fehler führt.

Um sicherzustellen, dass Sie den Startpunkt korrekt wählen und keine Indexfehler auftreten, sollten Sie folgende Überlegungen treffen:

- **Prüfen der Matrixdimensionen:** Stellen Sie sicher, dass die **peaks**-Matrix groß genug ist. Dies können Sie überprüfen mit **size(z)**, welche die Dimensionen der Matrix zurückgibt.
- **Anpassung des Startpunktes:** Vermeiden Sie Startpunkte zu nah am Rand, wenn Sie eine lokale Nachbarschaftsanalyse (wie eine 3x3-Matrix um den Startpunkt) durchführen möchten, um sicherzustellen, dass keine Indexüberschreitung stattfindet.

Wenn Sie mit der **peaks**-Funktion arbeiten und diese auf eine 100x100-Matrix anwenden, sollte der Zugriff auf [99, 2] normalerweise funktionieren, solange Sie die Analyse der Nachbarschaft innerhalb der gültigen Grenzen halten. Hier ist ein einfacher Weg, um sicherzustellen, dass Sie nicht über die Grenzen der Matrix hinausgehen:

```
x = 99; y = 2;
z = peaks(100); % Erstellen einer 100x100 Matrix
if x > 1 && x < size(z,1) && y > 1 && y < size(z,2)
    % Berechnung und Logik hier
else
```

```
disp('Startpunkt liegt zu nahe am Rand oder außerhalb der Matrixgrenzen.')  
end
```

Dieser Code stellt sicher, dass der Startpunkt nicht zu nahe am Rand liegt, um einen sicheren Zugriff auf die Nachbarschaft zu gewährleisten.

*Ja, aber ohne das selbst auszuprobieren, weiß sie natürlich nicht, dass die Optimierung von diesem Startwert an den Rand läuft.*

## 19 Frage

Verwenden Sie als Startwert [28, 55].

Beschreiben und erklären Sie den interessanten Weg, den der Optimierer nimmt.

### 19.1 Antwort

Um den Weg eines Optimierers in MATLAB zu verstehen, der mit der `peaks`-Funktion und einem Startwert von [28, 55] arbeitet, müssen wir zunächst ein wenig über die Natur der `peaks`-Funktion und ihre Topographie wissen. Die `peaks`-Funktion erzeugt eine Matrix, die eine dreidimensionale Oberfläche repräsentiert, welche mehrere lokale Maxima und Minima enthält. Diese Oberfläche ist nicht nur für visuelle Demonstrationen von Optimierungsalgorithmen geeignet, sondern auch um die Herausforderungen von Optimierungsalgorithmen in realen Szenarien zu simulieren, insbesondere wenn es mehrere lokale Optima gibt.

#### 19.1.1 Startpunkt [28, 55]

Der Startpunkt [28, 55] in der `peaks`-Matrix positioniert den Optimierer an einer spezifischen Stelle auf dieser Oberfläche. Ohne die spezifische Matrix zu sehen, ist es schwer zu sagen, wo genau dieser Punkt liegt in Bezug auf die Höhen und Tiefen der `peaks`-Funktion. Wir können jedoch einige allgemeine Annahmen über den Weg des Optimierers machen, basierend auf den Eigenschaften der `peaks`-Funktion und typischen Verhaltensweisen von Optimierungsalgorithmen wie dem Gradientenanstieg.

#### 19.1.2 Möglicher Weg des Optimierers

1. **Initialer Anstieg:** Der Optimierer beginnt am Punkt [28, 55] und untersucht die unmittelbare Nachbarschaft für höhere Werte. Die Richtung, in die er sich zuerst bewegt, hängt davon ab, in welcher Richtung die nächstgelegenen höheren Werte liegen.
2. **Navigation durch lokale Maxima:** Aufgrund der mehrfachen Maxima der `peaks`-Funktion könnte der Optimierer von einem lokalen Maximum zum nächsten springen. Jeder Schritt wird basierend auf der unmittelbaren Umgebung bewertet, was bedeutet, dass der Optimierer möglicherweise in einem lokalen Maximum stecken bleibt, wenn keine benachbarten Punkte höhere Werte aufweisen.
3. **Suche nach dem globalen Maximum:** Der entscheidende Teil des Weges wird sein, ob der Optimierer Wege findet, lokale Maxima zu überwinden und sich in Richtung des globalen Maximums zu bewegen. Dies ist oft der schwierigste Teil, da lokale Maxima den Algorithmus "fangen" können.

4. **Endposition:** Der endgültige Standpunkt des Optimierers hängt davon ab, ob er das globale Maximum erreicht oder in einem lokalen Maximum stecken bleibt. Dies zeigt die Begrenzung einfacher Optimierungsalgorithmen in komplexen Landschaften.

### 19.1.3 Visualisierung und Code-Beispiel in MATLAB

Um dies in MATLAB zu visualisieren, könnten Sie den folgenden Code verwenden:

```
% Erzeugen der peaks-Matrix
Z = peaks(100);

% Visualisierung der Oberfläche
figure;
surf(Z);
hold on;

% Startpunkt
x = 28;
y = 55;

% Markieren des Startpunkts
plot3(x, y, Z(y,x), 'ro', 'MarkerSize', 10, 'MarkerFaceColor', 'r');

% Simulieren des Optimierungswegs (Pseudocode, anpassen nach Bedarf)
for iter = 1:100 % Maximale Iterationsanzahl
    % Bestimmen der Nachbarschaft und finden des maximalen Wertes
    % (Überlassen der Details dem Anwender, einfache if-Logik hier zur Demonstration)
    if iter == 1
        plot3(x, y, Z(y,x), 'yo', 'MarkerSize', 10, 'MarkerFaceColor', 'y');
    else
        break;
    end
end
hold off;
```

Dieses Skript zeigt nur den Startpunkt und bereitet eine Schleife vor, die Sie mit einer tatsächlichen Logik für den Optimierungsweg füllen müssen, abhängig von Ihrer spezifischen Implementierung des Optimierungsalgorithmus.

*Ja, kann sie ja auch nicht können.*

## 20 Frage

Finden Sie einen Startpunkt im gleichen Tal, der Sie auf direktem Weg zum globalen Optimum führt.

Erläutern Sie, warum solch eine geringe Startwertvarianz zu solch einem signifikant anderen Verhalten des Optimierers führt.

## 20.1 Antwort

Die `peaks`-Funktion in MATLAB generiert eine Oberfläche mit verschiedenen Höhen und Tiefen, einschließlich mehrerer lokaler Maxima und Minima. Diese komplex strukturierte Landschaft ist ideal, um die Herausforderungen bei der Optimierung in realen Situationen zu simulieren, insbesondere wenn das Ziel darin besteht, globale Maxima zu finden, die von vielen lokalen Maxima umgeben sind.

### 20.1.1 Auswahl eines optimalen Startpunkts

Um einen Startpunkt zu finden, der den Optimierer direkt zum globalen Optimum führt, müssen Sie die Topologie der spezifischen `peaks`-Oberfläche verstehen. Ein effektiver Weg, um einen solchen Punkt zu identifizieren, ist das manuelle Untersuchen der Oberfläche oder das Verwenden von Analysetools, die die Gradienten der Oberfläche visualisieren. Ein direkter Weg zum globalen Maximum ist wahrscheinlicher, wenn Sie in einer "glatten" Region starten, die direkt zum globalen Maximum führt ohne von höheren Barrieren umgeben zu sein.

### 20.1.2 MATLAB-Beispiel

Um den Prozess zu veranschaulichen, könnte man zuerst das globale Maximum von `peaks` finden und dann rückwärts arbeiten, um zu sehen, welche Pfade potenziell dorthin führen könnten. Hier ist ein einfacher MATLAB-Code, um das globale Maximum einer `peaks`-Matrix zu identifizieren und dann eine Visualisierung zu erstellen:

```
% Generiere die Daten
Z = peaks(100);

% Finde das globale Maximum
[maxVal, linearIdx] = max(Z(:));
[yMax, xMax] = ind2sub(size(Z), linearIdx);

% Visualisiere die Oberfläche
figure;
surf(Z);
hold on;
title('Peaks-Funktion mit globalem Maximum');
xlabel('X-Achse');
ylabel('Y-Achse');
zlabel('Z-Wert');

% Markiere das globale Maximum
plot3(xMax, yMax, maxVal, 'kp', 'MarkerSize', 10, 'MarkerFaceColor', 'k');

% Empfohlener Startpunkt (Beispiel)
xStart = xMax - 10; % Annahme, etwas links vom globalen Maximum
yStart = yMax;

% Markiere den Startpunkt
plot3(xStart, yStart, Z(yStart, xStart), 'mo', 'MarkerSize', 10, 'MarkerFaceColor', 'm');
```

hold off;

### 20.1.3 Warum führt eine kleine Änderung des Startpunkts zu einem signifikant anderen Verhalten?

1. **Empfindlichkeit gegenüber lokalen Maxima:** Optimierungsalgorithmen, insbesondere solche, die auf Gradienten basieren, sind sehr empfindlich gegenüber der lokalen Topographie der Startposition. Ein Startpunkt, der in einem flachen Tal oder auf einer sanften Neigung liegt, könnte direkt zum höchsten Punkt führen, während ein Startpunkt auf der anderen Seite eines Hügels durch lokale Maxima blockiert werden könnte.
2. **Gradienten und Richtungen:** Die Richtung des steilsten Anstiegs variiert stark je nach Startposition. Kleine Änderungen in der Startposition können zu völlig anderen Pfaden führen, wenn die Gradienten der Funktion zu unterschiedlichen lokalen Maxima zeigen.
3. **Initialer Impuls:** Der anfängliche Schritt in einem Optimierungsprozess kann den Algorithmus auf einen Pfad festlegen, der entweder direkt zum Ziel führt oder in eine Sackgasse mündet. Dies ist besonders kritisch in komplexen Landschaften wie der von **peaks**.

Diese Faktoren unterstreichen, wie wichtig die Wahl des Startpunkts in Optimierungsproblemen ist, insbesondere in realen Anwendungen, wo die Kosten für das Finden des Optimums hoch sein können und ineffiziente Pfade vermieden werden müssen.

*Schön allgemein rausgeredet.*

# 14 Datenverarbeitung

April 12, 2024

## 1 Frage

Erstellen Sie im Matlab-Editor die Datei `kreis.svg` mit dem Inhalt aus dem Skript und verwenden Sie den Befehl `web`, um sie darzustellen.

Was bedeutet `xmlns`? Klappt es auch ohne diese Angabe?

Was verbirgt sich hinter der URL?

Wie könnten Sie den Kreis um 100 Pixel nach rechts verschieben?

Wie könnten Sie seinen Radius auf 150 vergrößern? Was geschieht dann mit seinem oberen Teil?

Was bedeutet `stroke`?

Was bedeutet das Doppelkreuz in den Farbangaben?

Warum können Sie die SVG-Datei nicht direkt aus dem Matlab-Editor heraus anstarten?

### 1.1 Antwort

Um Ihre Fragen zur Verwendung und Manipulation von SVG-Dateien in MATLAB zu beantworten, hier eine detaillierte Erläuterung:

#### 1.1.1 Was bedeutet `xmlns`?

`xmlns` steht für XML Namespace. Dieses Attribut definiert den Namensraum für das SVG-Element und ist essentiell für die korrekte Interpretation des SVG-Codes durch Browser und andere Werkzeuge, die XML verstehen. Wenn das Attribut fehlt, könnte das Dokument nicht korrekt als valide SVG-Datei interpretiert werden.

#### 1.1.2 Kann SVG auch ohne `xmlns` funktionieren?

Technisch gesehen kann eine SVG-Datei auch ohne das `xmlns`-Attribut angezeigt werden, insbesondere wenn sie direkt in HTML-Dokumente eingebettet wird. Allerdings könnte das Fehlen des `xmlns`-Attributs in standalone SVG-Dateien zu Problemen bei der Darstellung führen, insbesondere in strikteren Umgebungen oder älteren Browsern.

#### 1.1.3 Was verbirgt sich hinter der URL in `xmlns`?

Die URL `http://www.w3.org/2000/svg` weist darauf hin, dass es sich um eine SVG-Datei handelt, die gemäß den Standards der World Wide Web Consortium (W3C) definiert ist. Diese URL ist eine

Referenz auf die Definition des SVG-Standards und wird verwendet, um den SVG-Namensraum zu definieren.

#### **1.1.4 Wie können Sie den Kreis um 100 Pixel nach rechts verschieben?**

Um den Kreis in Ihrer SVG-Datei um 100 Pixel nach rechts zu verschieben, erhöhen Sie den Wert des `cx`-Attributs (center x) um 100. Wenn `cx="200"` ist, ändern Sie es zu `cx="300"`.

#### **1.1.5 Wie könnten Sie seinen Radius auf 150 vergrößern? Was geschieht dann mit seinem oberen Teil?**

Um den Radius zu vergrößern, ändern Sie das `r`-Attribut des Kreises. Wenn der Radius von 50 auf 150 erhöht wird, wird der Kreis größer und sein oberer Teil könnte über den oberen Rand des Viewports hinausragen, wenn nicht genügend Platz vorhanden ist. Um das zu vermeiden, sollten Sie auch den `cy`-Wert (center y) anpassen.

#### **1.1.6 Was bedeutet stroke?**

Das `stroke`-Attribut in SVG definiert die Farbe der Linie (Kontur), die den SVG-Formen umgibt. In Ihrem Fall definiert `stroke:#006600` eine dunkelgrüne Kontur um den Kreis.

#### **1.1.7 Was bedeutet das Doppelkreuz in den Farbangaben?**

Das Doppelkreuz `#` in einer Farbangabe wie `#006600` leitet einen hexadezimalen Farbcode ein. Dieser Code bestimmt die Farbe in der Form von RGB (Rot-Grün-Blau) Werten im Hexadezimalformat.

#### **1.1.8 Warum können Sie die SVG-Datei nicht direkt aus dem Matlab-Editor heraus starten?**

MATLABs Editor ist primär für die Bearbeitung und Ausführung von MATLAB-Code konzipiert und bietet keine direkte Unterstützung für das Öffnen oder Rendern von SVG-Dateien. Um eine SVG-Datei aus MATLAB heraus anzuzeigen, können Sie den Befehl `web('kreis.svg')` verwenden, der den Standard-Webbrowser öffnet, um die Datei anzuzeigen.

*Gute, relativ knappe Antworten.*

## **2 Frage**

Laden Sie die Datei `hsb.svg` vom Server, analysieren Sie sie Zeile für Zeile und verwenden Sie wiederum den Befehl `web`, um das Logo in einem neuen Tab des gleichen Browserfenster darzustellen.

Wie lautet der Parameter des `web`-Befehls, um die zweite Datei in einem neuen Tab darzustellen?

Was bewirken die `width`- und `height`-Angaben im `<img>`-Tag?

Wie viele Rechtecke und wie viele Quadrate werden erzeugt?

Durch welche Funktion welches Attributes welches Tags wird das Logo rotiert?

Um wie viel Grad und um welchen Punkt wird das Logo rotiert?

Wozu dient die translate-Funktion?

Was macht das -Tag?

Woher weiß sein fill-Attribut, welchen Farbverlauf es verwenden soll?

Erläutern Sie, wie das -Tag in diesem konkreten Fall funktioniert.

## 2.1 Antwort

Hier ist eine detaillierte Analyse der angegebenen SVG-Datei, sowie die Beantwortung Ihrer Fragen:

### 2.1.1 SVG-Datei-Analyse

- `<svg xmlns="http://www.w3.org/2000/svg" width="190" height="190">`
  - `xmlns`: Definiert den XML-Namensraum für das Dokument, notwendig für SVG-Dateien.
  - `width="190"` und `height="190"`: Setzt die Breite und Höhe des gesamten SVG-Bereichs in Pixeln.
- `<linearGradient id="hsb" x1="0%" y1="0%" x2="100%" y2="0%" gradientUnits="userSpaceOnUse">`
  - Definiert einen linearen Farbverlauf mit der ID "hsb", der horizontal von links nach rechts verläuft.
  - `gradientUnits="userSpaceOnUse"`: Der Farbverlauf wird relativ zum gesamten SVG-Bereich angewendet.
- `<stop offset="0%" stop-color="#003FA5"/>` und `<stop offset="100%" stop-color="#4CF2CC"/>`
  - Definiert die Farben des Verlaufs am Anfang (dunkelblau) und am Ende (türkis).
- `<g fill="url(#hsb)" transform="translate(0 -20) rotate(45 0 70)">`
  - `fill="url(#hsb)"`: Wendet den definierten Farbverlauf auf alle inneren Elemente an.
  - `transform`: Kombiniert eine Translation und eine Rotation für die Gruppe.
    - \* `translate(0 -20)`: Verschiebt alles um 20 Pixel nach oben.
    - \* `rotate(45 0 70)`: Rotiert alles um 45 Grad um den Punkt (0,70).
- `<rect ...>`: Definiert insgesamt fünf Rechtecke mit unterschiedlichen Positionen und Größen.

### 2.1.2 Antworten zu Ihren Fragen

1. **Parameter des web-Befehls für einen neuen Tab:** Um die Datei in einem neuen Tab zu öffnen, verwenden Sie `-new` als Parameter: `web('hsb.svg', '-new')`.
2. **Wirkung der width- und height-Angaben:** Diese definieren die Breite und Höhe des SVG-Canvas, also den Bereich, innerhalb dessen die Grafikelemente gezeichnet werden.
3. **Anzahl der Rechtecke und Quadrate:**
  - Rechtecke: 5
  - Quadrate: Die ersten zwei Rechtecke (jeweils 30x30) sind Quadrate.
4. **Rotation des Logos:**
  - Funktion: `rotate(45 0 70)`
  - Rotiert um 45 Grad um den Punkt (0,70).

5. **Zweck der translate-Funktion:** Verschiebt das Koordinatensystem, hier um 20 Pixel nach oben, was die Positionierung der nachfolgenden Elemente beeinflusst.
6. **Funktion des <g>-Tags:** Gruppirt SVG-Elemente, so dass Transformationen und Stile gemeinsam auf die gesamte Gruppe angewendet werden können.
7. **Farbverlauf im <g>-Tag:** Der fill-Attributwert `url(#hsb)` verweist auf die ID des definierten `<linearGradient>`, womit alle Elemente innerhalb der Gruppe diesen Farbverlauf nutzen.
8. **Funktionsweise des <linearGradient>-Tags im spezifischen Fall:**
  - Definiert einen horizontalen Farbverlauf von Dunkelblau zu Türkis, der über die gesamte Breite des SVG angewendet wird und durch die `gradientUnits` in Bezug auf den SVG-Bereich skaliert wird.

Jetzt werde ich die SVG-Datei mit dem Befehl `web` in einem neuen Tab anzeigen.

Um das SVG-Logo in einem neuen Tab darzustellen, sollten Sie den folgenden MATLAB-Befehl verwenden:

```
web('hsb.svg', '-new')
```

Dieser Befehl öffnet die SVG-Datei `hsb.svg` in einem neuen Tab Ihres Webbrowsers.

*Der INFO-Assistent kann die SVG-Datei nicht lesen, auch wenn wir sie ihm hochladen. Wir müssen ihm den Inhalt der Datei als ASCII-Text mit in die Frage packen.*

### 3 Frage

Laden Sie die Datei `obst.txt` vom Server, analysieren Sie sie Zeile für Zeile und verwenden Sie den Befehl `readtable`, um sie in eine Tabelle einzulesen.

Welches Zeichen müssen Sie dabei als `CommentStyle` angeben, damit die Kommentare ignoriert werden?

Wie stellt Matlab die Tabelle dar?

Welchen internen Datentyp hat die Tabelle?

Wie viele Zeilen und Spalten hat die Tabelle?

Warum werden die ersten Zeilen als Kommentar erkannt, obwohl sie nicht mit dem Kommentarzeichen eingeleitet werden?

Erkennt Matlab die Tabellenspaltenüberschriften? Wie kann das funktionieren?

Was macht Matlab aus der negativen Null?

Warum wird auch die Bananen-Zeile vernünftig eingelesen, obwohl ihre Daten teilweise in der falschen Spalte stehen?

Wie werden deutsche Sonderzeichen interpretiert?

Wie erkennt Matlab, ob es sich bei den Daten um Zahlen oder Zeichenketten handelt? Wie stellt es dies dar?

Wie werden die “e” in den Zahlen der Kirschen- und Weintraubenzeilen interpretiert?

Was bedeutet es, wenn eine Zahl mit einem Punkt beginnt?

### 3.1 Antwort

Um die Fragen zur Datei `obst.txt` und deren Behandlung in MATLAB zu beantworten, gehe ich wie folgt vor:

#### 3.1.1 Dateianalyse und MATLAB-Kommandos

Die Datei `obst.txt` enthält Daten und Kommentare. Die ersten zwei Zeilen sind nicht mit einem typischen Kommentarzeichen versehen, scheinen aber dennoch ignoriert zu werden, vermutlich wegen ihrer Struktur und Formatierung. Die Kopfzeile enthält die Spaltenbezeichnungen, und danach folgen die Daten. Kommentare am Ende sind durch Prozentzeichen (%) gekennzeichnet.

