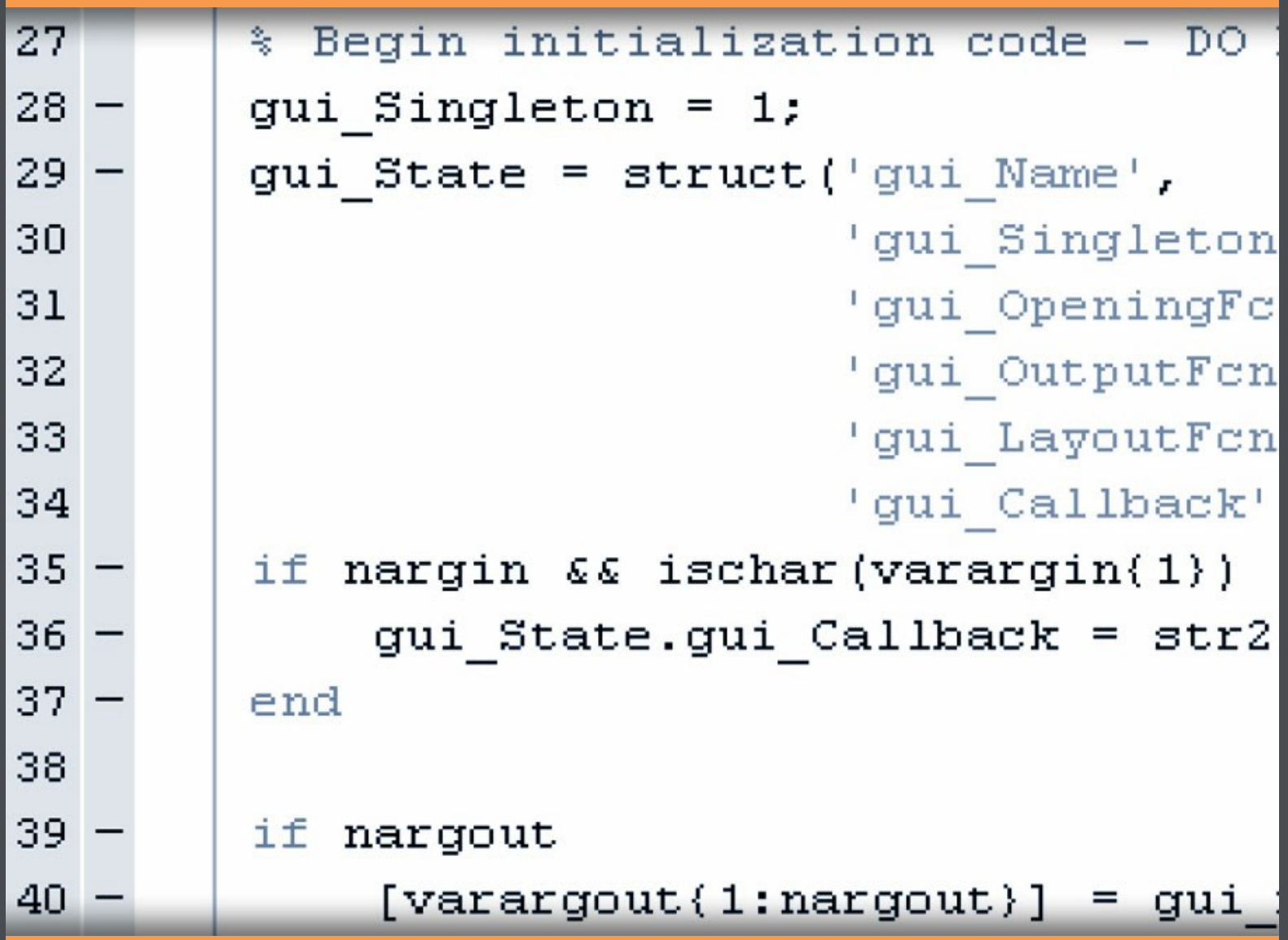
Matlab & Simulink: Aufgaben & Musterlösungen