1. **CommentStyle für das Einlesen:** Der `CommentStyle` für diese Datei sollte das Prozentzeichen (%) sein, da es zur Kennzeichnung von Kommentaren verwendet wird.
2. **MATLAB Darstellung der Tabelle:** MATLAB würde diese Daten als Tabelle (`table`) darstellen, wobei die erste gültige Zeile nach den ignorierten Zeilen als Spaltenüberschriften erkannt wird, falls sie die richtige Formatierung aufweist.
3. **Interner Datentyp:** Der interne Datentyp dieser Tabelle in MATLAB ist `table`.
4. **Zeilen und Spalten:** Die Tabelle hat 7 Zeilen (für jedes Obst eine Zeile) und 3 Spalten (`t`, `x`, `Name`).
5. **Erkennung von Kommentaren:** Die ersten Zeilen werden als Kommentar erkannt, wenn sie nicht den erwarteten Datenformatierungen entsprechen, etwa durch Fehlen des Trennzeichens oder durch das Format, das nicht zu den anderen Datenzeilen passt.
6. **Erkennung von Tabellenspaltenüberschriften:** MATLAB erkennt die Zeile, die auf die ignorierten Zeilen folgt, als Kopfzeile, wenn diese strukturell von den Datenzeilen abweicht und korrekt formatiert ist (z.B. durch Tabulator getrennt).
7. **Behandlung der negativen Null:** In MATLAB wird `-0.0` wie `0.0` behandelt; numerisch gibt es keinen Unterschied zwischen diesen Werten.
8. **Eingabe der Bananen-Zeile:** MATLAB liest die Bananen-Zeile korrekt ein, auch wenn Daten verschoben sind, da es auf das Format der Spalten reagiert und fehlende Daten entsprechend als fehlende Werte (`NaN`) handhabt.
9. **Interpretation deutscher Sonderzeichen:** Deutsche Sonderzeichen wie Ä sollten korrekt gelesen werden, wenn die Dateikodierung korrekt ist und MATLAB diese unterstützt.
10. **Dateninterpretation (Zahlen vs. Zeichenketten):** MATLAB bestimmt den Datentyp basierend auf dem Inhalt jeder Spalte. Numerische Daten werden als solche gelesen, während Text in Anführungszeichen oder offensichtlich nicht numerische Daten als Zeichenketten interpretiert werden.
11. **Interpretation der “e” in Zahlen:** Das “e” in den Zahlen wird als Teil der wissenschaftlichen Notation interpretiert (z.B. `2.525e1` für  $(2.525 \times 10^1)$ ).

12. **Bedeutung eines Punktes am Anfang einer Zahl:** Eine Zahl, die mit einem Punkt beginnt (`.3636e2`), wird als  $0.3636 \times 10^2$  interpretiert.

### 3.1.2 Beispiel für den MATLAB-Befehl `readtable`

```
opts = detectImportOptions('obst.txt', 'CommentStyle', '%');  
T = readtable('obst.txt', opts);
```

Dieser Befehl liest die `obst.txt` unter Berücksichtigung von Kommentaren, die mit `%` beginnen, und zeigt die Daten in einer Tabelle an.

*Auch hier haben wir den Text der Datei an die Frage angehängt.*

## 4 Frage

Wenden Sie den Befehl `summary` auf die Tabelle an.

Beschreiben Sie, was der Befehl darstellt.

### 4.1 Antwort

Der Befehl `summary` in MATLAB wird verwendet, um eine Zusammenfassung der statistischen Eigenschaften einer Tabelle oder eines Datenarrays zu geben. Wenn Sie `summary` auf eine Tabelle anwenden, erhalten Sie folgende Informationen für jede Spalte in der Tabelle:

1. **Datentyp:** Zeigt den Typ der Daten in der Spalte an (z.B. `double`, `categorical`, `string`).
2. **Anzahl der fehlenden Werte:** Gibt an, wie viele Einträge in der Spalte fehlen (`NaN` für numerische Daten oder leere Zellen für Strings).
3. **Anzahl der Nicht-fehlenden Werte:** Gibt an, wie viele Einträge in der Spalte gültige Daten enthalten.
4. **Zusammenfassung der Inhalte:**
  - Für numerische Daten: Statistische Maßzahlen wie Mittelwert, Median, Minimum, Maximum und Quartile.
  - Für kategoriale oder textbasierte Daten: Anzahl der eindeutigen Kategorien oder eine Liste der häufigsten Kategorien.

#### 4.1.1 Anwendungsbeispiel

Angenommen, Sie haben die Tabelle `T` aus der `obst.txt` erfolgreich geladen und sie enthält drei Spalten: `t`, `x` und `Name`. Der `summary`-Befehl würde folgendermaßen angewendet:

```
summary(T)
```

#### 4.1.2 Mögliche Ausgabe des `summary`-Befehls:

- `t`:
  - **Datentyp:** `double`
  - **Nicht-fehlende Werte:** 7
  - **Statistiken:** `min`, `max`, `median`, `mean` etc.
- `x`:
  - **Datentyp:** `double`

- **Nicht-fehlende Werte:** 7 (wenn keine fehlenden Werte vorhanden sind; falls Bananen fehlende Daten haben, würde es 6 sein)
- **Statistiken:** min, max, median, mean etc.
- **Name:**
  - **Datentyp:** string oder categorical (abhängig von der Eingabe)
  - **Nicht-fehlende Werte:** 7
  - **Anzahl der eindeutigen Kategorien:** 7 (wenn alle Namen eindeutig sind)

Diese Zusammenfassung bietet eine schnelle Übersicht über die Datenstruktur, die in der Tabelle enthalten ist, und hilft bei der Überprüfung der Datenqualität und -vollständigkeit.

*Die Anzahl der “Nicht-fehlenden Werte” wird offensichtlich nicht angezeigt; laut Hilfe gibt es nur “NumMissing”, die allerdings nicht in der Gesamtübersicht.*

## 5 Frage

Verwenden Sie den `plot`-Befehl, um die `x`-Werte über den `t`-Werten zu plotten.

Wie können Sie erreichen, dass die Datenpaare als rote Kreise auf einer roten Linie dargestellt werden?

### 5.1 Antwort

Um die Daten in MATLAB zu plotten, wobei die `x`-Werte über den `t`-Werten mit roten Kreisen auf einer roten Linie dargestellt werden, können Sie den `plot`-Befehl mit spezifischen Optionen für Marker und Liniestil verwenden. Hier ist ein einfaches Beispiel, wie Sie das erreichen können, vorausgesetzt, Ihre Tabelle heißt `T` und die entsprechenden Spalten sind `t` und `x`:

#### 5.1.1 MATLAB-Plot-Befehl

```
plot(T.t, T.x, 'ro-') % 'ro-' bedeutet rote Linie ('r'), Kreis-Marker ('o') und Linie verbunden
```

#### Detaillierte Erklärung:

- **T.t und T.x:** Dies sind die Vektoren der `t`- und `x`-Spalten aus Ihrer Tabelle `T`. Sie geben die `x`- und `y`-Koordinaten für den Plot an.
- **'ro-':**
  - `'r'` steht für die Farbe Rot.
  - `'o'` ist der Marker-Typ Kreis.
  - `'-'` verbindet die Punkte mit einer Linie.

#### 5.1.2 Zusätzliche Anpassungen

Wenn Sie weitere Anpassungen an der Darstellung vornehmen möchten, wie z.B. das Ändern der Markergröße oder der Linienbreite, können Sie zusätzliche Plot-Parameter verwenden:

```
plot(T.t, T.x, 'ro-', 'MarkerSize', 8, 'LineWidth', 2) % Größere Marker und dickere Linie
```

#### Erweiterte Darstellungsoptionen:

- **'MarkerSize', 8:** Legt die Größe der Marker auf 8 Punkte fest.

- `'LineWidth', 2`: Setzt die Breite der Linie auf 2 Punkte.

### 5.1.3 Darstellung bestätigen

Nachdem Sie den Plot-Befehl ausgeführt haben, erscheint ein Grafikfenster, das die `x`-Werte als Funktion der `t`-Werte zeigt, wobei jedes Datenpaar durch einen roten Kreis markiert und die Punkte mit einer roten Linie verbunden sind. Dies hilft bei der visuellen Analyse der Daten und zeigt Trends oder Muster auf.

### 5.1.4 Fazit

Durch die Verwendung von `plot` mit den oben genannten Optionen in MATLAB können Sie eine effektive visuelle Darstellung Ihrer Daten erreichen, die nicht nur informativ, sondern auch ästhetisch ansprechend ist.

*Bisschen ausführlich die Antwort, oder?*

## 6 Frage

Verwenden Sie den Befehl `pie3`, um die `x`-Werte als Tortendiagramm mit den zugeordneten Namen in einem darzustellen.

Werden die Äpfel angezeigt?

### 6.1 Antwort

Um ein 3D-Tortendiagramm (`pie3`) in MATLAB zu erstellen, das die `x`-Werte aus Ihrer Tabelle `T` verwendet und die zugehörigen Namen als Beschriftungen anzeigt, können Sie den `pie3`-Befehl verwenden. Hier ist ein Beispiel, wie Sie dies tun könnten, unter der Annahme, dass Ihre Tabelle `T` die Spalten `x` und `Name` enthält:

#### 6.1.1 MATLAB `pie3`-Befehl

```
pie3(T.x, ones(size(T.x)), T.Name) % Verwendet die 'x'-Werte für die Segmente und 'Name' für
```

#### 6.1.2 Erklärung der Befehlsparameter:

- `T.x`: Dies sind die `x`-Werte aus Ihrer Tabelle, die die Größe jedes Tortenstückes bestimmen.
- `ones(size(T.x))`: Erzeugt ein Array von Einsen der gleichen Größe wie `T.x`, was bedeutet, dass keine automatische Skalierung der Segmente basierend auf den `x`-Werten stattfindet.
- `T.Name`: Dies sind die Namen des Obstes, die als Labels für jedes Tortenstück verwendet werden.

#### 6.1.3 Anzeige der Äpfel

Da der `x`-Wert für Äpfel `-0.0` ist, wird dieses Segment im `pie3`-Diagramm möglicherweise nicht sichtbar sein, wenn die anderen Werte wesentlich größer sind. In MATLAB und den meisten anderen numerischen Systemen wird `-0.0` als `0.0` behandelt, und ein Wert von `0` in einem Tortendiagramm führt dazu, dass kein sichtbares Segment für diesen Eintrag erstellt wird.

### 6.1.4 Überprüfung der Sichtbarkeit

Um sicherzustellen, dass alle Einträge, einschließlich der Äpfel, im Tortendiagramm sichtbar sind, sollten Sie überprüfen, ob alle Werte positiv und signifikant sind im Vergleich zu anderen Werten. In diesem speziellen Fall, wo der Wert 0.0 ist, wird kein Segment für Äpfel angezeigt.

### 6.1.5 Alternative Vorgehensweise

Wenn Sie möchten, dass auch Werte, die 0 sind, im Diagramm dargestellt werden, könnten Sie einen kleinen positiven Wert zu allen Werten hinzufügen oder spezifisch zum Wert der Äpfel, um sicherzustellen, dass auch sie im Diagramm erscheinen:

```
T.x_adj = T.x + 0.1; % Fügt jedem Wert ein wenig hinzu, um sicherzustellen, dass alle sichtbar  
pie3(T.x_adj, ones(size(T.x_adj)), T.Name)
```

Diese Methode stellt sicher, dass auch die Äpfel im Diagramm erscheinen, allerdings könnte dies die tatsächlichen Verhältnisse der Daten verzerren. Wählen Sie die Methode basierend auf Ihrem spezifischen Bedarf an genauer Datenrepräsentation.

*Da gibt es gleich mehrere Fehler:*

*ones(size(T.x)):* Erzeugt ein Array von Einsen der gleichen Größe wie  $T.x$ , was bedeutet, dass keine automatische Skalierung der Segmente basierend auf den  $x$ -Werten stattfindet. In Wirklichkeit bewirkt dieser zusätzliche Parameter, dass alle Sektoren explodiert (also mit Abstand voneinander) dargestellt werden.

*In diesem speziellen Fall, wo der Wert 0.0 ist, wird kein Segment für Äpfel angezeigt. Doch, das Segment wird mit seinen Rändern dargestellt. Seine Breite (Fläche) ist natürlich null.*

## 7 Frage

Verwenden Sie den Befehl `sortrows`, um nach der dritten Spalte zu sortieren und weisen Sie die sortierte Tabelle der Variablen `t` zu.

Werden auch die ersten beiden Spalten mitsortiert?

Warum sieht der Plot der sortierten Tabelle so merkwürdig aus?

Wie sieht es mit dem Tortendiagramm aus?

### 7.1 Antwort

Um eine Tabelle in MATLAB nach der dritten Spalte (in Ihrem Fall wahrscheinlich die Spalte `Name`) zu sortieren, verwenden Sie den Befehl `sortrows`. Dieser Befehl sortiert die Tabelle basierend auf den Werten einer oder mehrerer Spalten und behält dabei die Zeilenbindung zwischen den Spalten bei. Das bedeutet, dass alle Spalten mitsortiert werden, jedoch in einer Art und Weise, dass die ursprünglichen Zusammenhänge zwischen den Werten innerhalb einer Zeile erhalten bleiben.

#### 7.1.1 Sortierung der Tabelle

```
t = sortrows(T, 3); % Sortiert die Tabelle T nach der dritten Spalte 'Name'
```

### 7.1.2 Antworten auf Ihre Fragen

1. **Werden auch die ersten beiden Spalten mitsortiert?** Ja, die gesamte Zeile wird als Einheit behandelt und entsprechend der Werte in der dritten Spalte sortiert. Das bedeutet, dass die Zuordnungen der Werte in den Spalten `t` und `x` zu ihren entsprechenden Namen in der Spalte `Name` unverändert bleiben, aber die gesamte Zeilenreihenfolge ändert sich, um eine aufsteigende (oder absteigende) Reihenfolge der Namen zu erreichen.
2. **Warum sieht der Plot der sortierten Tabelle so merkwürdig aus?** Wenn Sie die Tabelle nach dem Namen sortieren und dann ein Diagramm der `x`-Werte über den `t`-Werten plotten, kann der Plot unerwartet aussehen, weil die Reihenfolge der `t`-Werte nicht mehr der ursprünglich aufsteigenden oder logischen Reihenfolge folgt. Stattdessen folgt sie der alphabetischen Reihenfolge der Namen. Dies kann dazu führen, dass der Plot nicht mehr die reale zeitliche oder sequenzielle Entwicklung darstellt, sondern eher eine zufällige Anordnung aufweist, die durch die Sortierung nach Namen entsteht.
3. **Wie sieht es mit dem Tortendiagramm aus?** Das Tortendiagramm, das die `x`-Werte darstellt, ist von der Sortierung weniger betroffen in Bezug auf die visuelle Darstellung, da es sich um ein kategoriengestütztes Diagramm handelt und nicht um ein sequenzielles Diagramm wie ein Linien- oder Streudiagramm. Die Sektoren des Tortendiagramms werden einfach in der neuen Reihenfolge der Namen angezeigt, mit möglichen Änderungen in der Anordnung der Sektoren, jedoch ohne die visuelle Interpretation zu beeinträchtigen. Die Größe der Sektoren basiert weiterhin auf den `x`-Werten, und die Beschriftungen sind entsprechend der sortierten Namen.

### 7.1.3 Beispielcode für Plot und Tortendiagramm nach Sortierung

```
figure;  
plot(t.t, t.x, 'ro-'); % Plot der sortierten Werte kann "merkwürdig" aussehen  
title('Plot der sortierten Werte');
```

```
figure;  
pie3(t.x); % Tortendiagramm der sortierten Werte  
title('Tortendiagramm der sortierten Werte');
```

Durch diese Visualisierungen können Sie untersuchen, wie die Daten nach der Sortierung dargestellt werden, und es ermöglicht eine bessere Interpretation der Auswirkungen der Sortierung auf verschiedene Datendarstellungen.

*Gute Antworten*

## 8 Frage

Erzeugen Sie durch Indexierung eine Teiltabelle, die die ersten beiden Spalten der zweiten bis zur vierten Zeile enthält.

Wie können Sie geschweifte Klammern verwenden, um die eigentlichen Daten der Teiltabelle einer Matrix zuzuweisen?

## 8.1 Antwort

Um eine Teiltabelle in MATLAB zu erstellen, die nur die ersten beiden Spalten und bestimmte Zeilen (in diesem Fall die zweite bis vierte Zeile) aus einer vorhandenen Tabelle `T` umfasst, können Sie die Standardindexierungsfunktionen von MATLAB verwenden. Hierbei ist es wichtig zu verstehen, dass Tabellen in MATLAB flexibel in Bezug auf Indexierung sind und sowohl numerische als auch benannte Indizes unterstützen.

### 8.1.1 Erstellen der Teiltabelle

Um die gewünschte Teiltabelle zu erstellen, können Sie folgenden Code verwenden:

```
TeilT = T(2:4, 1:2); % Selektiert die Zeilen 2 bis 4 und die Spalten 1 und 2
```

Hierbei gibt `2:4` den Bereich der Zeilenindizes an und `1:2` den Bereich der Spaltenindizes. Dieser Befehl erstellt eine neue Tabelle `TeilT`, die nur die angegebenen Zeilen und Spalten enthält.

### 8.1.2 Zuweisung der eigentlichen Daten zu einer Matrix mit geschweiften Klammern

Um die tatsächlichen Datenwerte (nicht die Tabellenstruktur) einer Matrix zuzuweisen, verwenden Sie geschweifte Klammern `{}`. Dies ermöglicht den Zugriff auf die Daten innerhalb der Tabelle als Array, wobei die Tabelle quasi “entpackt” wird. Dies ist besonders nützlich, wenn Sie Operationen durchführen möchten, die spezifisch für numerische Arrays sind.

**Beispielcode für die Zuweisung mit geschweiften Klammern:**

```
MatrixDaten = TeilT{: , :}; % Konvertiert die Teiltabelle in eine numerische Matrix
```

Hierbei bewirkt `TeilT{: , :}`: `- :` für alle Zeilen. `- :` für alle Spalten.

Das Ergebnis ist, dass `MatrixDaten` eine numerische Matrix wird, die alle Daten aus den gewählten Zeilen und Spalten der ursprünglichen Tabelle `TeilT` enthält. Diese Matrix kann dann für weitere Berechnungen oder Analysen verwendet werden, die außerhalb des Tabellenformats durchgeführt werden sollen.

### 8.1.3 Zusammenfassung

Durch die Verwendung von geschweiften Klammern `{}` in MATLAB können Sie die eigentlichen Daten einer Tabelle extrahieren und diese Daten einer Matrix zuweisen. Dies ermöglicht eine direkte Manipulation und Verarbeitung der Daten, die mit herkömmlichen Array-Operationen kompatibel sind.

*So ist's richtig.*

## 9 Frage

Laden Sie die Datei `filme.xml` vom Server und analysieren Sie ihren Inhalt im Matlab-Editor.

Ist Ihnen aufgefallen, dass der Matlab-Editor auch XML-Dateien freundlicherweise farbig “syntax-highlightet”?

Wozu dient die erste Zeile der XML-Datei?

Beschreiben Sie, wie Elemente in XML grundsätzlich aufgebaut sind.

Wie werden Kommentare in XML gekennzeichnet?

Welchen Namen hat das Wurzelement? Ist dieser Name frei wählbar?

Beschreiben Sie, wie die Attribute in XML aufgebaut sind.

Wie können Sie spitze Klammern als Elementtext verwenden?

Wie viele Filme befinden sich in der XML-Datei?

Gibt es Elemente, die nicht in allen Filmen vorhanden sind?

War Ihnen klar, dass Hugo Weaving sowohl in Matrix als auch in Herr der Ringe mitgespielt hat?

## 9.1 Antwort

Die `filme.xml` Datei, die Sie bereitgestellt haben, bietet verschiedene Aspekte von XML, die untersucht werden können. Hier sind Antworten auf Ihre Fragen basierend auf dem Inhalt und der Struktur der Datei:

### 9.1.1 1. Syntax-Highlighting in MATLAB

Ja, MATLAB bietet freundlicherweise Syntax-Highlighting für XML-Dateien im Editor, was das Lesen und Bearbeiten von XML-Dateien erleichtert, indem es verschiedene Elemente wie Tags, Attribute und Kommentare in unterschiedlichen Farben darstellt.

### 9.1.2 2. Zweck der ersten Zeile in der XML-Datei

Die erste Zeile der XML-Datei, bekannt als XML-Deklaration, gibt die XML-Version und die Kodierung der Datei an. Hier ist sie:

```
<?xml version="1.0" encoding="utf-8"?>
```

Diese Deklaration informiert den Parser, dass es sich um eine XML-Version 1.0 handelt und dass die Kodierung der Datei UTF-8 ist. Dies ist wichtig für die korrekte Interpretation der Zeichen, insbesondere wenn Sonderzeichen oder nicht-lateinische Alphabete verwendet werden.

### 9.1.3 3. Aufbau von Elementen in XML

XML-Elemente sind grundsätzlich aus folgenden Teilen aufgebaut: - **Start-Tag**: Beginnt mit `<` gefolgt vom Elementnamen und endet mit `>`. - **Inhalt**: Der Text oder andere Elemente, die innerhalb dieses Tags enthalten sind. - **End-Tag**: Beginnt mit `</`, gefolgt vom gleichen Elementnamen wie im Start-Tag und endet mit `>`.

Beispiel:

```
<film>
  <!-- Inhalt hier -->
</film>
```

#### 9.1.4 4. Kommentare in XML

Kommentare in XML werden durch `<!--` eingeleitet und durch `-->` beendet. Beispiel:

```
<!-- Dies ist ein Kommentar ... -->
```

#### 9.1.5 5. Name des Wurzelements

Das Wurzelement in der bereitgestellten XML-Datei ist `<filme>`. Der Name des Wurzelements ist frei wählbar und sollte den Inhalt der XML-Datei repräsentieren.

#### 9.1.6 6. Aufbau von Attributen in XML

Attribute in XML bieten zusätzliche Informationen über Elemente und sind immer im Start-Tag des Elements enthalten. Sie bestehen aus einem Namen, gefolgt von einem Gleichheitszeichen und einem Wert in Anführungszeichen. Beispiel:

```
<filme info='Wurzelement' datum='04.04.2018'>
```

#### 9.1.7 7. Verwendung von spitzen Klammern im Elementtext

Um spitze Klammern innerhalb von Texten in XML zu verwenden, müssen sie als Entitäten kodiert werden: `&lt;` für `<` und `&gt;` für `>`. Alternativ können Daten in CDATA-Sektionen eingeschlossen werden, was verhindert, dass sie als XML-Markup interpretiert werden:

```
<![CDATA[<spitze> Klammern?]]>
```

#### 9.1.8 8. Anzahl der Filme in der XML-Datei

Die XML-Datei enthält acht `<film>` Elemente, also acht Filme.

#### 9.1.9 9. Elemente, die nicht in allen Filmen vorhanden sind

Einige Filme haben das Element `<untertitel>`, was in anderen Filmen wie “Matrix” oder “Kill Bill” fehlt. Dies zeigt, dass nicht alle Elemente in jedem Film vorhanden sein müssen.

#### 9.1.10 10. Hugo Weaving in Matrix und Herr der Ringe

Ja, Hugo Weaving spielt sowohl in “Matrix” als auch in der “Herr der Ringe”-Reihe mit, was interessante Querverbindungen zwischen den Filmen darstellt und zeigt, wie XML genutzt werden kann, um solche Informationen strukturiert darzustellen.

*Sehr schöne, hilfreiche Analyse!*

## 10 Frage

Laden Sie die Datei `xml2struct` vom MatlabCentral-Server herunter und verwenden Sie sie, um die Datei `filme.xml` in die Variable `daten` einzulesen.

Von welchem Datentyp ist die Variable `daten` dann? Welche Felder hat die Variable? Wird der Kommentar mit eingelesen?

## 10.1 Antwort

Die Anweisung, eine Datei von MatlabCentral herunterzuladen und die XML-Datei `filme.xml` mithilfe der `xml2struct`-Funktion einzulesen, bezieht sich auf einen typischen Workflow in MATLAB, wenn man mit XML-Dateien arbeitet. Leider kann ich keine Dateien direkt von externen Quellen herunterladen oder echten Code in einer externen MATLAB-Umgebung ausführen. Jedoch kann ich Ihnen erklären, wie Sie diesen Vorgang durchführen können und was zu erwarten ist.

### 10.1.1 Schritte zum Herunterladen und Verwenden von `xml2struct`:

#### 1. Herunterladen der Funktion `xml2struct`:

- Gehen Sie zur [MatlabCentral FileExchange](#) Webseite.
- Suchen Sie nach `xml2struct` und laden Sie die Funktion herunter.
- Speichern Sie die Funktion in einem Verzeichnis, das von MATLAB aus zugänglich ist.

#### 2. Einlesen der XML-Datei mit `xml2struct`:

- Stellen Sie sicher, dass die `filme.xml` Datei in einem verfügbaren Verzeichnis gespeichert ist.
- Verwenden Sie den folgenden MATLAB-Code, um die XML-Datei einzulesen:

```
daten = xml2struct('filme.xml');
```

### 10.1.2 Datentyp und Felder der Variable `daten`

Nach dem Einlesen der XML-Datei ist die Variable `daten` ein Strukturarray (`struct`), das die Struktur der XML-Datei widerspiegelt. Dies umfasst:

- **Felder**, die Elementen und Attributen in der XML-Datei entsprechen. Jedes XML-Element wird zu einem Feld in der Struktur, und die Attribute und Textinhalte werden entsprechend als Unterfelder abgebildet.
- **Datentyp**: `struct`.

### 10.1.3 Kommentare in der XML-Datei

Ob der Kommentar in der XML-Datei eingelesen wird, hängt von der spezifischen Implementierung der `xml2struct`-Funktion ab. Viele `xml2struct`-Implementierungen ignorieren Kommentare, weil sie nicht zum Dateninhalt beitragen und in der Regel nicht notwendig sind, um die Datenstruktur zu verstehen oder die Daten weiterzuverarbeiten. Wenn die Funktion jedoch Kommentare einschließt, könnten diese in einem separaten Feld oder speziell gekennzeichnet in der Struktur gespeichert sein.

### 10.1.4 Überprüfung der Felder in `daten`

Um die Felder in der Struktur `daten` zu überprüfen und zu sehen, ob Kommentare eingeschlossen sind, können Sie folgenden MATLAB-Befehl verwenden:

```
disp(fieldnames(daten));
```

### 10.1.5 Zusammenfassung

Sofern Sie `xml2struct` korrekt aus MatlabCentral heruntergeladen und in Ihrem MATLAB-Workspace implementiert haben, sollten Sie in der Lage sein, die XML-Daten effektiv in eine MATLAB-Struktur zu konvertieren und zu analysieren. Überprüfen Sie die Dokumentation oder

den Hilfetext der heruntergeladenen Funktion, um spezifische Details zum Umgang mit Kommentaren und anderen speziellen XML-Features zu verstehen.

*Es ist immer wieder faszinierend, wie sie - ohne die Aufgabe wirklich durchzuführen - trotzdem präzise Antworten geben kann.*

## 11 Frage

Sprechen Sie das Wurzelement an.

Welche drei Felder hat das Wurzelement?

### 11.1 Antwort

Um das Wurzelement in der Struktur, die aus einer XML-Datei mittels `xml2struct` erstellt wurde, anzusprechen, können Sie direkt auf das Hauptfeld der Struktur zugreifen. In Ihrem Fall, wenn die XML-Datei ein Wurzelement `<filme>` hat, wird es in MATLAB als ein Hauptfeld der Struktur `daten` repräsentiert.

#### 11.1.1 Zugriff auf das Wurzelement

Das Wurzelement und seine Felder können wie folgt angesprochen werden:

```
wurzel = daten.filme;
```

#### 11.1.2 Felder des Wurzelements

Die Felder innerhalb des Wurzelements hängen von den Attributen und Unterelementen ab, die in der XML-Datei definiert sind. Typischerweise würde die `xml2struct`-Funktion die Attribute des Wurzelements und alle enthaltenen Kind-Elemente in Unterstrukturen umwandeln.

Basierend auf Ihrer XML-Dateistruktur hat das Wurzelement `<filme>` die folgenden potenziellen Felder:

1. **Attribute:** Attribute des Wurzelements, wie `info` und `datum`, die als Unterfelder der Struktur abgelegt werden könnten. Diese Attribute sind im Start-Tag des Wurzelements definiert.

```
info = daten.filme.Attributes.info;  
datum = daten.filme.Attributes.datum;
```

2. **Kind-Elemente:** Jedes `<film>`-Element wird wahrscheinlich als Teil eines Array von Strukturen oder als separate Felder innerhalb der `filme`-Struktur abgebildet. Jedes dieser Elemente würde weitere Unterfelder für seine eigenen Attribute und Kind-Elemente haben (z.B. `<titel>`, `<jahr>`, `<regie>`, `<besetzung>`).

3. **Kommentare und Spezialtexte:** Wenn die `xml2struct`-Funktion Kommentare einliest, könnten diese auch als ein spezielles Feld gespeichert werden, obwohl dies weniger üblich ist.

#### 11.1.3 Beispiel zur Überprüfung der Felder

Um die genauen Felder des Wurzelements zu überprüfen, können Sie folgenden MATLAB-Code verwenden, der alle obersten Felder des `daten.filme`-Elements auflistet:

```
disp(fieldnames(daten.filme));
```

Dieser Befehl gibt eine Liste aller Felder aus, die im Wurzelement `filme` enthalten sind, und gibt Ihnen damit Aufschluss darüber, wie die XML-Struktur in MATLAB abgebildet wurde.

#### 11.1.4 Zusammenfassung

Durch den Zugriff auf `daten.filme` können Sie das Wurzelement Ihrer XML-Struktur ansprechen. Die drei Hauptfelder sind typischerweise die Attribute, die Kind-Elemente (wie die Filme selbst) und möglicherweise spezielle Felder für Kommentare oder CDATA-Sektionen, abhängig von der spezifischen Implementierung von `xml2struct`.

*Faszinierend, wie gut die "vermuteten" Antworten sind! Sie übersieht allerdings das Feld `boese_zeichen`.*

## 12 Frage

Sprechen Sie das `Attributes`-Feld und dessen `datum`-Feld an.

Als welchen Datentyp gibt `xml2struct` das Datum aus?

### 12.1 Antwort

Um auf das `Attributes`-Feld und speziell auf das `datum`-Feld im Wurzelement der mittels `xml2struct` erstellten Struktur zuzugreifen, verwenden Sie den Zugriffspfad, der durch die Struktur der XML-Datei bestimmt wird. Hier ist, wie Sie darauf zugreifen könnten:

#### 12.1.1 Zugriff auf das `Attributes`-Feld und das `Datum`-Feld

Angenommen, Sie haben die XML-Datei in eine MATLAB-Struktur namens `daten` eingelesen, und das Wurzelement ist `filme`. Der Zugriff auf das `datum`-Attribut würde folgendermaßen aussehen:

```
datum = daten.filme.Attributes.datum;
```

Dieser Code greift auf das `Attributes`-Feld des Wurzelements `filme` zu und extrahiert den Wert des `datum`-Attributes.

#### 12.1.2 Datentyp des `Datum`-Feldes

Der Datentyp des `datum`-Feldes, das von der Funktion `xml2struct` ausgegeben wird, ist typischerweise ein Zeichenkette (`char` oder `string`), da XML-Attribute als Text in den Tags gespeichert sind und `xml2struct` diese Textdaten standardmäßig als Zeichenketten in MATLAB konvertiert.

#### 12.1.3 Beispielcode

Um den Datentyp des `datum`-Feldes zu überprüfen, können Sie den `class`-Befehl verwenden:

```
datum_typ = class(datum);  
disp(['Der Datentyp des Datum-Feldes ist: ', datum_typ]);
```

Dieser Befehl gibt den Datentyp der Variable `datum` aus, die den Wert des `datum`-Attributes speichert. Es wird erwartet, dass das Ergebnis `char` oder `string` ist, abhängig davon, ob Ihre

MATLAB-Version und die spezifische `xml2struct`-Implementierung Zeichenketten standardmäßig als `char`-Arrays oder als `string`-Objekte behandeln.

#### 12.1.4 Zusammenfassung

Die Funktion `xml2struct` konvertiert XML-Attribute standardmäßig in MATLAB-Zeichenketten, da XML-Attribute im XML-Dokument als Text repräsentiert werden. Der Zugriff auf diese Attribute durch die von `xml2struct` erstellte Struktur ermöglicht es Ihnen, die in den Attributen gespeicherten Daten direkt zu verwenden oder weiter zu verarbeiten.

*Alles sehr gut.*

## 13 Frage

Sprechen Sie `boese_zeichen` und sein einziges Feld an.

Wie nennt `xml2struct` das Feld von `boese_zeichen`? Werden die spitzen Klammern korrekt ausgegeben?

### 13.1 Antwort

Um auf das Element `boese_zeichen` und sein einziges Feld aus einer durch `xml2struct` konvertierten XML-Struktur zuzugreifen, gehen wir davon aus, dass Sie die XML-Datei bereits in eine Strukturvariable namens `daten` geladen haben. Im Element `boese_zeichen` wird spezieller Text mithilfe der CDATA-Sektion eingefügt, die dazu dient, Textdaten einzuschließen, die sonst als XML-Markup interpretiert werden könnten.

#### 13.1.1 Zugriff auf das Element `boese_zeichen`

Der Zugriff auf das Element `boese_zeichen` und sein Inhalt erfolgt typischerweise so:

```
boese_zeichen_content = daten.filme.boese_zeichen.Text;
```

#### 13.1.2 Feldname von `boese_zeichen` in der Struktur

Das Feld, das den Inhalt von `boese_zeichen` enthält, wird typischerweise `Text` genannt, wenn Sie `xml2struct` verwenden. Dies hängt von der Implementierung von `xml2struct` ab. Einige Versionen könnten auch andere Feldnamen verwenden, je nachdem, wie die Funktion programmiert wurde, um Elemente und ihre Textinhalte zu handhaben.

#### 13.1.3 Ausgabe der spitzen Klammern

Wenn der Text in `boese_zeichen` in einer CDATA-Sektion eingeschlossen ist (`<![CDATA[<spitze> Klammern?]]>`), sollte `xml2struct` den Inhalt zwischen den CDATA-Tags ohne Interpretation oder Änderung erfassen. Das bedeutet, dass die spitzen Klammern als Teil des Textes erhalten bleiben und korrekt ausgegeben werden sollten. Sie werden nicht als XML-Markup behandelt, sondern als reiner Text.

### 13.1.4 Überprüfung der Ausgabe

Um zu überprüfen, ob die spitzen Klammern korrekt ausgegeben werden, können Sie einfach den Inhalt von `boese_zeichen_content` in MATLAB anzeigen lassen:

```
disp(['Inhalt von boese_zeichen: ', boese_zeichen_content]);
```

### 13.1.5 Zusammenfassung

Der Zugriff auf Text, der innerhalb einer CDATA-Sektion in einer XML-Datei steht, über eine mittels `xml2struct` konvertierte Struktur, sollte den Text exakt so wiedergeben, wie er in der Originaldatei steht, einschließlich aller Sonderzeichen wie spitze Klammern. Die CDATA-Sektion wird genutzt, um sicherzustellen, dass die Daten innerhalb ihrer Grenzen nicht als XML-Elemente oder Attribute interpretiert werden.

*Ja, nur dass unsere `xml2struct` nicht “`daten.filme.boese_zeichen.Text`” sondern “`daten.filme.boese_zeichen.CDATA`” verwendet. Aber das kann sie natürlich nicht wissen.*

## 14 Frage

Sprechen Sie den ersten Film an.

Warum müssen Sie dabei den Filmindex in geschweifte Klammern setzen?

Wie formatiert `xml2struct` die Tatsache, dass es mehrere Regisseur\_innen und mehrere Schauspieler\_innen gibt?

### 14.1 Antwort

Um den ersten Film in der Struktur, die von `xml2struct` erzeugt wurde, anzusprechen, müssen Sie verstehen, wie `xml2struct` Listen von Elementen behandelt, die im XML mehrfach auftreten können, wie etwa die Filme in Ihrer XML-Datei.

#### 14.1.1 Zugriff auf den ersten Film

Wenn Sie die XML-Daten mit `xml2struct` eingelesen haben, werden Elemente, die mehrfach auftreten, typischerweise in einem Zellenarray gespeichert. Um auf den ersten Film zuzugreifen, verwenden Sie den Index 1 innerhalb geschweiften Klammern, weil in MATLAB geschweifte Klammern verwendet werden, um auf den Inhalt von Zellenarrays zuzugreifen. Hier ist, wie Sie darauf zugreifen könnten:

```
erster_film = daten.filme.film{1};
```

#### 14.1.2 Warum geschweifte Klammern?

Geschweifte Klammern (`{}`) in MATLAB werden benutzt, um auf einzelne Elemente innerhalb eines Zellenarrays zuzugreifen. Da `xml2struct` jede Instanz von wiederholten XML-Elementen (wie `film`) in einem separaten Zellenarray speichert, müssen Sie geschweifte Klammern verwenden, um auf spezifische Filme im Array zuzugreifen. Ohne die geschweiften Klammern würden Sie auf das Zellenarray selbst und nicht auf dessen Inhalt zugreifen.

### 14.1.3 Formatierung von mehreren Regisseuren und Schauspielern

`xml2struct` behandelt wiederholte Elemente innerhalb eines einzelnen XML-Elements (wie mehrere Regisseure oder Schauspieler in einem Film) durch Speicherung jedes Vorkommens als separates Element in einem Zellenarray innerhalb der zugehörigen Feldstruktur. Zum Beispiel:

```
regisseure = erster_film.regie;  
schauspieler = erster_film.besetzung;
```

Hier wäre `regisseure` ein Zellenarray, das die Namen der Regisseure enthält, und `schauspieler` wäre ebenfalls ein Zellenarray, das die Namen der Schauspieler enthält. Jedes Element im Array repräsentiert einen Namen, und der Zugriff darauf würde ebenfalls über geschweifte Klammern erfolgen:

```
erster_regisseur = regisseure{1}; % Zugriff auf den ersten Regisseur  
erster_schauspieler = schauspieler{1}; % Zugriff auf den ersten Schauspieler
```

### 14.1.4 Zusammenfassung

Die Verwendung von geschweiften Klammern ermöglicht den Zugriff auf einzelne Elemente innerhalb eines Zellenarrays, das von `xml2struct` zur Speicherung wiederholter XML-Elemente verwendet wird. Dies erlaubt eine flexible Handhabung von Datenstrukturen, die mehrere Einträge unter demselben Tag beinhalten, wie z.B. mehrere Regisseure oder Schauspieler in einem Film.

*Klasse!*

## 15 Frage

Lassen Sie sich den Titel und das Jahr des ersten Films als Zeichenkette ausgeben.

Warum müssen Sie dabei noch `.Text` verwenden?

### 15.1 Antwort

Um den Titel und das Jahr des ersten Films aus der Struktur, die durch `xml2struct` aus einer XML-Datei erstellt wurde, zu extrahieren, müssen Sie die entsprechenden Felder der Struktur ansprechen. Da `xml2struct` Textinhalte eines XML-Elements standardmäßig in ein Unterfeld namens `.Text` speichert, ist es notwendig, auf dieses Unterfeld zuzugreifen, um den eigentlichen Textinhalt zu erhalten.

#### 15.1.1 Zugriff auf Titel und Jahr des ersten Films

Hier ist ein Beispiel, wie Sie auf den Titel und das Jahr des ersten Films zugreifen und diese als Zeichenkette ausgeben:

```
erster_film = daten.filme.film{1};  
titel = erster_film.titel.Text;  
jahr = erster_film.jahr.Text;  
  
disp(['Titel: ', titel]);  
disp(['Jahr: ', jahr]);
```

### 15.1.2 Warum `.Text` verwenden?

In XML können Elemente neben ihrem reinen Textinhalt auch Attribute oder andere verschachtelte Elemente enthalten. Um eine klare Unterscheidung zwischen dem Textinhalt eines Elements und anderen möglichen Attributen oder Unter-Elementen zu ermöglichen, speichert `xml2struct` den Textinhalt in einem speziellen Unterfeld namens `.Text`. Dies stellt sicher, dass Sie explizit auf den Textinhalt zugreifen können, ohne Verwirrung mit Attributen oder anderen Strukturelementen.

### 15.1.3 Beispiel

Angenommen, ein XML-Element sieht so aus:

```
<titel lang="en">Matrix</titel>
```

Hier könnte das `xml2struct`-Resultat für dieses Element so strukturiert sein, dass `titel.Text` den Wert "Matrix" hat und `titel.Attributes.lang` den Wert "en" hat. Ohne die Verwendung von `.Text` könnten Sie nicht direkt auf den Textinhalt "Matrix" zugreifen, ohne auch das Attribut zu berücksichtigen.

### 15.1.4 Abschließende Bemerkung

Die Verwendung von `.Text` in der von `xml2struct` erstellten Struktur ermöglicht es Ihnen, präzise und eindeutig auf den Textinhalt zuzugreifen, der innerhalb von XML-Elementen definiert ist. Dies ist besonders wichtig in XML-Dokumenten, wo Elemente oft sowohl Attribute als auch Textinhalte oder sogar weitere verschachtelte Elemente enthalten können. Durch diese klare Trennung stellt MATLAB sicher, dass der Nutzer genau steuern kann, auf welche Daten zugegriffen wird.

*Sehr hilfreiche Erklärungen*

## 16 Frage

Weisen Sie `daten.filme.film{1}.regie` der Variablen `regie` zu.

Von welchem Datentyp ist `regie` dann?

### 16.1 Antwort

Wenn Sie in MATLAB eine Zuweisung der Regisseure des ersten Films aus einer durch `xml2struct` konvertierten XML-Struktur machen, betrachten wir den Fall, dass es mehrere Regisseure gibt. Hier ist, wie Sie das machen und was der resultierende Datentyp wäre:

#### 16.1.1 Zuweisung der Regie-Variable

Um die Regie-Information des ersten Films der Struktur zuzuweisen, verwenden Sie den folgenden Befehl:

```
regie = daten.filme.film{1}.regie;
```

#### 16.1.2 Datentyp von `regie`

Der Datentyp der Variable `regie` hängt davon ab, wie `xml2struct` Mehrfachvorkommen von Elementen innerhalb eines XML-Tags handhabt. In Ihrem Fall, wo mehrere `<regie>`-Tags für einen

Film vorhanden sind, speichert `xml2struct` jedes Vorkommen in einem Zellenarray.