Simon Ströbel



Simon Ströbel

Einführungskurs Matlab & Simulink

Aufgaben & Musterlösungen

Einführungskurs Matlab & Simulink: Aufgaben & Musterlösungen

ISBN 978-87-7681-870-8

Inhalt

1 Vorwort

Dieses Buch beinhaltet 28 begleitende Übungsaufgaben und Musterlösungen zum Buch „Einführungskurs Matlab & Simulink“. Die Aufgaben sind analog zu diesem Begleitband thematisch in sieben Kapitel gegliedert:

1. Grundlagen
2. Matlab als Programmiersprache
3. Ausgewählte mathematische Anwendungen
4. Differentialgleichungen
5. Simulink
6. Graphical User Interfaces

Die Aufgaben sind von unterschiedlichem Schwierigkeitsgrad, wobei jedoch alle Aufgaben mit den im Begleitband vorgestellten Befehlen und Techniken lösbar sein sollten. Hinweise zu weiterführenden Informationen und Hintergründen sind an entsprechender Stelle gegeben, sofern das notwendige physikalisch/mathematisch/technische Wissen zum Lösen/Verständnis der Aufgaben von Ingenieursstudenten an Fachhochschulen im Grundstudium nicht vorausgesetzt werden kann.

Alle Musterlösungen wurden mit dem Matlab Release R2011b getestet. Kopiert und als M-Files abgespeichert sollten diese durchgehend lauffähig sein. Die Musterlösungen sollen dabei lediglich als „Vorschläge“ verstanden werden. Diese wurden hauptsächlich unter dem Gesichtspunkt erstellt, dass Sie einfach zu lesen, zu verstehen und nachzuvollziehen sind. Effizienz und Ausführungsgeschwindigkeit der Programme war hingegen keine primäre Zielsetzung. Der Leser soll jedoch gerne eigene (unter verschiedenen Gesichtspunkten) „optimale“ Lösungen erarbeiten, die deutlich von den hier zu findenden Musterlösungen abweichen können.

Ich wünsche dem Leser viel Spaß und Erfolg beim Erarbeiten der Aufgaben! Fragen, Anregungen, Lob und Kritik können gerne jederzeit an mich gerichtet werden: simon.stroebel@hs-furtwangen.de

Simon Ströbel
Schömberg, im Oktober 2011

2 Grundlagen

2.1 Vektoren und Matrizen 1

2.1.1 Aufgabe

- Erzeugen Sie die Matrix `E = rand(5,5)`
- Speichern Sie mittels des `size` Befehls die Dimension von E in der Variablen x.
- Weisen Sie mit Hilfe von `x` der Variablen `m` die Anzahl der Zeilen von E zu und der Variablen n die Anzahl der Spalten von `E`.
- Speichern Sie in der Variablen `F` die rechte obere 3x3 Untermatrix von `E`.

2.1.2 Lösung

```
% ------------------------------------------------------------------------
% Übungsaufgaben Matlab und Simulink
% Kapitel 2
% Aufgabe 1: Vektoren und Matrizen 1
% ------------------------------------------------------------------------

E = rand(5,5);      % gleichverteilte Zufallsmatrix E, Dimension 5x5
x = size(E);        % Dimension von E in x speichern
m = x(1);           % Anzahl Zeilen von E
n = x(2);           % Anzahl Spalten von E
F = E(1:3,3:5);     % Untermatrix von E
```

2.2 Vektoren und Matrizen 2

2.2.1 Aufgabe

- Erzeugen Sie zunächst einen äquidistant geteilten Vektor `v` mit -2π als kleinstem, $+2\pi$ als größtem Wert und der Schrittweite $\pi/16$.
- Ermitteln Sie die Länge des Vektors und speichern Sie diesen Wert in der Variablen `laenge`.
- Erzeugen (initialisieren) Sie eine nur mit Nullen besetzte Matrix M der Dimension `laenge`x3.
- Die Matrix `M` soll mit Werten gefüllt werden. Die erste Zeile von M soll die Werte von `v` enthalten, die zweite Zeile die Werte von `sin(v)` und die dritte Spalte die Werte von `cos(v)`.