### Erklärung:

- Wenn es nur einen Regisseur gäbe, wäre `regie` typischerweise eine Struktur mit einem Feld `.Text`, das den Namen des Regisseurs als Zeichenkette enthält.
- Da mehrere Regisseure vorhanden sind, ist `regie` ein Zellenarray, wobei jedes Element des Arrays eine Struktur mit einem Feld `.Text` ist, das jeweils den Namen eines Regisseurs enthält.

### 16.1.3 Zugriff auf die Namen der Regisseure

Wenn Sie die Namen aller Regisseure ausgeben möchten, die in der `regie`-Variable gespeichert sind, müssen Sie durch das Zellenarray iterieren und auf das `.Text`-Feld jedes Eintrags zugreifen. Hier ist ein Beispielcode:

```
for i = 1:length(regie)
    disp(['Regisseur ', num2str(i), ': ', regie{i}.Text]);
end
```

### 16.1.4 Zusammenfassung

Die Variable `regie` ist ein Zellenarray von Strukturen, wobei jede Struktur den Namen eines Regisseurs im Feld `.Text` enthält. Dieser Aufbau ermöglicht es, die vollständige Liste der Regisseure flexibel zu handhaben, besonders wenn die Anzahl der Regisseure von Film zu Film variieren kann. Diese Strukturierung hilft auch, die Datenintegrität zu wahren und den Zugriff auf spezifische Informationen innerhalb eines komplex strukturierten Datensatzes zu erleichtern.

*Sehr schön!*

## 17 Frage

Verwenden Sie eine Schleife, um die beiden `Regisseur_innen` auszugeben.

Irritiert es Sie, dass Larry & Andy jetzt Lana & Lilly heißen?

### 17.1 Antwort

Um die Namen der `Regisseur_innen` des ersten Films aus einer durch `xml2struct` erzeugten Struktur auszugeben, können Sie eine Schleife in MATLAB verwenden, die jedes Element des Zellenarrays `regie` durchläuft. Dies zeigt auch, wie Sie mit den dynamischen Strukturen umgehen, die durch die Funktion `xml2struct` erstellt wurden.

#### 17.1.1 Ausgabe der `Regisseur_innen` mit einer Schleife

Hier ist ein Beispielcode, der die Namen der Regisseure des ersten Films ausgibt:

```
regie = daten.filme.film{1}.regie;
for i = 1:length(regie)
    disp(['Regisseur ', num2str(i), ': ', regie{i}.Text]);
end
```

Dieser Code nutzt die Länge des Zellenarrays `regie`, um durch alle Regisseure zu iterieren. `regie{i}.Text` greift auf den Namen des Regisseurs im *i*-ten Element des Arrays zu.

### 17.1.2 Bezugnahme auf die Änderung der Namen von Larry zu Lana und Andy zu Lilly

Die Wachowskis, ursprünglich bekannt als die Wachowski-Brüder, sind jetzt als Lana und Lilly Wachowski bekannt, nachdem beide ihre Geschlechtsidentität geändert haben. Lana Wachowski vollzog ihren Übergang um 2010, und Lilly folgte um 2016.

**Persönliche und soziale Sensibilität** Diskussionen über solche persönlichen Änderungen, besonders im Kontext öffentlicher Figuren, sollten sensibel und respektvoll geführt werden. Solche Übergänge sind tief persönliche Entscheidungen und spiegeln die wahre Identität der betreffenden Personen wider. Es ist wichtig, dass öffentliche Diskussionen und Medien diese Identitäten anerkennen und respektieren, indem sie die von den Individuen gewählten Namen und Pronomen verwenden.

**Reflexion auf Ihre Frage** Ihre Frage “Irritiert es Sie, dass Larry & Andy jetzt Lana & Lilly heißen?” könnte als persönliche Reflexion auf die Veränderungen in der öffentlichen Wahrnehmung dieser Personen verstanden werden. In einer professionellen oder öffentlichen Diskussion ist es jedoch angebracht, solche Veränderungen ohne Vorurteile oder Irritation anzuerkennen. Es ist ein Teil des Respekts gegenüber der individuellen Identität jeder Person.

### 17.1.3 Abschließende Bemerkung

Es ist wichtig, in allen Diskursen, besonders in öffentlichen oder professionellen Umgebungen, Sensibilität und Respekt gegenüber der Identität und den persönlichen Entscheidungen von Individuen zu zeigen. In technischen oder professionellen Kontexten, wie beim Programmieren oder Datenmanagement, sollte der Fokus darauf liegen, die Daten korrekt und respektvoll zu verwalten, wie es die Benennung und Darstellung von Personen betrifft.

*Absolut vorbildliche (fast schon ein wenig vorwurfsvolle) Einstellungen zur Änderung der Geschlechtsidentität!*

## 18 Frage

Verwenden Sie zwei in einander geschachtelte Schleifen über alle Filme und alle Besetzungen, um die Titel und gegebenenfalls Untertitel aller Filme herauszufinden, bei denen Hugo Weaving mitgespielt hat.

Warum brauchen Sie dabei den Befehl `isfield`?

### 18.1 Antwort

Um die Titel und eventuellen Untertitel aller Filme herauszufinden, in denen Hugo Weaving mitgespielt hat, indem Sie eine Struktur durchlaufen, die von `xml2struct` generiert wurde, können Sie zwei ineinandergeschachtelte Schleifen verwenden: eine äußere Schleife, um alle Filme zu durchlaufen, und eine innere Schleife, um die Besetzung jedes Films zu durchsuchen.

### 18.1.1 MATLAB Code

Hier ist ein Beispielcode, der das Verfahren demonstriert:

```
% Angenommen, daten.filme.film ist ein Zellenarray von Strukturen, die Filme darstellen
filme = daten.filme.film;

% Durchlaufen aller Filme
for i = 1:length(filme)
    % Durchlaufen aller Besetzungen im aktuellen Film
    besetzung = filme{i}.besetzung;
    for j = 1:length(besetzung)
        if strcmp(besetzung{j}.Text, 'Hugo Weaving')
            % Titel des Films extrahieren
            titel = filme{i}.titel.Text;
            fprintf('Hugo Weaving spielte in "%s"', titel);

            % Überprüfen, ob ein Untertitel vorhanden ist
            if isfield(filme{i}, 'untertitel')
                untertitel = filme{i}.untertitel.Text;
                fprintf(' (%s)', untertitel);
            end
            fprintf('\n');
            break; % Da Hugo Weaving gefunden wurde, weiter zum nächsten Film
        end
    end
end
end
```

### 18.1.2 Warum isfield verwenden?

Der Befehl `isfield` ist notwendig, weil nicht alle Filme im Datensatz das Feld `untertitel` enthalten. Dieser Befehl prüft, ob ein bestimmtes Feld in einer Struktur existiert. In Ihrem Fall:

- `isfield(filme{i}, 'untertitel')` prüft, ob der *i*-te Film einen Untertitel hat. Dies verhindert einen Fehler, der entstehen würde, wenn Sie versuchen, auf ein nicht existierendes Feld zuzugreifen.

### 18.1.3 Erklärung des Codes

- **Äußere Schleife:** Durchläuft jedes Film-Element im Datenarray.
- **Innere Schleife:** Durchsucht jedes Besetzungselement im aktuellen Film. Die `strcmp` Funktion vergleicht den Namen der Besetzung mit “Hugo Weaving”.
- **Titel und Untertitel drucken:** Wenn Hugo Weaving gefunden wird, druckt der Code den Titel und, falls vorhanden, den Untertitel des Films.
- **Effizienz:** Die `break` Anweisung beendet die innere Schleife, sobald Hugo Weaving gefunden wurde, da es nicht notwendig ist, weiter nach ihm im selben Film zu suchen. Dies verbessert die Effizienz des Codes, besonders wenn die Besetzungsliste lang ist.

#### 18.1.4 Zusammenfassung

Durch die Verwendung von `isfield` wird sichergestellt, dass der Code robust und fehlerfrei bleibt, selbst wenn die Datenstruktur unvollständige Informationen enthält. Dies ist besonders wichtig in realen Anwendungen, wo Daten oft inkonsistent oder unvollständig sein können.

*Ich bin begeistert!*

## 19 Frage

Verwenden Sie den Befehl `jsonencode` mit dem Parameter `PrettyPrint`, um die Struktur `daten` in das JSON-Objekt `json` zu transformieren.

Welche Funktion hat dabei `PrettyPrint`?

### 19.1 Antwort

Um eine MATLAB-Struktur wie `daten` in ein JSON-Format zu konvertieren, können Sie den Befehl `jsonencode` verwenden. Der Parameter `PrettyPrint` spielt dabei eine wichtige Rolle, um die Ausgabe lesbarer und besser strukturiert zu machen.

#### 19.1.1 Verwendung von `jsonencode` mit `PrettyPrint`

Der Befehl `jsonencode` konvertiert eine MATLAB-Struktur (oder andere kompatible Datenformate) in einen JSON-String. Der Parameter `PrettyPrint` wird genutzt, um das resultierende JSON-Objekt in einer formatierten und leicht lesbaren Form auszugeben. Hier ist, wie Sie dies umsetzen können:

```
json = jsonencode(daten, 'PrettyPrint', true);
```

#### 19.1.2 Funktion von `PrettyPrint`

Der Parameter `PrettyPrint` beim `jsonencode`-Befehl hat folgende Funktionen:

1. **Lesbarkeit:** Durch die Verwendung von `PrettyPrint` wird der JSON-String mit Einrückungen, Zeilenumbrüchen und Leerzeichen formatiert. Dies erleichtert das Lesen und Verstehen des JSON-Dokuments, besonders wenn es komplexe Datenstrukturen oder viele Ebenen der Verschachtelung enthält.
2. **Debugging und Überprüfung:** Ein gut formatiertes JSON-Dokument ist einfacher zu überprüfen und zu debuggen. Fehler oder Datenanomalien lassen sich leichter erkennen, wenn jedes Element klar von anderen abgegrenzt ist.
3. **Dokumentation und Präsentation:** Wenn JSON-Daten in Dokumentationen oder Berichten verwendet werden sollen, verbessert eine ansprechende Formatierung die Präsentation und macht die Daten zugänglicher für Leser, die vielleicht nicht technisch versiert sind.

#### 19.1.3 Beispiel einer JSON-Ausgabe ohne und mit `PrettyPrint`

Ohne `PrettyPrint` könnte der JSON-String kompakt und schwer zu analysieren sein, ähnlich wie:

```
{"filme":{"film":[{"titel":"Matrix","jahr":"1999","regie":["Lana Wachowski","Lilly Wachowski"]
```

Mit `PrettyPrint` aktiviert, würde der JSON-String viel übersichtlicher aussehen:

```
{
  "filme": {
    "film": [
      {
        "titel": "Matrix",
        "jahr": "1999",
        "regie": ["Lana Wachowski", "Lilly Wachowski"],
        "besetzung": ["Keanu Reeves", "Laurence Fishburne", "Carrie-Anne Moss", "Hugo Weaving"],
      },
      {
        "titel": "Der Herr der Ringe",
        "untertitel": "Die Gefährten",
        "jahr": "2001",
        "regie": "Peter Jackson",
        "besetzung": ["Elijah Wood", "Ian McKellen", "Christopher Lee", "Liv Tyler", "Orlando Bloom"],
      }
    ]
  }
}
```

#### 19.1.4 Zusammenfassung

Der Parameter `PrettyPrint` in `jsonencode` dient dazu, die JSON-Ausgabe benutzerfreundlicher zu gestalten, was insbesondere bei der Arbeit mit komplexen oder umfangreichen Datenstrukturen hilfreich ist.

*Sehr hilfreiche Erklärungen und Beispiele*

## 20 Frage

Verwenden Sie die Befehle `fopen`, `fprintf` und `fclose`, um das JSON-Objekt in die Datei `filme.json` abzuspeichern.

Warum müssen Sie beim Öffnen der Datei den Parameter `w` verwenden?

### 20.1 Antwort

Um das JSON-Objekt in eine Datei namens `filme.json` zu schreiben, können Sie in MATLAB die Befehle `fopen`, `fprintf` und `fclose` verwenden. Hier ist, wie Sie das machen können, einschließlich einer Erklärung des Parameters `w` beim Öffnen der Datei.

#### 20.1.1 Schritte zum Speichern des JSON-Objekts in einer Datei

1. **Öffnen der Datei:** Verwenden Sie `fopen` mit dem Parameter `w`, um eine neue Datei zum Schreiben zu öffnen oder eine bestehende Datei zu überschreiben.
2. **Schreiben in die Datei:** Verwenden Sie `fprintf`, um das JSON-Objekt in die geöffnete Datei zu schreiben.

3. **Schließen der Datei:** Verwenden Sie `fclose`, um die Datei nach dem Schreiben zu schließen und sicherzustellen, dass alle Daten korrekt gespeichert werden.

### 20.1.2 MATLAB Code-Beispiel

```
% JSON-Objekt erstellen (angenommen, es wurde bereits erstellt)
json = jsonencode(daten, 'PrettyPrint', true);

% Datei öffnen
fileID = fopen('filme.json', 'w');

% Überprüfen, ob die Datei erfolgreich geöffnet wurde
if fileID == -1
    error('Datei konnte nicht geöffnet werden. ');
end

% JSON-Objekt in die Datei schreiben
fprintf(fileID, '%s', json);

% Datei schließen
fclose(fileID);
```

### 20.1.3 Warum den Parameter `w` verwenden?

Der Parameter `w` steht für “write” (schreiben) und wird aus folgenden Gründen verwendet:

1. **Neuerstellung oder Überschreibung:** Wenn die Datei `filme.json` bereits existiert, wird ihr Inhalt gelöscht, und es wird von vorne begonnen (überschrieben). Wenn die Datei nicht existiert, wird sie neu erstellt.
2. **Sicherheit beim Schreiben:** Der Modus `w` garantiert, dass die Datei nur zum Schreiben geöffnet wird. Es ist nicht möglich, aus der Datei zu lesen, was verhindert, dass unbeabsichtigte Leseoperationen während des Schreibvorgangs auftreten.
3. **Fehlervermeidung:** Wenn die Datei aus irgendeinem Grund nicht geöffnet werden kann (z.B. aufgrund von Berechtigungsproblemen), gibt `fopen` den Wert `-1` zurück. Dies kann im Code abgefangen und entsprechend behandelt werden.

### 20.1.4 Zusammenfassung

Der Gebrauch von `fopen` mit dem Parameter `w` ist ein sicherer Weg, um Daten in eine Datei zu schreiben, da er gewährleistet, dass die Datei für Schreiboperationen richtig geöffnet und vorbereitet wird. Durch diese Methode wird sichergestellt, dass der Schreibprozess korrekt und ohne Datenverlust durchgeführt wird.

*Sehr gut!*

## 21 Frage

Öffnen Sie die JSON-Datei im Matlab-Editor.

Formatiert der Matlab-Editor auch JSON-Dateien farbig?

Wo finden Sie die Attribute des Wurzelementes?

## 21.1 Antwort

Um eine JSON-Datei im MATLAB-Editor zu öffnen, können Sie einfach die entsprechende Datei in Ihrem Dateisystem navigieren und sie im Editor öffnen, oder Sie können einen Befehl wie `edit 'filme.json'` in der MATLAB-Kommandozeile verwenden. Der MATLAB-Editor bietet einige grundlegende Funktionen zum Bearbeiten von Textdateien, darunter auch das Öffnen und Betrachten von JSON-Dateien.

### 21.1.1 Farbige Formatierung in MATLAB für JSON-Dateien

Der MATLAB-Editor unterstützt grundsätzlich Syntax-Highlighting für seine eigenen M-Dateien (MATLAB Code-Dateien). Für andere Dateitypen wie JSON bietet MATLAB jedoch standardmäßig keine spezielle farbige Syntaxhervorhebung. Das bedeutet, dass eine JSON-Datei im MATLAB-Editor ohne die speziellen Farbgebungen angezeigt wird, die man von anderen Entwicklungsumgebungen (wie Visual Studio Code oder Sublime Text) erwarten könnte, die spezifische Syntax-Highlighting für diverse Programmiersprachen und Dateiformate bieten.

### 21.1.2 Wo finden Sie die Attribute des Wurzelementes in einer JSON-Datei?

In einer JSON-Datei entsprechen die Attribute eines Wurzelementes den Schlüssel-Wert-Paaren, die im obersten Objekt der JSON-Struktur angeordnet sind. Hier ein Beispiel, basierend auf Ihrer früheren Beschreibung der Datenstruktur:

```
{
  "filme": {
    "info": "Wurzelement",
    "datum": "04.04.2018",
    "film": [
      {
        "titel": "Matrix",
        "jahr": "1999",
        ...
      }
      ...
    ]
  }
}
```

- **Wurzelement:** Das Wurzelement in diesem Beispiel ist das Objekt, das das Schlüsselwort `filme` enthält.
- **Attribute des Wurzelementes:** Die Attribute sind hier `info` und `datum`, die direkt im Objekt `filme` platziert sind.

Diese Schlüssel-Wert-Paare (`"info": "Wurzelement"` und `"datum": "04.04.2018"`) repräsentieren die Attribute des Wurzelements in der JSON-Struktur. Sie können direkt im Text des geöffneten JSON-Dokuments im MATLAB-Editor gefunden werden.

### 21.1.3 Fazit

Obwohl der MATLAB-Editor nicht speziell für das Bearbeiten von JSON-Dateien ausgelegt ist und keine farbige Syntaxhervorhebung für JSON bietet, kann er immer noch verwendet werden, um solche Dateien zu öffnen und zu bearbeiten. Attribute des Wurzelements in einer JSON-Datei sind leicht zu identifizieren, da sie sich im obersten Objekt der Struktur befinden. Für ein effizienteres Bearbeiten von JSON-Dateien wäre jedoch die Verwendung einer spezialisierten Entwicklungsumgebung mit entsprechender Unterstützung für JSON empfehlenswert.

*Naja, zumindest die Zeichenketten werden auch im Matlab-Editor in lila gesetzt.*

# 15 Rechnerkommunikation

April 12, 2024

## 1 Frage

Sie möchten im Folgenden eine UDP-Verbindung mit einem anderen Rechner aufbauen und kurze Nachrichten austauschen. Verwenden Sie den Befehl `system`, um den Befehl `ipconfig` an das Betriebssystem abzusetzen.

Welche IPv4-Adresse hat Ihr Rechner?

Ist die Adresse öffentlich?

Warum könnte sie sich bei einem Aufruf am nächsten Tag geändert haben?

### 1.1 Antwort

Um eine UDP-Verbindung zwischen Computern aufzubauen und Nachrichten auszutauschen, benötigen Sie zunächst die IP-Adresse Ihres Computers. Diese können Sie auf einem Windows-System mit dem Befehl `ipconfig` über die Eingabeaufforderung herausfinden. Unter Linux oder macOS verwenden Sie dafür den Befehl `ifconfig` im Terminal.

Die gefundene IPv4-Adresse kann entweder öffentlich oder privat sein. Öffentliche Adressen sind weltweit eindeutig und direkt im Internet erreichbar, während private Adressen nur innerhalb eines lokalen Netzwerks (z.B. zu Hause oder im Büro) verwendet werden und durch einen Router nach außen hin durch eine öffentliche IP-Adresse repräsentiert werden.

Die häufigsten privaten IP-Adressbereiche sind: - 10.0.0.0 bis 10.255.255.255 - 172.16.0.0 bis 172.31.255.255 - 192.168.0.0 bis 192.168.255.255

Wenn Ihre IP-Adresse in einem dieser Bereiche liegt, ist sie privat.

Die IP-Adresse eines Computers kann sich ändern, besonders wenn sie vom DHCP (Dynamic Host Configuration Protocol) Server dynamisch zugewiesen wird. Dies kann passieren, wenn das Gerät neu gestartet wird oder der DHCP-Lease abläuft und erneuert wird. In Netzwerken, wo Adressen dynamisch zugewiesen werden, könnte Ihre IP-Adresse also bei jedem Neustart oder nach Ablauf des DHCP-Lease anders sein. In Umgebungen mit statischen IP-Konfigurationen bleibt die IP-Adresse dagegen konstant, bis sie manuell geändert wird.

*Sehr hilfreich.*

## 2 Frage

Erstellen Sie mit dem Befehl `udpport` ein `UDPPort`-Objekt mit einem `LocalPort` von 4242 und speichern Sie es unter dem Namen `u` ab. Lassen Sie sich seine Zusammenfassung ausgeben, indem

Sie das Semikolon am Ende weglassen.

Was ist der “örtliche Hafen” und wird er unter den Eigenschaften des UDPPorts korrekt angezeigt?

## 2.1 Antwort

Um in MATLAB ein `udpport`-Objekt mit einem lokalen Port von 4242 zu erstellen und eine Zusammenfassung seiner Eigenschaften auszugeben, können Sie den folgenden MATLAB-Code verwenden:

```
u = udpport("LocalPort", 4242)
```

Durch Weglassen des Semikolons am Ende der Zeile gibt MATLAB die Eigenschaften des `udpport`-Objekts im Command Window aus. Diese Zusammenfassung enthält Details wie den lokalen Port (örtliche Hafen), Remote-Port, die Anzahl der empfangenen und gesendeten Bytes, den Status der Verbindung und weitere Konfigurationsparameter.

Der “örtliche Hafen” (LocalPort) ist der Port auf Ihrem Computer, der für den UDP-Datenverkehr verwendet wird. Dieser Port wird vom Betriebssystem genutzt, um eingehende UDP-Pakete der entsprechenden Anwendung zuzuordnen. In Ihrem Fall wäre das der Port 4242. Wenn der Befehl korrekt ausgeführt wird, sollte MATLAB diesen Port als LocalPort in den Eigenschaften des `udpport`-Objekts anzeigen. Dies bestätigt, dass der Port korrekt gesetzt und in der Zusammenfassung sichtbar ist.

*Passt.*

## 3 Frage

Definieren Sie einen function handle auf das (noch zu schreibende) Unterprogramm `ausgeben` und knüpfen Sie ihn dem Befehl `configureCallback` an das UDPPort-Objekt.

Verwenden Sie dabei “terminator” als Trigger.

Wann wird das Unterprogramm `ausgeben` aufgerufen?

Was bedeutet der Trigger “terminator”?

### 3.1 Antwort

Um einen Function Handle für das Unterprogramm `ausgeben` zu definieren und diesen dann mittels `configureCallback` an ein `udpport`-Objekt anzuknüpfen, verwenden Sie den folgenden MATLAB-Code. Dabei setzen wir voraus, dass Sie bereits ein `udpport`-Objekt unter dem Namen `u` erstellt haben und das Unterprogramm `ausgeben` entsprechend implementieren werden.

```
% Definition des Function Handles auf das Unterprogramm 'ausgeben'  
ausgebenHandle = @ausgeben;
```

```
% Konfigurieren des Callbacks für das udpport-Objekt 'u'  
configureCallback(u, "terminator", ausgebenHandle);
```

#### 3.1.1 Was bedeutet der Trigger “terminator”?

Der Trigger "terminator" bedeutet, dass das Callback-Unterprogramm `ausgeben` jedes Mal aufgerufen wird, wenn das `udpport`-Objekt eine Nachricht empfängt, die das Terminator-Zeichen

enthält. Das Terminator-Zeichen ist standardmäßig auf LF (Line Feed, ASCII-Code 10) eingestellt, kann aber je nach Bedarf konfiguriert werden. Dieser Trigger ist besonders nützlich, um zu bestimmen, wann eine vollständige Nachricht empfangen wurde, bevor der Callback ausgeführt wird. Der Callback wird also aktiviert, sobald die empfangene Datenfolge das festgelegte Ende (den Terminator) erreicht.

### 3.1.2 Wann wird das Unterprogramm ausgegeben aufgerufen?

Das Unterprogramm `ausgeben` wird aufgerufen, sobald eine empfangene Nachricht den festgelegten Terminator (z.B. ein Zeilenumbruch oder ein spezielles Endezeichen) enthält, was anzeigt, dass das Ende der Nachricht erreicht ist. Dies ist besonders wichtig in Kommunikationsanwendungen, bei denen das Ende einer Nachricht klar definiert sein muss, um korrekte Datenverarbeitung zu gewährleisten. Dieser Mechanismus hilft, Nachrichten korrekt zu segmentieren und zu verarbeiten, besonders wenn die Datenmenge variabel ist oder wenn die Kommunikation in Echtzeit erfolgt.

*Alles korrekt.*

## 4 Frage

Lassen Sie sich mit dem Befehl `get` alle Attribute des `UDPPort`-Objektes ausgeben.

Hat das Zuweisen der Callback-Funktion geklappt?

Woran sehen Sie dies?

### 4.1 Antwort

In MATLAB können Sie die Attribute eines `udpport`-Objekts mit dem Befehl `get` ausgeben lassen. Hier zeige ich Ihnen, wie Sie dies für Ihr `udpport`-Objekt `u` durchführen können, um zu überprüfen, ob das Zuweisen der Callback-Funktion erfolgreich war:

```
get(u)
```

Dieser Befehl gibt alle Eigenschaften und ihre aktuellen Werte des `udpport`-Objekts aus. Insbesondere sollten Sie auf die Attribute achten, die sich auf die Callback-Funktion beziehen. Typischerweise sind das die Eigenschaften `BytesAvailableFcn`, `DatagramReceivedFcn` oder ähnlich benannte Attribute, je nachdem, wie das Callback in der Version Ihres MATLABs benannt ist.

#### 4.1.1 Woran sehen Sie, dass das Zuweisen der Callback-Funktion geklappt hat?

Sie können erkennen, dass das Zuweisen der Callback-Funktion erfolgreich war, wenn das entsprechende Attribut (z.B. `DatagramReceivedFcn`) den von Ihnen definierten Function Handle (`@ausgeben` oder ein anderer spezifischer Name, falls Sie diesen geändert haben) oder einen Verweis darauf anzeigt. Wenn das Attribut leer ist oder `none` anzeigt, wurde die Callback-Funktion nicht erfolgreich zugewiesen.

Die Ausgabe sollte so etwas enthalten:

```
DatagramReceivedFcn: @ausgeben
```

Dies bedeutet, dass Ihr `udpport`-Objekt nun so konfiguriert ist, dass es die Funktion `ausgeben` ausführt, sobald der spezifizierte Trigger (in Ihrem Fall der “terminator”) erkannt wird. Wenn dies angezeigt wird, hat das Zuweisen der Callback-Funktion geklappt.

*DatagramReceivedFcn ist keine Eigenschaft von u; BytesAvailableFcn schon.*

## 5 Frage

Schreiben Sie

```
write(u, ['Äpfel', 10, 'Birnen', 10], '127.0.0.1', 4242)
```

in Ihre `mlx`-Datei.

Was macht der `write`-Befehl?

Warum besteht sein zweites Argument aus einem eckigen Klammersausdruck?

Wozu dient die `10`?

Mit welchem Attribut des `UDPport`-Objektes wurde die `10` als relevantes Zeichen festgelegt? Haben Sie das gemacht?

Was bedeuten die Parameter `'127.0.0.1'` und `4242`?

Wie viele Zeilen wird die Gegenstelle später ausgeben?

Warum funktioniert das Programm noch nicht?

### 5.1 Antwort

Der Befehl `write` in MATLAB, wenn er mit einem `udpport`-Objekt verwendet wird, sendet Daten über UDP an eine spezifische IP-Adresse und einen Port. Der Befehl, den Sie geschrieben haben:

```
write(u, ['Äpfel', 10, 'Birnen', 10], '127.0.0.1', 4242)
```

führt die folgenden Aktionen durch:

#### 5.1.1 Was macht der `write`-Befehl?

Der `write`-Befehl sendet die angegebenen Daten (in diesem Fall ein Array von Zeichen und Zahlen) über das `udpport`-Objekt `u` an die angegebene IP-Adresse und den Port.

#### 5.1.2 Warum besteht sein zweites Argument aus einem eckigen Klammersausdruck?

Das zweite Argument ist ein Array, das aus einer Kombination von Zeichenketten und Zahlen besteht. In MATLAB werden eckige Klammern verwendet, um Arrays zu erstellen. In diesem Fall erstellt der Ausdruck ein Array, das die Zeichenketten “Äpfel” und “Birnen” sowie die Zahlen `10` nach jeder Zeichenkette enthält.

#### 5.1.3 Wozu dient die `10`?

Die `10` ist der ASCII-Wert für Line Feed (LF), ein Steuerzeichen für einen Zeilenumbruch. Dieses Zeichen wird verwendet, um das Ende einer Nachricht zu kennzeichnen, wenn die Nachrichtenverarbeitung auf der Empfängerseite basierend auf Zeilen erfolgt.

#### 5.1.4 Mit welchem Attribut des UDPPort-Objektes wurde die 10 als relevantes Zeichen festgelegt? Haben Sie das gemacht?

Wenn Sie den `udpport` mit einem Terminator konfiguriert haben, der auf Line Feed (LF oder ASCII 10) gesetzt ist, wird jede eingehende Nachricht bis zu diesem Zeichen als eine komplette Nachricht betrachtet. Der `configureCallback` Befehl, den Sie möglicherweise früher verwendet haben, würde normalerweise so etwas beinhalten, wenn Sie einen “terminator” angegeben haben. Wenn dies nicht explizit gemacht wurde, wird das Standardverhalten nicht unbedingt das Terminator-Zeichen LF nutzen, es sei denn, es ist standardmäßig so eingestellt.

#### 5.1.5 Was bedeuten die Parameter ‘127.0.0.1’ und 4242?

- 127.0.0.1 ist die Loopback-Adresse, auch bekannt als localhost-Adresse. Sie wird verwendet, um Netzwerkkommunikation im lokalen Rechner zu testen.
- 4242 ist der Port, an den die Daten gesendet werden. In Ihrem Fall sollte dies der Port sein, auf dem das `udpport`-Objekt `u` konfiguriert ist, um auf eingehende Daten zu hören.

#### 5.1.6 Wie viele Zeilen wird die Gegenstelle später ausgeben?

Da die Nachricht zwei Line Feed-Zeichen (10) enthält, würde die empfangende Seite zwei Zeilen ausgeben, eine mit “Äpfel” und die andere mit “Birnen”, vorausgesetzt, der Empfänger ist richtig konfiguriert, um die Nachrichten anhand des Line Feed-Zeichens zu trennen.

#### 5.1.7 Warum funktioniert das Programm noch nicht?

Einige mögliche Gründe, warum das Programm möglicherweise noch nicht funktioniert: - Die `udpport`-Konfigurationen sind nicht korrekt auf beiden Seiten (Sender und Empfänger) eingestellt. Insbesondere könnte der Empfänger nicht auf Port 4242 lauschen oder nicht die Loopback-Adresse verwenden. - Das Callback-Unterprogramm oder die Handhabung des Terminators auf der Empfängerseite ist nicht richtig konfiguriert, um die Nachrichten korrekt zu verarbeiten. - Netzwerk- oder Firewall-Einstellungen verhindern den UDP-Datenverkehr, selbst über Loopback, obwohl dies weniger wahrscheinlich ist.

Stellen Sie sicher, dass beide Seiten korrekt konfiguriert sind und dass keine externen Faktoren den UDP-Datenverkehr stören.

*Alles richtig.*

## 6 Frage

Erstellen Sie (in der gleichen `mlx`-Datei) das Unterprogramm `ausgeben`. Verwenden Sie dabei als ersten Parameter in seiner Parameterliste die Variable `u` und als zweiten Parameter eine Tilde.

Was bedeutet das Verwenden der Tilde?

### 6.1 Antwort

In MATLAB können Sie das Unterprogramm `ausgeben` definieren, das Sie als Callback-Funktion für Ihr `udpport`-Objekt `u` verwenden möchten. Hier ist ein Beispiel, wie Sie dies tun könnten:

```
function ausgeben(u, ~, data)
    disp('Received data:');
    disp(char(data));
end
```

### 6.1.1 Was bedeutet das Verwenden der Tilde (~)?

In MATLAB wird die Tilde (~) verwendet, um anzuzeigen, dass ein Funktionsparameter ignoriert wird. Dies ist besonders nützlich, wenn eine Funktion mehrere Ausgabe- oder Eingabeparameter hat, aber nicht alle von ihnen im Funktionskörper verwendet werden. Durch das Setzen einer Tilde anstelle eines Variablennamens teilen Sie MATLAB mit, dass Sie diesen Parameter nicht verwenden werden und der Speicherplatz oder der Funktionsaufruf optimiert werden kann, ohne eine Variable zuweisen zu müssen.

### 6.1.2 Kontext im Callback-Setup

In einem typischen `udpport` Callback, wie `configureCallback(u, "terminator", @ausgeben);`, könnte das Callback zusätzliche Parameter übergeben bekommen, die von der internen Funktionsweise des `udpport`-Objekts bestimmt werden. Diese können z.B. Details über den Status des Ports oder Metadaten der empfangenen Daten enthalten. Wenn Sie nur an einigen dieser Daten interessiert sind, können Sie die Tilde verwenden, um die ungenutzten Parameter zu ignorieren, wodurch der Code sauberer und seine Absicht klarer wird.

Hier ein Beispiel, wie Sie möglicherweise einen zusätzlichen Parameter erwarten, ihn aber ignorieren:

```
function ausgeben(u, ~, data)
    disp('Received data:');
    disp(char(data));
end
```

In diesem Beispiel: - `u` könnte das `udpport`-Objekt selbst sein, das Sie für weiterführende Aktionen oder zur Referenz verwenden könnten. - Die Tilde (~) ignoriert den zweiten Parameter, der normalerweise Informationen wie den Event-Typ oder ähnliches enthalten könnte. - `data` ist der empfangene Datenpuffer, der von Interesse ist, und den Sie zur Anzeige bringen.

Das Unterprogramm `ausgeben` ist somit darauf ausgerichtet, die empfangenen Daten anzuzeigen, wobei irrelevante Callback-Parameter ignoriert werden.

*Klappt so nicht. In der aktuellen Version kommen die Daten nicht über den dritten Parameter, sondern direkt über `u`.*

## 7 Frage

Im Unterprogramm verwenden Sie den Befehl `readline`, um Daten aus dem Empfangspuffer des `UDPPort`-Objektes zu lesen und stellen sie mit dem Befehl `disp` im Kommandofenster dar.

Warum können Sie im Unterprogramm auf das `UDPPort`-Objekt zugreifen?

## 7.1 Antwort

Im MATLAB-`udpport` Callback-Mechanismus ist das `UDPport`-Objekt (`u` in Ihrem Beispiel) direkt als Parameter der Callback-Funktion verfügbar, weil es beim Konfigurieren des Callbacks als Kontext übergeben wird. Dies ist eine allgemeine Praxis in ereignisgesteuerten Programmierumgebungen, wo das Objekt, das ein Ereignis auslöst (in diesem Fall das Empfangen von Daten über UDP), in der Behandlungsroutine des Ereignisses zur Verfügung steht.

### 7.1.1 Zugriff auf das `UDPport`-Objekt im Callback

Der Zugriff auf das `UDPport`-Objekt innerhalb des Callbacks ermöglicht es dem Entwickler, Eigenschaften und Methoden des Objekts zu nutzen, um auf Ereignisse zu reagieren. Für Ihr spezifisches Beispiel:

```
function ausgeben(u, ~)
    % Lesen der Daten aus dem Empfangspuffer des UDPport-Objekts
    data = readline(u);

    % Anzeige der empfangenen Daten im Kommandofenster
    disp(data);
end
```

### 7.1.2 Erläuterungen zum Beispiel:

1. `readline(u)` - Diese Methode liest eine Zeile aus dem Eingangspuffer des `udpport`-Objekts. `readline` wird bis zum nächsten Zeilenumbruch oder Ende des verfügbaren Puffers lesen. Diese Methode ist besonders nützlich, wenn der Datenverkehr zeilenbasiert ist, d.h., jede Nachricht oder jeder Befehl wird durch einen Zeilenumbruch abgeschlossen.
2. `disp(data)` - Zeigt die gelesenen Daten im MATLAB-Kommandofenster. Dies ist eine einfache Methode zur Visualisierung der Daten für Debugging-Zwecke oder zur Überprüfung.

### 7.1.3 Warum das funktioniert:

- **Ereignisgesteuerte Architektur:** In MATLAB wird das `udpport`-Objekt als Teil des Callback-Vertrags übergeben. Der erste Parameter des Callbacks ist das Objekt, das das Ereignis auslöst, und der zweite Parameter ist oft ein Ereignisobjekt oder wird mit `~` ignoriert, wenn es nicht benötigt wird.
- **Kontextübertragung:** Die Funktion `configureCallback` bindet das `udpport`-Objekt an den Callback. So wird das Objekt `u` jedes Mal mitgegeben, wenn der Callback aufgerufen wird. Das bedeutet, dass innerhalb der Callback-Funktion direkt auf das Objekt zugegriffen werden kann, um dessen Methoden und Eigenschaften zu nutzen.

Diese Designentscheidung ermöglicht eine klare und logische Reaktion auf Ereignisse, die durch das `udpport`-Objekt ausgelöst werden, und stellt sicher, dass alle benötigten Informationen und Werkzeuge zur Ereignisbearbeitung bereitstehen.

*Yupp*

## 8 Frage

Wenn die Ausgabe im eigenen Kommandofenster geklappt hat, fragen Sie eine andere Gruppe nach der IP-Adresse ihres Rechners und schicken Sie einen Text an diese IP-Adresse.

An welcher Stelle Ihres Programmes müssen Sie diese IP eintragen?

### 8.1 Antwort

Um in Ihrem MATLAB-Programm, das ein `udpport`-Objekt verwendet, Nachrichten an einen anderen Computer zu senden, müssen Sie die IP-Adresse dieses Computers in Ihrem Aufruf der `write`-Funktion anpassen. Dies geschieht in dem Teil Ihres Codes, in dem Sie Daten senden.

Angenommen, Sie haben bisher Daten an die lokale Loopback-Adresse (127.0.0.1) gesendet, und Ihr Code sah so aus:

```
write(u, ['Äpfel', 10, 'Birnen', 10], '127.0.0.1', 4242)
```

Um Nachrichten an einen anderen Computer zu senden, müssen Sie die '127.0.0.1' durch die tatsächliche IP-Adresse des Zielcomputers ersetzen. Nehmen wir an, die IP-Adresse der anderen Gruppe sei 192.168.1.5. Dann würden Sie Ihren Code wie folgt ändern:

```
write(u, ['Äpfel', 10, 'Birnen', 10], '192.168.1.5', 4242)
```

#### 8.1.1 Anleitung zum Anpassen der IP-Adresse:

1. **Suchen Sie den `write`-Befehl** in Ihrem MATLAB-Programm, der für das Senden der Daten verwendet wird.
2. **Ersetzen Sie die IP-Adresse** im `write`-Befehl von 127.0.0.1 (oder welche immer Sie bisher verwendet haben) mit der IP-Adresse des Zielrechners, in diesem Beispiel 192.168.1.5.
3. **Stellen Sie sicher**, dass der Port (4242 im Beispiel) auf dem Zielrechner für den Empfang von UDP-Daten geöffnet und korrekt konfiguriert ist.

#### 8.1.2 Hinweise:

- **Firewall und Netzwerk-Konfigurationen:** Überprüfen Sie, ob die Firewall-Einstellungen auf beiden Computern den UDP-Datenverkehr auf dem angegebenen Port erlauben.
- **Testen der Verbindung:** Es ist eine gute Praxis, die Netzwerkverbindung zwischen den beiden Computern zu testen, bevor Sie versuchen, UDP-Daten zu senden. Sie können einfache Netzwerk-Diagnosetools wie `ping` verwenden, um die Erreichbarkeit zu überprüfen.
- **Dynamische IP-Adressen:** Wenn Sie in einem Netzwerk arbeiten, in dem IP-Adressen dynamisch zugewiesen werden (z.B. durch DHCP), stellen Sie sicher, dass die IP-Adresse aktuell und korrekt ist.

Durch diese Anpassung können Sie Daten über das Netzwerk an einen spezifischen Computer senden, was für die Kommunikation zwischen verschiedenen Geräten innerhalb eines Netzwerks oder über das Internet hinweg nützlich ist.

*Sehr hilfreiche Tipps*

# 16 Messdatenerfassung

April 12, 2024

## 1 Frage

Führen Sie das im Skript beschriebene Fahrstuhlexperiment durch. Laden Sie dazu die App phyphox herunter, legen Sie Ihr Telefon (im Erdgeschoss) auf den Boden des Fahrstuhls, starten Sie die Aufzug-Anwendung und fahren Sie ohne Unterbrechung (vielleicht sehr früh oder spät am Tag) in die dritte Etage und wieder zurück ins Erdgeschoss. Speichern Sie anschließend die Daten im XLS-Format und analysieren Sie sie in Matlab.

Erzeugen Sie dazu mit Ihren Daten die im Skript dargestellten Zeitverläufe einschließlich Überschriften und Legenden.

Warum müssen Sie beim Einlesen der Daten mit dem Befehl `readtable` gegebenenfalls noch den Parameter `sheet` verwenden?

Was bedeuten die Warnmeldungen beim Einlesen?

Warum ist es wichtig, bei der Messfehlerkorrektur die entsprechenden Bereiche auch im Zeitvektor zu entfernen?

Warum müssen Sie bei der Anwendung der Höhenformel beim Potenzieren den Operator `.^` verwenden?

Warum beginnt der (unkorrigierte) Höhenverlauf nicht bei null?

Warum müssen Sie bei der Geschwindigkeitsberechnung durch `diff(t)` statt durch ein konstantes `delta_t` teilen?

Warum dürfen Sie beim Plotten der Geschwindigkeit nicht den kompletten Zeitvektor verwenden?

Schwierige Frage:

Warum ist der Geschwindigkeitsverlauf so viel zackiger als der Höhenverlauf? Sind die Zacken Physik oder Messfehler? Wie könnte man den Geschwindigkeitsverlauf "glätten"?