2.2.2 Lösung

```
% ------------------------------------------------------------------------
% Übungsaufgaben Matlab und Simulink
% Kapitel 2
% Aufgabe 2: Vektoren und Matrizen 2
% ------------------------------------------------------------------------

v = -2*pi:pi/16:2*pi;
laenge = length(v);
M = zeros(laenge,3);
M(:,1) = v;
M(:,2) = sin(v);
M(:,3) = cos(v);
```

2.3 Matrixmanipulation 1

2.3.1 Aufgabe

- Versuchen Sie möglichst effektiv, das heißt mit wenigen Codezeilen, die Matrizen X, Y, und Z (siehe unten) zu generieren.
- Sie sollen dabei bereits die jeweils vorhandene Matrix für die Generierung der nachfolgenden verwenden.
- Versuchen Sie nicht jedes Element von X einzeln einzutippen, sondern entsprechend sinnvolle Funktionen und Methoden der Matrixmanipulation zur Hilfe zu nehmen. Versuchen Sie deshalb zunächst eine entsprechende Struktur im Aufbau der Matrizen zu erkennen.

$$X = \begin{bmatrix} 1 & 1 & 1 & 1 & 2 & 2 \\ 1 & 1 & 1 & 1 & 2 & 2 \\ 3 & 3 & 4 & 4 & 4 & 4 \\ 3 & 3 & 4 & 4 & 4 & 4 \\ 3 & 3 & 4 & 4 & 4 & 4 \end{bmatrix} \quad Y = \begin{bmatrix} 1 & 1 & 1 & 1 & 2 & 2 \\ 1 & 1 & 5 & 5 & 2 & 2 \\ 3 & 3 & 5 & 5 & 4 & 4 \\ 3 & 3 & 5 & 5 & 4 & 4 \\ 3 & 3 & 4 & 4 & 4 & 4 \end{bmatrix} \quad Z = \begin{bmatrix} 2 & 2 & 1 & 1 & 1 & 1 \\ 2 & 2 & 5 & 5 & 1 & 1 \\ 4 & 4 & 5 & 5 & 3 & 3 \\ 4 & 4 & 5 & 5 & 3 & 3 \\ 4 & 4 & 4 & 4 & 3 & 3 \end{bmatrix}$$

2.3.2 Lösung

```
% ------------------------------------------------------------------------
% Übungsaufgaben Matlab und Simulink
% Kapitel 2
% Aufgabe 3: Matrixmanipulation 1
% ------------------------------------------------------------------------

H1 = 1*ones(2,4);          % Hilfsmatrizen generieren
H2 = 2*ones(2,2);
H3 = 3*ones(3,2);
H4 = 4*ones(3,4);

X = [H1,H2;H3,H4];  % Matrix X aus Hilfsmatrizen zusammensetzen

% Alternative 1: X = vertcat(horzcat(A1,A2),horzcat(A3,A4));
% Alternative 2: X = [1*ones(2,4),2*ones(2,2);3*ones(3,2),4*ones(3,4)];

Y = X;              % Matrix Y initialisieren
Y(2:4,3:4) = 5;     % Für Y den „mittleren" Elementen neue Werte zuweisen

Z = fliplr(Y);      % Z ist Y horizontal gespiegelt
```

2.4 Matrixmanipulation 2

2.4.1 Aufgabe

- Erzeugen sie die quadratische 100x100 Matrix `A` mit der `magic()` Funktion. `A = magic(100);`
- Erstellen Sie (geschickt) den Zeilenvektor `b` mit der Dimension 1x200, in welchem die Elemente der ersten und der letzten Zeile von A stehen.
- Erzeugen sie (geschickt) einen Zeilenvektor `c`, indem lediglich die Elemente von `b` stehen, die einen *Zahlenwert* im abgeschlossenen Intervall [50, 100] haben. Hinweis: verwenden Sie hierzu beispielsweise logische Operatoren, Vergleichsoperatoren und/oder die `find()` Funktion.
- Bestimmen Sie die Anzahl der Elemente in `c`.