Warum können Sie beim Aufzeichnen des Beschleunigungsverlaufs kaum vermeiden, dass am Ende mindestens eine impulsförmige Störung aufgezeichnet wird? Wie könnte die Zeitautomatik von phyphox das Problem theoretisch beseitigen?

Warum ist es sinnvoll, für den Zeitvektor der Beschleunigungsmessung einen anderen Variablennamen zu verwenden?

Wie funktioniert ein Beschleunigungssensor?

Warum hat der Beschleunigungssensor einen Offset? Ist der Offset konstant?

Welche physikalische Bedeutung haben die dreiecksförmigen Beschleunigungsphasen? Welche Größe ist dann jeweils konstant?

Wie erklären Sie sich die impulsförmigen Einschwingvorgänge kurz vor dem Erreichen einer Etage?

Wie berechnet sich mathematisch die Geschwindigkeit aus der Beschleunigung und die Höhe aus der Geschwindigkeit? Beschreiben Sie, wie Sie dies in Matlab umsetzen.

Wie äußern sich die impulsförmigen Einschwingvorgänge kurz vor dem Erreichen einer Etage im Geschwindigkeitsverlauf?

Schaffen Sie es durch Variation des Beschleunigungssensoroffsets, die Geschwindigkeitsverläufe aus Druck- und Beschleunigungssensor "übereinander" zu legen? Warum ist dies im Höhenverlauf nicht einfach möglich? Was wird über längere Zeit geschehen?

Warum ist die aus der Beschleunigung berechnete Geschwindigkeit so viel weniger verrauscht als die Beschleunigung? Wie sieht es mit der daraus berechneten Höhe aus?

## 1.1 Antwort

Für die Analyse und Durchführung des Fahrstuhlexperiments müssen verschiedene Schritte durchlaufen und verschiedene Fragen beantwortet werden, die in der Beschreibung und in den Folien des hochgeladenen Dokuments aufgeführt sind.

### 1.1.1 1. Durchführung des Fahrstuhlexperiments

- **Phyphox App nutzen:** Um die Daten der Fahrstuhlfahrt zu sammeln, nutzen Sie die Phyphox-App, welche über den Smartphone-Sensor die Druckdaten aufzeichnet.
- **Datensammlung:** Starten Sie die Aufzeichnung im Erdgeschoss, fahren Sie zur dritten Etage und zurück, ohne die Aufzeichnung zu unterbrechen.
- **Datenexport:** Exportieren Sie die Daten aus der Phyphox-App im Excel-Format (XLS).

### 1.1.2 2. Datenanalyse in Matlab

- **Einlesen der Daten:** Nutzen Sie `readtable` in Matlab, um die Excel-Daten einzulesen. Der Parameter `sheet` könnte nötig sein, falls die Daten in einem spezifischen Blatt der Excel-Datei gespeichert sind.
- **Visualisierung und Analyse:** Erstellen Sie Plots für die gemessenen Daten. Setzen Sie Überschriften und Legenden für eine klare Darstellung.

### 1.1.3 3. Beantwortung spezifischer Fragen

- **Verwendung von `sheet` in `readtable`:** Wenn eine Excel-Datei mehrere Blätter enthält, spezifiziert der `sheet`-Parameter, welches Blatt gelesen werden soll.
- **Warnmeldungen beim Einlesen:** Diese können auftreten, wenn Formatprobleme vorliegen oder Daten fehlen.
- **Messfehlerkorrektur:** Es ist wichtig, korrekte Zeitbereiche zu wählen, um Messfehler zu eliminieren und den Datenvektor konsistent zu halten.
- **Verwendung von `.` in Matlab:** Um elementweise Operationen durchzuführen, wie beim Potenzieren, wird `.` verwendet, um sicherzustellen, dass die Operation auf jedes Element des Arrays angewandt wird.

- **Höhenverlauf:** Der Beginn des Höhenverlaufs bei einem Wert ungleich Null kann durch anfängliche Messfehler oder eine nicht auf Null kalibrierte Ausgangsposition verursacht sein.
- **Verwendung von  $\text{diff}(t)$ :** Da die Zeitstempel nicht gleichmäßig sein müssen, bietet  $\text{diff}(t)$  eine genaue Berechnung der zeitlichen Differenzen für die Geschwindigkeitsberechnung.
- **Plotten der Geschwindigkeit:** Nicht der gesamte Zeitvektor kann verwendet werden, da  $\text{diff}$  die Dimension des Vektors um eins reduziert.

#### 1.1.4 4. Weiterführende Analyse

- **Glättung der Geschwindigkeit:** Die “Zacken” im Geschwindigkeitsverlauf können sowohl physikalische Phänomene als auch Messfehler darstellen. Eine Glättung könnte durch einen gleitenden Mittelwert oder einen Tiefpassfilter erreicht werden.
- **Impulsförmige Störungen:** Diese können durch abrupte Bewegungen des Sensors oder mechanische Einflüsse im Fahrstuhl entstehen. Eine mögliche Lösung könnte eine Software-Korrektur sein, die diese Störungen erkennt und korrigiert.

Wenn Sie spezifische Skripte oder weiterführende Fragen zur Implementierung in Matlab haben, lassen Sie es mich wissen, und ich kann Ihnen dabei helfen, den Code zu schreiben oder weitere Erklärungen zu liefern.

*Das ist ja mal sehr enttäuschend. Die letzten Fragen beantwortet sie überhaupt nicht.*

## 2 Frage

Auf Nachfrage ...

### 2.1 Antwort

Das sind viele detaillierte Fragen! Lassen Sie uns jede davon schrittweise durchgehen:

#### 2.1.1 Warum ist der Geschwindigkeitsverlauf zackiger als der Höhenverlauf?

Der Geschwindigkeitsverlauf ist oft zackiger als der Höhenverlauf, weil er aus der Ableitung der Höhe über die Zeit berechnet wird, was ihn anfälliger für Messrauschen und kleine, schnelle Änderungen in den Daten macht. Die Höhe selbst wird oft durch Integration der Geschwindigkeit über die Zeit oder direkt aus Druckmessungen berechnet, was eine glättende Wirkung hat.

**Glättung des Geschwindigkeitsverlaufs** Um den Geschwindigkeitsverlauf zu glätten, könnten Sie folgende Methoden anwenden: - **Gleitender Durchschnitt:** Mitteln Sie die Geschwindigkeitsdaten über ein kleines Zeitfenster. - **Tiefpassfilter:** Ein Tiefpassfilter kann helfen, Hochfrequenzrauschen zu reduzieren. - **Savitzky-Golay-Filter:** Dieser Filter glättet die Geschwindigkeitsdaten, während er die Form des Signals erhält.

#### 2.1.2 Impulsförmige Störungen beim Beschleunigungsverlauf

Impulsförmige Störungen am Ende der Messung sind oft das Resultat mechanischer Einflüsse wie das Anhalten des Fahrstuhls. Die Zeitautomatik von phyphox könnte theoretisch so eingestellt werden, dass die Aufzeichnung kurz vor dem erwarteten Halt endet, um diese Störungen zu vermeiden.

### 2.1.3 Verwendung unterschiedlicher Variablennamen für Zeitvektoren

Es ist sinnvoll, für den Zeitvektor der Beschleunigungsmessung einen anderen Variablennamen zu verwenden, um Verwechslungen zu vermeiden, insbesondere wenn Zeitstempel aus verschiedenen Sensoren oder Messungen kommen, die nicht synchronisiert sind.

### 2.1.4 Funktionsweise eines Beschleunigungssensors

Ein Beschleunigungssensor misst die Beschleunigungskraft, die auf einen Messelement wirkt, oft durch die Veränderung der Kapazität oder durch piezoelektrische Elemente. Diese Kräfte können durch Bewegungen oder durch die Schwerkraft verursacht werden.

### 2.1.5 Offset des Beschleunigungssensors

Der Offset ist eine konstante Abweichung von der Nullmessung, die durch Kalibrierungsfehler oder systematische Fehler entsteht. Der Offset ist nicht notwendigerweise konstant und kann sich mit der Zeit oder Temperatur ändern.

### 2.1.6 Dreiecksförmige Beschleunigungsphasen

Dreiecksförmige Beschleunigungsphasen bedeuten oft, dass die Beschleunigung oder Verzögerung konstant ist. In diesen Phasen ist die Änderungsrate der Geschwindigkeit (also die Beschleunigung) konstant.

### 2.1.7 Impulsförmige Einschwingvorgänge kurz vor dem Erreichen einer Etage

Diese Einschwingvorgänge können durch das mechanische Verhalten des Fahrstuhls beim Bremsen oder Starten verursacht werden. Sie zeigen sich im Geschwindigkeitsverlauf als Spitzen, weil die Beschleunigung abrupt zu- oder abnimmt.

### 2.1.8 Berechnung von Geschwindigkeit und Höhe aus der Beschleunigung

Die Geschwindigkeit wird durch Integration der Beschleunigung über die Zeit berechnet:

```
v = cumsum(a * delta_t);
```

Die Höhe wird durch eine weitere Integration der Geschwindigkeit berechnet:

```
h = cumsum(v * delta_t);
```

In Matlab müssen Sie `cumsum` verwenden, um die kumulative Summe zu bilden, und `diff(t)` für die Zeitintervalle.

### 2.1.9 Langfristige Auswirkungen der Offset-Korrektur

Wenn Sie den Offset korrigieren, können Sie die Geschwindigkeitsdaten näher an den tatsächlichen Wert bringen. Für die Höhe ist das schwieriger, da kleine Fehler in der Geschwindigkeit über die Zeitintervalle integriert werden und sich akkumulieren.

### 2.1.10 Rauschen in der berechneten Geschwindigkeit und Höhe

Die aus der Beschleunigung berechnete Geschwindigkeit ist weniger verrauscht, da die Integration eine glättende Wirkung hat. Die daraus berechnete Höhe kann jedoch akkumulierte Fehler aus der Geschwindigkeit enthalten, was sie potenziell verrauschter macht.

Jede dieser Antworten öffnet die Tür für tiefere Analysen und genauere Kalibrierungsmethoden, um die Datenqualität weiter zu verbessern.

*Das klingt schon sehr viel besser. Offensichtlich darf man nicht zu viele Fragen auf einmal stellen.*

*Allerdings erwähnt sie den Ruck nicht:*

## 3 Frage

Überleg noch mal, was ein dreieckiger Beschleunigungsverlauf bedeutet.

### 3.1 Antwort

Ein dreieckiger Beschleunigungsverlauf zeigt tatsächlich, dass die Beschleunigung zunächst zunimmt und dann wieder abnimmt oder umgekehrt. Das bedeutet, dass die Änderungsrate der Beschleunigung, also die Ruck genannt wird, konstant ist. Ruck ist die Ableitung der Beschleunigung nach der Zeit und gibt an, wie schnell sich die Beschleunigung selbst ändert.

#### 3.1.1 Physikalische Bedeutung dreieckiger Beschleunigungsphasen

Dreieckige Beschleunigungsphasen sind typisch für Bewegungen, bei denen die Antriebskraft über eine bestimmte Zeit linear zu- oder abnimmt und dann wieder in umgekehrter Weise ab- oder zunimmt. Dies kann bei mechanischen Systemen auftreten, in denen Kräfte gesteuert rampenförmig verändert werden. Zum Beispiel könnte in einem Fahrstuhl das Antriebssystem so gestaltet sein, dass es langsam die Kraft erhöht, um den Fahrstuhl zu beschleunigen und dann die Kraft reduziert, um eine gleichmäßige Fahrt zu ermöglichen, bevor es komplett stoppt.

#### 3.1.2 Praktische Beispiele

In einem Fahrstuhlexperiment könnte ein dreieckiger Beschleunigungsverlauf darauf hindeuten, dass der Fahrstuhl sanft beschleunigt und ebenso sanft wieder abgebremst wird, um eine angenehme Fahrt zu gewährleisten und ruckartige Bewegungen zu vermeiden. Der konstante Ruck während dieser Phasen bedeutet, dass die Steuerung der Geschwindigkeitsänderung (also der Beschleunigung) in einer kontrollierten und vorhersagbaren Weise erfolgt.

Dieses Verhalten wird oft in automatisierten Systemen verwendet, um die mechanische Belastung zu minimieren und die Lebensdauer der verwendeten Maschinen zu maximieren. In Matlab könnten solche Daten analysiert werden, indem man die zweite Ableitung der Geschwindigkeit (also die erste Ableitung der Beschleunigung) betrachtet, um die konstanten Ruckphasen zu identifizieren und zu analysieren.

*Sie weiß es doch ...*

# 17 Texterkennung

April 25, 2024

## 1 Frage

Schreiben Sie die Ziffern 1,2,3,...8,9,0 zehn mal auf ein Tablet oder zur Not auf ein Stück Papier (möglichst mit horizontalen und vertikalen Hilfslinien).

Alle Ziffern müssen dabei vollständig innerhalb ihrer (gedachten) Hilfslinienkästchen liegen.

Scannen Sie die Schriftprobe ein (zur Not fotografieren Sie sie möglichst verzerrungsfrei) und erzeugen ein Bild von etwa  $1000 \times 1000$  Pixeln.

Lesen Sie das Bild ein und stellen Sie es dar.

Speichern Sie die Anzahl der Zeilen und die Anzahl der Spalten des Bildes in den Variablen `n_zeilen` und `n_spalten` ab.

Werden die vertikalen und horizontalen Linien auch wirklich vertikal und horizontal dargestellt?

### 1.1 Antwort

Für die Aufgabe, die Sie beschrieben haben, benötigen Sie ein Skript, das ein Bild von handgeschriebenen Ziffern einliest, darstellt und die Anzahl der Zeilen und Spalten des Bildes speichert. Hier ist ein Beispiel für ein MATLAB-Skript, das diese Aufgaben erledigen könnte:

```
% Bild einlesen
bild = imread('pfad_zum_bild.jpg'); % Ersetzen Sie 'pfad_zum_bild.jpg' mit dem tatsächlichen

% Bild darstellen
figure; % Erzeugt eine neue Figur
imshow(bild); % Zeigt das Bild an
title('Eingelesenes Bild von Ziffern');

% Anzahl der Zeilen und Spalten speichern
n_zeilen = size(bild, 1);
n_spalten = size(bild, 2);

% Ausgabe der Dimensionen des Bildes
disp(['Anzahl der Zeilen: ', num2str(n_zeilen)]);
disp(['Anzahl der Spalten: ', num2str(n_spalten)]);
```

Dieses Skript geht davon aus, dass das Bild bereits in einem geeigneten Format (z.B. JPEG) und der korrekten Auflösung vorliegt. Sie sollten den Bildpfad entsprechend anpassen, wo das Bild gespeichert ist.

Was die Überprüfung der Ausrichtung der Linien im Bild angeht, so könnte dies visuell oder durch eine Bildanalyse erfolgen, die beispielsweise Kantenerkennung nutzt, um sicherzustellen, dass die Linien tatsächlich horizontal und vertikal sind. Wenn Sie Hilfe bei der Erweiterung des Skripts für solche Analysen benötigen, lassen Sie es mich wissen!

*Passt.*

## 2 Frage

Wandeln Sie das Bild in Graustufen um und stellen Sie es dar.

Warum müssen Sie dabei auch eine graue Farbtabelle auswählen?

### 2.1 Antwort

Um ein Bild in MATLAB in Graustufen umzuwandeln und darzustellen, nutzen Sie folgende Schritte im Code. Die Auswahl einer grauen Farbtabelle ist wichtig, weil `imshow` standardmäßig eine Farbtabelle (`colormap`) benutzt, die der Datenart des Bildes entspricht. Bei Graustufenbildern sollten die Pixelwerte direkt als Intensitäten von Schwarz bis Weiß interpretiert werden. Wenn keine passende Farbtabelle gewählt wird, könnten die Graustufen fälschlicherweise mit Farben dargestellt werden, was zu einer irreführenden visuellen Interpretation führen kann.

Hier ist das MATLAB-Skript, um ein Bild zu Graustufen zu konvertieren und korrekt anzuzeigen:

```
% Bild einlesen (Annahme: Bild liegt bereits im Arbeitsverzeichnis)
bild = imread('pfad_zum_bild.jpg'); % Pfad zum Bild anpassen

% Bild in Graustufen umwandeln
bild_grau = rgb2gray(bild);

% Bild in Graustufen darstellen
figure; % Erzeugt eine neue Figur
imshow(bild_grau); % Zeigt das Bild in Graustufen an
title('Bild in Graustufen');
colormap gray; % Setzt die Farbtabelle auf Graustufen

% Überprüfen der Farbtabelle
disp('Aktuell verwendete Farbtabelle:');
disp(colormap);
```

Das Skript nimmt an, dass das ursprüngliche Bild in Farbe ist. Falls das Bild bereits in Graustufen vorliegt, ist der Schritt `rgb2gray` nicht notwendig. Die Funktion `colormap gray` sorgt dafür, dass die Intensitätswerte des Bildes auf einer Skala von Schwarz bis Weiß abgebildet werden, was für eine korrekte Darstellung der Graustufen erforderlich ist. Ohne diese Anweisung könnte MATLAB eine Standard-Farbtabelle verwenden, die zu einer farbigen Darstellung von eigentlich grauen Bildern führen könnte.

*Klappt wunderbar. Und natürlich verwendet sie konsequenterweise `rgb2gray`.*

## 3 Frage

Entfernen Sie die Hilfslinie und einen eventuellen Grauschleier, indem Sie alle Pixel mit einem Wert größer als eine willkürliche Grenze auf 255 setzen. Starten Sie mit einem Grenzwert von 200 und verändern Sie ihn so lange, bis keine Hilfslinien mehr vorhanden sind, die Ziffern aber möglichst unbeschadet bleiben. Zeichnen Sie das Ergebnis.

Wie können Sie bei der Hilfslinientfernung ohne eine for-Schleife auskommen?

### 3.1 Antwort

In MATLAB können Sie die Hilfslinien und einen möglichen Grauschleier aus einem Bild entfernen, indem Sie alle Pixel mit einem Wert über einer bestimmten Grenze auf den maximalen Wert (255 für ein 8-Bit Graustufenbild) setzen. Dies kann effizient ohne eine for-Schleife mit Matrixoperationen durchgeführt werden, die in MATLAB besonders schnell sind.

Hier ist ein Beispielcode, der zeigt, wie Sie diese Operation durchführen können:

```
% Bild einlesen und in Graustufen umwandeln (Annahme: das Bild ist in Farbe)
bild = imread('pfad_zum_bild.jpg'); % Pfad zum Bild anpassen
bild_grau = rgb2gray(bild);

% Grenzwert setzen
grenzwert = 200;

% Entfernen der Hilfslinien ohne for-Schleife
bild_bereinigt = bild_grau; % Kopie des Graustufenbilds erstellen
bild_bereinigt(bild_bereinigt > grenzwert) = 255;

% Ergebnis darstellen
figure;
imshow(bild_bereinigt);
title('Bild nach Entfernung der Hilfslinien');
colormap gray; % Sicherstellen, dass die Darstellung in Graustufen erfolgt
```

#### 3.1.1 Erklärung

- **Matrixoperationen:** Der Ausdruck `bild_bereinigt(bild_bereinigt > grenzwert) = 255;` ist eine vektorisierte Operation in MATLAB. Er setzt alle Pixelwerte, die den Grenzwert überschreiten, direkt auf 255. Diese Art der Operation ist sehr effizient, da sie intern optimiert wird und nicht jede einzelne Komponente der Matrix einzeln in einer Schleife durchläuft.

#### 3.1.2 Anpassung des Grenzwerts

Wenn nach der ersten Anpassung noch Hilfslinien sichtbar sind oder die Ziffern zu stark beeinträchtigt werden, sollten Sie den Grenzwert schrittweise anpassen. Verringern Sie den Grenzwert, um weniger aggressive Korrekturen zu machen, oder erhöhen Sie ihn, um mehr des Grauschleiers zu entfernen. Sie können dies tun, indem Sie den Wert von `grenzwert` im Skript ändern und das Ergebnis erneut überprüfen.

Das interaktive Anpassen des Grenzwerts und das sofortige Feedback durch die Darstellung des Ergebnisses ist eine effektive Methode, um die Qualität der Bildbearbeitung sicherzustellen.

*Schön erklärt!*

## 4 Frage

Um den Kontrast zu erhöhen, ersetzen Sie alle Pixelwerte, die gleich oder kleiner als die in der vorherigen Aufgabe gewählte Grenze sind, durch den Wert null. Stellen Sie das Bild dar.

Welche Werte besitzt das Bild jetzt nur noch?

Könnte man seinen Zahlentyp anpassen, um Speicherplatz zu sparen?

### 4.1 Antwort

Um den Kontrast zu erhöhen, indem alle Pixelwerte, die kleiner oder gleich einem bestimmten Grenzwert sind, auf null gesetzt werden, können Sie weiterhin die effizienten Matrixoperationen in MATLAB nutzen. Der Code für diese Aufgabe sieht dann so aus:

```
% Bild einlesen und in Graustufen umwandeln (Annahme: das Bild ist in Farbe)
bild = imread('pfad_zum_bild.jpg'); % Pfad zum Bild anpassen
bild_grau = rgb2gray(bild);

% Grenzwert setzen, unter dem die Pixel auf 0 gesetzt werden sollen
grenzwert = 200;

% Kontrast erhöhen
bild_kontrast = bild_grau;
bild_kontrast(bild_kontrast > grenzwert) = 255;
bild_kontrast(bild_kontrast <= grenzwert) = 0;

% Ergebnis darstellen
figure;
imshow(bild_kontrast);
title('Bild mit erhöhtem Kontrast');
colormap gray; % Sicherstellen, dass die Darstellung in Graustufen erfolgt
```

#### 4.1.1 Erklärung

Dieses Skript setzt alle Pixelwerte, die den Grenzwert überschreiten, auf 255 und alle anderen auf 0. Dadurch wird der Kontrast des Bildes deutlich erhöht, da nur noch die zwei Extremwerte 0 und 255 vorhanden sind.

#### 4.1.2 Bildwerte und Speicherplatz

Das bearbeitete Bild besitzt jetzt nur noch zwei eindeutige Pixelwerte: - **0**: Für dunklere Bereiche (alle Originalwerte  $\leq 200$ ) - **255**: Für hellere Bereiche (alle Originalwerte  $> 200$ )

### 4.1.3 Speicherplatzeinsparungen

Da das Bild nur noch zwei Werte enthält, könnte man theoretisch den benötigten Speicherplatz reduzieren, indem man anstatt eines 8-bit Graustufenformats ein binäres Format (1-bit pro Pixel) verwendet. In MATLAB und vielen anderen Bildverarbeitungs-umgebungen ist dies jedoch nicht standardmäßig vorgesehen, da die meisten Operationen und Funktionen mindestens 8-bit Werte erwarten. Trotzdem wäre es technisch möglich, durch eine Anpassung auf ein binäres Format Speicherplatz zu sparen, insbesondere wenn das Bild in einer Umgebung verarbeitet oder gespeichert wird, die dies unterstützt.

Für die meisten Anwendungen und zur Vereinfachung der Handhabung bleibt es jedoch praktikabel, das Bild als 8-bit Graustufenbild zu belassen, auch wenn nur zwei Werte genutzt werden. Dies vermeidet Kompatibilitätsprobleme mit Software und Bibliotheken, die möglicherweise keine binären Bilder unterstützen.

*Sehr salomonisch!*

## 5 Frage

Jetzt beginnt die Segmentierung. Initialisieren Sie einen leeren Vektor `kanten_zeilen`, in dem Sie diejenigen Zeilenindizes abspeichern werden, die vor dem Beginn und hinter dem Ende einer Ziffernzeile liegen. Beginnen Sie eine Schleife von der zweiten bis zur vorletzten Bildzeile und verwenden Sie die folgende if-Bedingung:

```
if ...
all (bild(zeile, :)) && ...
(~all (bild(zeile + 1, :)) || ...
~all (bild(zeile - 1, :)))
```

Erläutern Sie ausführlich, welche Fälle die if-Bedingung findet.

### 5.1 Antwort

Für die Segmentierung von Ziffern in einem Bild, insbesondere wenn Sie nach horizontalen Zeilen suchen, bei denen der Übergang von einer komplett gefüllten (weißen) zu einer nicht komplett gefüllten (teilweise oder ganz schwarzen) Zeile oder umgekehrt stattfindet, können Sie die vorgeschlagene if-Bedingung verwenden. Dieser Ansatz ist nützlich, um die Grenzen der Ziffern zu identifizieren.

#### 5.1.1 MATLAB-Code-Snippet

Hier ist der MATLAB-Code, um den leeren Vektor zu initialisieren und die Schleife zu beginnen:

```
% Annahme: bild_kontrast ist das Bild nach Kontrasterhöhung
bild = bild_kontrast; % Dies könnte das Ergebnis aus der vorherigen Aufgabe sein.

% Initialisieren Sie einen leeren Vektor für Kantenzeilen
kanten_zeilen = [];

% Schleife durch das Bild von der zweiten bis zur vorletzten Zeile
```

```

for zeile = 2:size(bild, 1) - 1
    if all(bild(zeile, :)) && (~all(bild(zeile + 1, :)) || ~all(bild(zeile - 1, :)))
        kanten_zeilen = [kanten_zeilen, zeile];
    end
end

```

### 5.1.2 Erläuterung der if-Bedingung

Die if-Bedingung in dieser Schleife dient dazu, die Zeilen zu erkennen, die als “Kanten” zwischen Ziffern und Hintergrund dienen. Die Bedingung wird in drei Teile zerlegt:

1. `all(bild(zeile, :))`:
  - Diese Bedingung überprüft, ob alle Pixel in der aktuellen Zeile den Wert 255 haben (da das Bild nur 0 und 255 als Werte hat, bedeutet dies, dass die Zeile komplett weiß ist).
2. `~all(bild(zeile + 1, :))`:
  - Dies überprüft, ob in der Zeile direkt unter der aktuellen Zeile nicht alle Pixel den Wert 255 haben. Das bedeutet, dass mindestens ein Pixel der Zeile darunter den Wert 0 hat, also schwarz ist.
3. `~all(bild(zeile - 1, :))`:
  - Ähnlich überprüft dies, ob in der Zeile direkt über der aktuellen Zeile nicht alle Pixel den Wert 255 haben.

Die gesamte Bedingung `all(bild(zeile, :)) && (~all(bild(zeile + 1, :)) || ~all(bild(zeile - 1, :)))` wird wahr, wenn: - Die aktuelle Zeile komplett weiß ist, und - Die Zeile darüber oder darunter mindestens ein schwarzes Pixel enthält.

### 5.1.3 Was findet diese Bedingung?

Diese Bedingung findet die Zeilen, die an den oberen und unteren Rändern einer Ziffer liegen. Wenn eine Ziffer beginnt oder endet, wechselt die Zeile von komplett weiß zu mindestens teilweise schwarz (oder umgekehrt). Diese Zeilen sind kritisch für die Segmentierung, da sie die Grenzen angeben, an denen eine Ziffer startet oder endet. Diese Technik ist besonders effektiv bei Bildern mit deutlich getrennten Ziffern und einem kontrastreichen Hintergrund.

*Ich bin echt beeindruckt!*

## 6 Frage

Wenn die if-Bedingung erfüllt ist, setzen Sie alle Pixel der aktuellen Zeile auf den Wert 42 und hängen den Zeilenindex an `kanten_zeilen` an.

Was steht dann letztendlich in `kanten_zeilen`?

### 6.1 Antwort

Wenn die if-Bedingung erfüllt ist, wird in der Zeile des Bildes eine visuelle Markierung (hier durch den Wert 42) gesetzt und der Index dieser Zeile wird zum Vektor `kanten_zeilen` hinzugefügt. Dies hilft dabei, die genauen Positionen im Bild zu identifizieren, an denen eine Ziffer beginnt oder endet, oder wo ein Übergang von einem komplett weißen Bereich zu einem Bereich mit mindestens einem

schwarzen Pixel (und umgekehrt) stattfindet. Dies ist besonders nützlich für die Segmentierung und Analyse von Bildern in weiteren Verarbeitungsschritten.

### 6.1.1 Erweiterter MATLAB-Code

Hier ist der erweiterte Code, der zeigt, wie Sie diese Operation durchführen:

```
% Angenommen, bild_kontrast ist das Bild nach Kontrasterhöhung
bild = bild_kontrast; % Dies könnte das Ergebnis aus der vorherigen Aufgabe sein.

% Initialisieren Sie einen leeren Vektor für Kantenzeilen
kanten_zeilen = [];

% Schleife durch das Bild von der zweiten bis zur vorletzten Zeile
for zeile = 2:size(bild, 1) - 1
    if all(bild(zeile, :)) && (~all(bild(zeile + 1, :)) || ~all(bild(zeile - 1, :)))
        bild(zeile, :) = 42; % Setze alle Pixel der Zeile auf 42
        kanten_zeilen = [kanten_zeilen, zeile]; % Füge Zeilenindex zu kanten_zeilen hinzu
    end
end

% Optional: Zeige das modifizierte Bild an
figure;
imshow(bild, []);
title('Modifiziertes Bild mit markierten Kantenzeilen');
```

### 6.1.2 Was steht in kanten\_zeilen?

Der Vektor `kanten_zeilen` enthält die Indizes aller Zeilen, die die Kriterien der `if`-Bedingung erfüllen. Diese Zeilenindizes repräsentieren die Übergänge zwischen komplett weißen und nicht komplett weißen Zeilen. Da die Zeilen mit dem Wert 42 markiert wurden, zeigt `kanten_zeilen` effektiv die Positionen an, an denen im Bild visuelle “Kanten” oder Grenzen von Ziffern zu sehen sind.

### 6.1.3 Interpretation

- In `kanten_zeilen` sind somit die Indizes der Zeilen gespeichert, wo:
  - Eine komplett weiße Zeile direkt an eine Zeile grenzt, die mindestens ein schwarzes Pixel enthält.
- Diese Indizes sind sehr nützlich für Aufgaben wie:
  - Automatische Texterkennung (OCR),
  - Bildsegmentierung für weiterführende Analysen,
  - Maschinelles Lernen, wo genaue Bildsegmente als Eingabe benötigt werden.

Insgesamt hilft dieser Ansatz dabei, die strukturellen Informationen des Bildes zu verstehen und bietet eine Basis für die automatisierte Verarbeitung und Analyse von bildbasierten Daten.

*Alles sehr schön ausgedrückt.*

## 7 Frage

Führen Sie das gleiche Vorgehen für die Spalten des Bildes durch und stellen Sie das Bild (vergrößert) dar. Analysieren Sie genau, ob die Begrenzungslinien an den richtigen Stellen zu finden sind.

Wie viele Begrenzungslinien sind es?

Warum wohl?

### 7.1 Antwort

Um eine ähnliche Operation für die Spalten des Bildes durchzuführen, müssen wir den Prozess anpassen, um die vertikalen Übergänge zu markieren. Dies ist hilfreich, um die Begrenzungen von Ziffern innerhalb einer Zeile zu erkennen. Der MATLAB-Code wird angepasst, um die Spalten statt der Zeilen zu durchlaufen. Hierbei wird auch geprüft, ob eine Spalte komplett weiß ist und entweder die vorherige oder die nachfolgende Spalte nicht komplett weiß ist.

#### 7.1.1 MATLAB-Code-Snippet

Hier ist der Code, um die gleiche Vorgehensweise auf die Spalten anzuwenden:

```
% Angenommen, bild_kontrast ist das Bild nach Kontrasterhöhung
bild = bild_kontrast; % Dies könnte das Ergebnis aus der vorherigen Aufgabe sein.

% Initialisieren Sie einen leeren Vektor für Kanten in Spalten
kanten_spalten = [];

% Schleife durch das Bild von der zweiten bis zur vorletzten Spalte
for spalte = 2:size(bild, 2) - 1
    if all(bild(:, spalte)) && (~all(bild(:, spalte + 1)) || ~all(bild(:, spalte - 1)))
        bild(:, spalte) = 42; % Setze alle Pixel der Spalte auf 42
        kanten_spalten = [kanten_spalten, spalte]; % Füge Spaltenindex zu kanten_spalten hinzu
    end
end

% Bild darstellen (vergrößert)
figure;
imshow(bild, []);
title('Modifiziertes Bild mit markierten Kanten in den Spalten');
```

#### 7.1.2 Analyse der Begrenzungslinien

Nachdem das Bild verarbeitet und die Begrenzungslinien markiert wurden, müssen Sie das Bild visuell überprüfen, um festzustellen, ob die Begrenzungslinien korrekt gesetzt wurden.

#### 7.1.3 Warum gibt es eine bestimmte Anzahl von Begrenzungslinien?

Die Anzahl der Begrenzungslinien, die gefunden werden, hängt davon ab:

- **Wie viele Ziffern im Bild sind:** Jede Ziffer sollte typischerweise zwei Begrenzungslinien haben (eine am Anfang und eine am Ende der Ziffer).
- **Die Qualität der Segmentation:** Wenn die Ziffern sehr nahe beieinander liegen oder sich überlappen, könnte der Algorithmus Schwierigkeiten haben, korrekte Grenzen zu setzen.
- **Kontrast und Schärfe der Ziffern:** Niedriger Kontrast oder unscharfe Kanten können dazu führen, dass der Algorithmus nicht alle Übergänge erkennt.

#### 7.1.4 Wie viele Linien sollten erwartet werden?

Wenn das Bild klar ist und die Ziffern gut getrennt sind, sollten Sie zwei Begrenzungslinien für jede Ziffer erwarten. Wenn das Bild 10 Ziffern enthält, sollte der Algorithmus 20 Begrenzungslinien identifizieren, vorausgesetzt jede Ziffer ist individuell erkennbar.

Die genaue Überprüfung und gegebenenfalls die Anpassung der Schwelle oder des Algorithmus könnten notwendig sein, um die bestmögliche Segmentierung zu erzielen.

*Sehr richtig.*

## 8 Frage

Jetzt wird es etwas schwieriger. Sie möchten die einzelnen Ziffern in einem cell array namens ziffern ablegen.

Dazu ermitteln Sie die Länge von kanten\_zeilen und kanten\_spalten und schreiben eine Doppelschleife über jeweils jedes zweite Element der beiden Vektoren:

```
for kante_zeile = 1 : 2 : n_kanten_zeilen
for kante_spalte = 1 : 2 : n_kanten_spalten
ziffern{(kante_zeile + 1)/2, (kante_spalte + 1)/2} = ...
bild ( ...
kanten_zeilen(kante_zeile) + 1 : ...
kanten_zeilen(kante_zeile + 1) - 1, ...
kanten_spalten(kante_spalte) + 1: ...
kanten_spalten(kante_spalte + 1) - 1);
end
end
```

Erläutern Sie ausführlich jeden Teilausdruck der Doppelschleife.

(Denken Sie dazu bitte in Ruhe erst einmal selbst darüber nach, wie Sie an die einzelnen Ziffern herankommen und wie Sie sie dann in das cell array eintüten würden ...)

### 8.1 Antwort

Die vorgeschlagene Doppelschleife ist ein effizienter Ansatz, um einzelne Ziffern aus einem Bild herauszuschneiden und sie in einem Cell Array zu speichern. Jede Ziffer befindet sich zwischen

den Kantenlinien, die zuvor identifiziert wurden. Lassen Sie uns jeden Teil des Codes detailliert durchgehen, um die Funktionsweise zu verstehen:

### 8.1.1 Schleifenstruktur

Die beiden verschachtelten Schleifen durchlaufen die Arrays `kanten_zeilen` und `kanten_spalten`. Diese Arrays enthalten die Indizes der Zeilen und Spalten im Bild, die die Ränder der Ziffern markieren. Da jede Ziffer von zwei Linien (oben und unten für Zeilen, links und rechts für Spalten) umgeben ist, erfolgt das Iterieren über jedes zweite Element:

```
for kante_zeile = 1 : 2 : n_kanten_zeilen
    for kante_spalte = 1 : 2 : n_kanten_spalten
```

- `1:2:n_kanten_zeilen` und `1:2:n_kanten_spalten`: Start bei Index 1 und nehme jeden zweiten Index bis zum Ende des Arrays. Dies überspringt jedes zweite Element, um nur die Startpunkte jeder Ziffer zu berücksichtigen (da das Ende einer Ziffer der Anfang der nächsten ist).

### 8.1.2 Extraktion und Speicherung der Ziffern

Innerhalb der Schleifen extrahiert der Code einen Teil des Bildes, der einer Ziffer entspricht:

```
ziffern{(kante_zeile + 1)/2, (kante_spalte + 1)/2} = ...
    bild ( ...
        kanten_zeilen(kante_zeile) + 1 : ...
        kanten_zeilen(kante_zeile + 1) - 1, ...
        kanten_spalten(kante_spalte) + 1: ...
        kanten_spalten(kante_spalte + 1) - 1);
```

- `{(kante_zeile + 1)/2, (kante_spalte + 1)/2}`: Dieses Konstrukt dient dazu, die richtige Position im Cell Array `ziffern` zu indizieren. Da die Schleifenvariablen in Schritten von zwei laufen, muss der Index für das Cell Array angepasst werden, damit er bei 1 beginnt und inkrementell weiterläuft.
- `bild(...)`: Dieser Teil extrahiert ein Teilbild oder Subbild aus dem Gesamtbild `bild`. Die Grenzen des Subbilds werden durch die Kanten bestimmt:
  - `kanten_zeilen(kante_zeile) + 1 : kanten_zeilen(kante_zeile + 1) - 1`: Dies gibt die Zeilenindizes des Bildes von der Zeile direkt nach der aktuellen Kante bis zur Zeile direkt vor der nächsten Kante.
  - `kanten_spalten(kante_spalte) + 1 : kanten_spalten(kante_spalte + 1) - 1`: Dies gibt die Spaltenindizes des Bildes von der Spalte direkt nach der aktuellen Kante bis zur Spalte direkt vor der nächsten Kante.

### 8.1.3 Zweck des Cell Arrays `ziffern`

Das Cell Array `ziffern` wird verwendet, um die extrahierten Ziffernbilder zu speichern. Jede Zelle im Array enthält ein Bild einer Ziffer, wodurch sie getrennt von den anderen verwaltet und bei Bedarf einzeln bearbeitet oder analysiert werden kann.

### 8.1.4 Zusammenfassung

Durch diesen Ansatz wird jedes Bildsegment, das eine Ziffer enthält, aus dem Gesamtbild herausgeschnitten und in einer strukturierten Weise in einem Cell Array gespeichert. Dies erleichtert die weitere Verarbeitung jeder einzelnen Ziffer für Aufgaben wie maschinelles Lernen oder OCR (Optical Character Recognition).

*Vorbildliche Erläuterungen!*

*Und sie lobt mich sogar: “Die vorgeschlagene Doppelschleife ist ein effizienter Ansatz, um einzelne Ziffern aus einem Bild herauszuschneiden und sie in einem Cell Array zu speichern.”*

## 9 Frage

Sie möchten jetzt bei jeder einzelnen Ziffer die umrandenden weißen Zeilen und Spalten entfernen.

Bestimmen Sie dazu die Anzahl der Zeilen `n_beispiele` und Spalten `n_ziffern` des cell arrays und beginnen Sie eine Doppelschleife mit den Laufindizes `beispiel` und `ziffer` (Achtung: Machen Sie sich bitte den Unterschied zwischen dem Cell Array `ziffern` und dem Laufindex `ziffer` klar) über alle Beispiele und Ziffern.

In der Doppelschleife verwenden Sie die folgenden Befehle

```
z = ziffern{beispiel, ziffer};
```

```
z(:, all(z)) = [];
```

```
z(all(z'), :) = [];
```

```
ziffern{beispiel, ziffer} = z;
```

Erläutern Sie jeden einzelnen Befehl.

### 9.1 Antwort

In der gestellten Aufgabe geht es darum, jede Ziffer im Bild von unnötigen weißen Rändern (Zeilen und Spalten, die nur weiße Pixel enthalten) zu befreien. Dies wird oft als “Trimming” oder “Cropping” der leeren Ränder bezeichnet, um die Ziffern kompakter zu machen und die Effizienz nachfolgender Bildverarbeitungsprozesse zu steigern. Hier ist eine detaillierte Erklärung der einzelnen Befehle in der Doppelschleife:

#### 9.1.1 Schleifenstruktur

```
n_beispiele = size(ziffern, 1);
```

```
n_ziffern = size(ziffern, 2);
```

```
for beispiel = 1:n_beispiele
```

```
    for ziffer = 1:n_ziffern
```

- **n\_beispiele** und **n\_ziffern**: Diese Variablen speichern die Anzahl der Zeilen und Spalten des Cell Arrays `ziffern`, was die Gesamtanzahl der Beispiele und Ziffern in Ihrem Datensatz repräsentiert.

- `for beispiel = 1:n_beispiele` und `for ziffer = 1:n_ziffern`: Diese Schleifen durchlaufen alle Zellen des Cell Arrays `ziffern`, sodass jede Ziffer bearbeitet wird.

### 9.1.2 Extraktion und Bearbeitung einer Ziffer

```
z = ziffern{beispiel, ziffer};
```

- `z = ziffern{beispiel, ziffer}`: Hier wird die Ziffer aus der entsprechenden Zelle des Cell Arrays in eine temporäre Variable `z` geladen. `z` enthält das Bild der Ziffer als Matrix von Pixelwerten.

### 9.1.3 Entfernen leerer Spalten

```
z(:, all(z)) = [];
```

- `z(:, all(z))`: Dieser Ausdruck überprüft jede Spalte der Ziffer `z`. Die Funktion `all(z)` gibt einen logischen Vektor zurück, bei dem jeder Wert `true` ist, wenn alle Pixel in einer Spalte weiß sind (angenommen, weiß ist durch den Wert 255 repräsentiert und das Bild ist binarisiert).
- `z(:, all(z)) = []`: Leere Spalten (nur weiße Pixel enthaltend) werden aus der Matrix entfernt.

### 9.1.4 Entfernen leerer Zeilen

```
z(all(z'), :) = [];
```

- `z(all(z'), :)`: Ähnlich wie oben, jedoch für Zeilen. `z'` ist die transponierte Matrix von `z`, sodass `all(z')` für jede Zeile in `z` überprüft, ob alle Pixel weiß sind.
- `z(all(z'), :) = []`: Leere Zeilen werden entfernt.

### 9.1.5 Speichern der bearbeiteten Ziffer zurück ins Cell Array

```
ziffern{beispiel, ziffer} = z;
```

- `ziffern{beispiel, ziffer} = z`: Die bearbeitete Ziffer `z`, von der nun alle unnötigen weißen Ränder entfernt wurden, wird zurück in das Cell Array an der gleichen Position gespeichert.

### 9.1.6 Zusammenfassung

Diese Operationen dienen dazu, das Bild jeder Ziffer zu bereinigen, indem unnötige weiße Ränder entfernt werden, was typischerweise die Effektivität der Bilderkennung und -verarbeitung in nachfolgenden Schritten, wie maschinelles Lernen oder OCR, verbessert. Durch die Reduzierung der Bildgröße auf nur die relevante Ziffer wird auch Speicherplatz gespart und die Verarbeitungsgeschwindigkeit kann erhöht werden.

*Ich find's gut!*

## 10 Frage

Sie möchten jetzt jede Ziffer auf  $20 \times 20$  Pixel skalieren.

Verwenden Sie dazu in einer weiteren Doppelschleife den Befehl `imresize`, um jede einzelne Ziffer zu skalieren.

Stellen Sie exemplarisch irgendeine Ziffer dar.

Warum ist das Bild jetzt nicht mehr schwarz-weiß?

## 10.1 Antwort

Um jede Ziffer im Bild auf die gewünschte Größe von 20x20 Pixel zu skalieren, wird in MATLAB der Befehl `imresize` verwendet. Diese Funktion ist sehr nützlich für das Skalieren von Bildern, kann aber zu leichten Veränderungen in den Pixelwerten führen, was die Farbintegrität betrifft, insbesondere bei binären Bildern. Hier sehen wir uns den entsprechenden Code und die Erklärung an:

### 10.1.1 Skalierungscode

```
n_beispiele = size(ziffern, 1);
n_ziffern = size(ziffern, 2);

for beispiel = 1:n_beispiele
    for ziffer = 1:n_ziffern
        z = ziffern{beispiel, ziffer};
        z = imresize(z, [20 20]); % Skaliere die Ziffer auf 20x20 Pixel
        ziffern{beispiel, ziffer} = z;
    end
end

% Exemplarische Darstellung einer Ziffer
figure;
imshow(ziffern{1, 1}, []);
title('Skalierte Ziffer');
```

### 10.1.2 Erklärung von `imresize`

- `imresize(z, [20 20])`: Diese Funktion skaliert das Bild `z` auf die Größe 20x20 Pixel. Die Funktion verwendet standardmäßig Bilineare Interpolation, wenn keine spezifische Methode angegeben wird. Diese Methode berechnet den Wert eines Pixels basierend auf einer gewichteten Kombination der vier nächsten Pixel aus dem Originalbild, was zu neuen Grauwerten führen kann.

### 10.1.3 Warum ist das Bild jetzt nicht mehr strikt schwarz-weiß?

Das resultierende Bild ist nach dem Skalieren möglicherweise nicht mehr strikt schwarz-weiß aus den folgenden Gründen:

1. **Interpolationseffekte**: Die Standardinterpolationsmethode (bilinear oder bikubisch) in `imresize` führt dazu, dass neue Pixelwerte als gewichtete Mittelwerte der umgebenden Originalpixel berechnet werden. Wenn das Originalbild Schwarz-Weiß-Pixel enthält (typischerweise 0 für Schwarz und 255 für Weiß), können die interpolierten Pixel Grauwerte zwischen 0

und 255 annehmen. Dies tritt auf, weil die Pixel an den Rändern von schwarzen und weißen Bereichen Mischwerte zugewiesen bekommen.

2. **Antialiasing:** `imresize` kann optional ein Antialiasing durchführen, um die Bildqualität zu verbessern, was ebenfalls zur Graustufenerzeugung beiträgt.

#### 10.1.4 Lösungen

Um sicherzustellen, dass das Bild nach der Skalierung schwarz-weiß bleibt, können Sie:

- Eine “nearest” Interpolation in `imresize` verwenden, die einfach den nächsten Nachbarpixelwert übernimmt und keine Mischung vornimmt:

```
z = imresize(z, [20 20], 'nearest');
```

- Nach der Interpolation eine Binarisierung durchführen, um alle Pixel, die einen bestimmten Schwellenwert überschreiten, auf Weiß zu setzen und alle anderen auf Schwarz:

```
z = imbinarize(z);
```

Diese Anpassungen helfen dabei, die Bildqualität zu kontrollieren und sicherzustellen, dass die Bilder nach der Skalierung den gewünschten schwarz-weißen Charakter behalten.

*Hilfreich! imbinarize kannte ich noch nicht.*

## 11 Frage

Sie möchten jetzt ein weiteres cell array mit dem Namen `mittelwerte` erzeugen, das die Mittelwertbilder aller zehn Ziffern enthält. Dazu summieren Sie in einer weiteren Doppelschleife (die äußere Schleife läuft über alle Ziffern, die innerere über alle Beispiele) jeweils die Pixelwerte aller Beispiele einer Ziffer und teilen anschließend (noch innerhalb der Schleife über alle Ziffern) durch die Anzahl der Beispiele. Stellen Sie exemplarisch das Mittelwertbild irgendeiner Ziffer dar.

Warum müssen Sie beim Addieren der Pixelwerte den Befehl `double` verwenden?

Warum müssen Sie in der äußeren Zifferschleife das aktuelle Mittelwert-Zellenfeld initialisieren?

### 11.1 Antwort

Um ein Cell Array namens `mittelwerte` zu erstellen, das die gemittelten Bilder aller zehn Ziffern enthält, ist ein methodischer Ansatz notwendig. Sie müssen zunächst alle Bilder einer bestimmten Ziffer aufsummieren und dann durch die Anzahl der Beispiele dividieren. Hier ist, wie Sie dies in MATLAB implementieren können:

#### 11.1.1 Schritt-für-Schritt-Anleitung

1. **Initialisierung des `mittelwerte` Cell Arrays:**
  - Dieses Array wird verwendet, um das gemittelte Bild jeder Ziffer zu speichern.
2. **Doppelschleife zur Berechnung der Mittelwerte:**
  - Die äußere Schleife läuft über jede Ziffer (0 bis 9).
  - Die innere Schleife läuft über alle Beispiele der aktuellen Ziffer.

### 11.1.2 MATLAB-Code

```
n_ziffern = 10; % Angenommen, es gibt 10 Ziffern (0-9)
n_beispiele = size(ziffern, 1); % Anzahl der Beispiele pro Ziffer
mittelwerte = cell(1, n_ziffern); % Initialisierung des Cell Arrays für Mittelwerte

for ziffer = 1:n_ziffern
    summe = zeros(20, 20); % Initialisiere die Summe als Matrix aus Nullen
    for beispiel = 1:n_beispiele
        z = double(ziffern{beispiel, ziffer}); % Umwandlung in double zur sicheren Pixeladdi
        summe = summe + z; % Addiere das Beispiel zur Summe
    end
    mittelwerte{ziffer} = summe / n_beispiele; % Berechne den Mittelwert und speichere ihn
end

% Exemplarische Darstellung des Mittelwertbildes der ersten Ziffer
figure;
imshow(mittelwerte{1}, []);
title('Mittelwertbild der Ziffer 1');
```

### 11.1.3 Warum double verwenden?

Der Befehl `double` wird verwendet, um sicherzustellen, dass die Pixelwerte korrekt addiert werden, ohne Überlaufprobleme oder Präzisionsverluste, die bei integer-Typen auftreten können. MATLAB arbeitet intern am effizientesten mit `double`, und viele Bildbearbeitungsfunktionen erwarten Eingaben in diesem Format. Dies ist besonders wichtig, wenn große Summen gebildet werden, da Integer-Typen möglicherweise nicht den benötigten Wertebereich abdecken können.

### 11.1.4 Warum initialisieren?

Die Initialisierung der Summenmatrix `summe` für jede Ziffer in der äußeren Schleife ist notwendig, um sicherzustellen, dass die Summation für jede Ziffer von Null beginnt. Ohne diese Initialisierung könnten frühere Werte die Ergebnisse verfälschen, da sonst auf bereits bestehenden Werten aufgebaut wird, was zu inkorrekten Berechnungen führt. Jede Ziffer benötigt eine "frische" Summe, um korrekte Durchschnittsbilder zu erzeugen.

### 11.1.5 Zusammenfassung

Dieser Ansatz bietet eine klare und strukturierte Methode, um die durchschnittlichen Bilder der Ziffern in einem Datensatz zu berechnen, was für die Bildverarbeitung und maschinelles Lernen von großem Nutzen sein kann. Solche Durchschnittsbilder sind hilfreich, um die allgemeine Form und Ausprägung der Ziffern zu visualisieren, was besonders in der Erkennung und Klassifizierung nützlich ist.

*Und wieder gelobt: "Dieser Ansatz bietet eine klare und strukturierte Methode ..." :-)*

## 12 Frage

Im Folgenden wollen Sie alle Beispielziffern klassifizieren lassen.

Warum ist das eigentlich ein bisschen geschummelt?

Was müssten Sie eigentlich klassifizieren lassen?

## 12.1 Antwort

Die Klassifizierung aller Beispielziffern aus demselben Datensatz, aus dem auch die Mittelwerte berechnet wurden, kann tatsächlich ein wenig “geschummelt” erscheinen. Hier sind die Gründe und die tatsächlich notwendigen Schritte für eine korrekte Klassifizierung:

### 12.1.1 Warum ist es “geschummelt”?

1. **Überanpassung (Overfitting):** Wenn Sie ein Modell oder eine Klassifizierungsmethode auf denselben Daten testen, auf denen Sie auch trainiert haben, besteht die Gefahr, dass das Modell zu gut auf diese spezifischen Daten angepasst ist. Es kann dann möglicherweise die Eigenschaften und Muster dieser speziellen Daten sehr gut erkennen, aber es ist unklar, wie gut es mit neuen, unbekanntem Daten umgehen kann. Das Modell könnte also einfach die spezifischen Merkmale dieser Trainingsbeispiele “auswendig lernen” anstatt zu generalisieren.
2. **Validierung der Leistung:** Wenn Sie die Genauigkeit oder Leistungsfähigkeit einer Klassifizierungsmethode beurteilen wollen, sollten Sie dies immer anhand von Daten tun, die nicht für das Training verwendet wurden. Andernfalls erhalten Sie möglicherweise ein verzerrtes Bild der tatsächlichen Effektivität Ihres Klassifizierungsansatzes.

### 12.1.2 Was müssten Sie stattdessen tun?

1. **Aufteilen des Datensatzes:** Der korrekte Ansatz wäre, den gesamten Datensatz in mindestens zwei Teile zu teilen – einen Trainingsdatensatz und einen Testdatensatz. In vielen Fällen wird auch ein dritter Teil, der Validierungsdatensatz, verwendet, um die Hyperparameter zu optimieren, bevor die endgültige Leistung auf dem Testdatensatz bewertet wird.
2. **Kreuzvalidierung:** Eine weitere Methode zur Überprüfung der Generalisierbarkeit und zur Vermeidung von Überanpassung ist die Kreuzvalidierung, typischerweise k-fache Kreuzvalidierung. Dabei wird der Datensatz in k Teile aufgeteilt, wobei jeder Teil einmal als Testdatensatz und die übrigen k-1 Teile als Trainingsdatensatz verwendet werden. Dieser Prozess wird k-mal wiederholt, wobei jedes Mal ein anderer Teil des Datensatzes als Testdatensatz verwendet wird. Dies hilft, eine robustere Schätzung der Modelleistung zu erhalten.
3. **Unabhängiger Testdatensatz:** Idealerweise haben Sie Zugriff auf einen komplett unabhängigen Datensatz, der nicht zur Entwicklung des Klassifizierungsmodells verwendet wurde. Dies könnte ein neu gesammelter Datensatz oder ein öffentlich verfügbarer Benchmark-Datensatz sein, der allgemein zur Bewertung von Klassifizierungsmodellen verwendet wird.

### 12.1.3 Zusammenfassung

Kurz gesagt, das Klassifizieren aller Beispielziffern, die zum Trainieren des Modells verwendet wurden, gibt Ihnen kein klares Bild von der Fähigkeit des Modells, mit neuen Daten umzugehen. Stattdessen sollten Sie das Modell mit neuen, unabhängigen Daten testen, um seine tatsächliche Leistung und Generalisierungsfähigkeit zu beurteilen. Dies ist ein entscheidender Schritt, um sicherzustellen, dass Ihr Klassifizierungsansatz praktisch anwendbar und zuverlässig ist.

*Sehr gute Erklärungen!*

## 13 Frage

Initialisieren Sie einen Vektor `fehler`, der später die Unterschiede des Probanden zu den Mittelwerten der gelernten Ziffern enthält. Innerhalb der üblichen Doppelschleife über alle Beispielziffern starten Sie jetzt eine dritte Schleife über alle Mittelwerte mit der Laufvariable `mittelwert`. In dieser innersten Schleife füllen Sie den Fehlervektor nach folgendem Schema:

```
fehler(mittelwert) = ...
sum (sum (abs ( ...
mittelwerte{mittelwert} - ...
double (ziffern{beispiel, ziffer})))));
```

Erläutern Sie ausführlich, was der Befehl leistet.

Warum brauchen Sie zwei `sum`-Befehle?

Wozu ist der `abs`-Befehl notwendig?

### 13.1 Antwort

In dem vorgeschlagenen Code-Snippet wird ein Klassifikationsverfahren durchgeführt, bei dem jede Beispielziffer mit den gemittelten Bildern aller Ziffern verglichen wird. Dies geschieht durch die Berechnung des absoluten Fehlers zwischen den Pixelwerten der Beispielziffer und jedem Mittelwertbild. Der Zweck ist es, herauszufinden, welche gemittelte Ziffer dem aktuellen Beispiel am ähnlichsten ist, basierend auf dem kleinsten Fehler.

#### 13.1.1 Schritt-für-Schritt-Analyse des Befehls

##### 1. Initialisierung des Fehlervektors:

```
fehler = zeros(1, n_ziffern); % Annahme, dass n_ziffern die Anzahl der Ziffernkategorie
```

##### 2. Befehl innerhalb der Schleife:

```
fehler(mittelwert) = sum(sum(abs(mittelwerte{mittelwert} - double(ziffern{beispiel, ziff
```

- `mittelwerte{mittelwert} - double(ziffern{beispiel, ziffer})`: Hier wird die Differenz zwischen dem gemittelten Bild einer Ziffer und der aktuellen Beispielziffer berechnet. Da die Ziffern aus dem Cell Array möglicherweise als Ganzzahl-Typ gespeichert sind, wird die Funktion `double` verwendet, um sicherzustellen, dass die Operationen korrekt im Gleitkommazahl-Format ausgeführt werden. Dies verhindert Datentypkonflikte und Ungenauigkeiten bei der Subtraktion.
- `abs(...)`: Der absolute Wert dieser Differenz wird genommen. Dies ist notwendig, weil die Differenzen negativ oder positiv sein können, und für den Fehlervergleich interessiert nur die Größe des Unterschieds, nicht die Richtung.
- `sum(sum(...))`: Hier werden zwei `sum`-Operationen verwendet. Die erste `sum`-Operation summiert die Elemente jeder Zeile der resultierenden Matrix (aus dem `abs`-Befehl), was einen Vektor ergibt. Die zweite `sum`-Operation summiert dann alle Elemente dieses Vektors, um einen einzigen Skalar zu erhalten, der den gesamten Fehler für diese Ziffer darstellt.

darstellt. Diese doppelte Summation ist notwendig, weil der Fehler über alle Pixel des Bildes aggregiert werden muss, und die Bilder sind 2D-Matrizen.

### 13.1.2 Warum sind diese Schritte notwendig?

- **Doppelte Summation:** Da die Bilder als Matrizen vorliegen, müssen die Fehler über alle Pixel (also über beide Dimensionen der Matrix) aggregiert werden. Eine einfache `sum` würde nur die Summe entlang einer Dimension liefern, wodurch immer noch eine Vektorsumme nötig wäre.
- **Absoluter Wert:** Der absolute Wert ist erforderlich, um eine faire und aussagekräftige Fehlermetrik zu gewährleisten. Ohne den absoluten Wert könnten sich positive und negative Differenzen aufheben, was zu einem irreführenden Fehlermaß führen würde, das tatsächliche Unterschiede zwischen den Bildern nicht korrekt wiedergibt.

### 13.1.3 Zusammenfassung

Dieser Ansatz des Fehlervergleichs ist typisch für einfache Bildklassifikationsaufgaben, wo die Ähnlichkeit zwischen Bildern basierend auf Pixelwertdifferenzen bewertet wird. Es handelt sich um eine grundlegende Methode, die in komplexeren Systemen oft durch ausgefeiltere Techniken wie Merkmalsextraktion und maschinelles Lernen ergänzt oder ersetzt wird.

*Wunderbar!*

## 14 Frage

Außerhalb der innersten, aber noch innerhalb der Doppelschleife, bestimmen Sie jetzt den Index des minimalen Fehlers und weisen ihn mit dem zweiten Rückgabeparameter des `min`-Befehls der entsprechenden Zeile und Spalte der Matrix `index_min` zu, die dann am Ende für jeden Probanden die erkannte Klasse enthält:

```
[~, index_min(beispiel,ziffer)] = min (fehler);
```

Kontrollieren Sie sorgfältig, ob jeder Proband korrekt erkannt wurde.

Stellen Sie mit dem Befehl `imagesc` eine Heatmap des unbereinigten Fehlers irgendeines Probanden und seiner Mittelwertmatrix dar.

Spendieren Sie dem Bild mal ein `colorbar`, um die genaue Zuordnung der Fehlerwerte zu den Farben zu erkennen.

Warum empfiehlt sich hier die Verwendung von `imagesc`?

### 14.1 Antwort

#### 14.1.1 Anweisungen zur Identifikation der minimalen Fehler und Darstellung mit `imagesc`

Nachdem die Fehler für jede Ziffer berechnet wurden, ist der nächste Schritt, den minimalen Fehler zu identifizieren. Dies wird getan, um festzustellen, welche Mittelwert-Ziffer der Beispielziffer am ähnlichsten ist, was einem Klassifizierungsprozess entspricht.

### 14.1.2 Bestimmen des minimalen Fehlers

```
[~, index_min(beispiel, ziffer)] = min(fehler);
```

- **min(fehler)**: Diese Funktion gibt den kleinsten Wert im Vektor `fehler` zurück. Der Vektor `fehler` enthält die aggregierten Fehlerwerte zwischen der Beispielziffer und jedem Mittelwertbild.
- **[~, index\_min(beispiel, ziffer)]**: Das Tilde-Symbol (`~`) wird verwendet, um den minimalen Fehlerwert zu ignorieren, da nur der Index dieses minimalen Werts von Interesse ist. Dieser Index zeigt an, welche Ziffer (von den gemittelten Ziffern) der Beispielziffer am ähnlichsten ist.
- **index\_min(beispiel, ziffer)**: Dies speichert den Index des minimalen Fehlers in einer Matrix, wobei jeder Eintrag den Index der ähnlichsten Ziffer für jedes Beispiel und jede Ziffer darstellt.

### 14.1.3 Darstellung der Fehlermatrix mit `imagesc`

```
figure;  
imagesc(fehler); % Zeigt die Fehlermatrix für eine bestimmte Beispielziffer  
colorbar; % Fügt eine Farbleiste hinzu, um die Fehlerwerte zu visualisieren  
title('Heatmap des Fehlers');
```

#### Warum die Verwendung von `imagesc` empfohlen wird:

1. **Dynamische Skalierung**: `imagesc` skaliert die Farbkodierung der Daten dynamisch zwischen dem kleinsten und größten Wert der Matrix. Dies ist besonders hilfreich, wenn die Fehlerwerte stark variieren, da es hilft, die Unterschiede zwischen den höchsten und niedrigsten Werten visuell zu betonen.
2. **Farbkodierung**: Mit `imagesc` können Sie sofort visuelle Unterschiede in den Daten erkennen. Dies ist nützlich, um zu sehen, welche Ziffern gut und welche schlecht erkannt wurden. Die Farbleiste (`colorbar`) bietet eine direkte Korrelation zwischen den Farben und den tatsächlichen Fehlerwerten, was eine genaue Interpretation der Heatmap ermöglicht.
3. **Bessere Visualisierung der Datenverteilung**: `imagesc` bietet eine klare Visualisierung der Verteilung der Fehler über alle Ziffern hinweg, was bei der Diagnose von Klassifizierungsfehlern oder der Überprüfung der Klassifikationsgenauigkeit hilft.

### 14.1.4 Abschluss

Durch den Einsatz von `imagesc` zur Darstellung des Fehlervektors können Sie schnell und effektiv erkennen, wie gut jede Ziffer klassifiziert wurde. Dies ist ein wertvolles Tool, um die Leistung Ihres Klassifizierungsalgorithmus visuell zu überprüfen und zu verstehen, welche Ziffern möglicherweise schwerer zu erkennen sind, und bietet die Grundlage für weitere Verbesserungen des Klassifizierungsprozesses.

*Sehr schön beschrieben!*