2.4.2 Lösung

```
% ------------------------------------------------------------------------
% Übungsaufgaben Matlab und Simulink
% Kapitel 2
% Aufgabe 4: Matrixmanipulation 2
% ------------------------------------------------------------------------

A = magic(100);             % Matrix A generieren
b = [A(1,:),A(100,:)];      % Erste und letzte Zeile im Vektor b speichern
ind = b>=50 & b<=100;       % Die Indizes der Elemente im Intervall bestimmen
c = b(ind);                 % Die Elemente im Intervall im Vektor c abspeichern
anzahl = length(c);         % Die Anzahl der Elemente im Vektor c bestimmen
```

2.5 Polarplot

2.5.1 Aufgabe

- Der Befehl `polar()` arbeitet ähnlich wie der Befehl `plot()`, nur dass die Zeichnung nicht in kartesischen Koordinaten sondern in Polarkoordinaten erstellt und skaliert wird. Aufruf: `h = polar(phi,r)`
- Hierbei ist `phi` die Winkelangabe und r der Radius in Polarkoordinaten. `h` ist die zu dem Plot gehörende Handles Structure.
- Zeichnen Sie die durch die folgenden Vektoren definierten Punkte in Polarkoordinaten:
  `phi = [-1:0.01:1]*2*pi;`
  `r = sin(phi).*cos(phi);`
- Lassen Sie sich anschließend die Eigenschaften (*properties*) des Plots mit der `get(h)` Funktion anzeigen.
- Versuchen Sie die Eigenschaften des Plots so zu ändern, dass
  - die Strichstärke `('LineWidth')` des Plots 2 beträgt.
  - als Markierungen `(,Marker')` Kreise `('o')` benutzt werden.
  - Die Farbe `('Color')` des Plots im RGB-Farbraum den Wert `[.9 .3 .7]` hat.
- Tauschen Sie zum Vergleich den Befehl `polar()` gegen den Befehl `plot()` aus.

2.5.2 Lösung

```
% ------------------------------------------------------------------------
% Übungsaufgaben Matlab und Simulink
% Kapitel 2
% Aufgabe 5: Polarplot
% ------------------------------------------------------------------------

% --- Vektoren erstellen ---
      phi = [-1:0.01:1]*2*pi;
      r = sin(phi).*cos(phi);

% --- Plot in Polarkoordinaten inkl. Handels Structure ---
      h = polar(phi,r);

% --- properties im Command Window anzeigen ---
      get(h)
% --- properties entsprechend ändern ---
      set(h,'LineWidth',2)
      set(h,'Marker','o')
      set(h,'Color',[.9 .3 .7])

% --- oder auch durch einen Aufruf von set(): ---
      set(h,'LineWidth',2,'Marker','o','Color',[.9 .3 .7])
```

3 Matlab als Programmiersprache

3.1 Reihenentwicklung der Eulerschen Zahl (Skript)

3.1.1 Hintergrund

Die eulersche Zahl *e* ist über folgende *unendliche Reihe* definiert:

$$e = 1 + \sum_{k=1}^{\infty} \frac{1}{k!} = 1 + \frac{1}{1} + \frac{1}{1 \cdot 2} + \frac{1}{1 \cdot 2 \cdot 3} + \frac{1}{1 \cdot 2 \cdot 3 \cdot 4} + \ldots$$

Diese kann somit folglich über die *endliche Summe*

$$e = 1 + \sum_{k=1}^{n} \frac{1}{k!} = 1 + \frac{1}{1} + \frac{1}{1 \cdot 2} + \frac{1}{1 \cdot 2 \cdot 3} + \frac{1}{1 \cdot 2 \cdot 3 \cdot 4} + \ldots + \frac{1}{1 \cdot 2 \cdot 3 \cdot 4 \cdot \ldots \cdot n}$$

angenähert werden.

3.1.2 Aufgabe

Erstellen Sie ein Matlab *Skript*, das folgendes leistet:

- Zu Beginn wird eine beliebige ganze Zahl in der Variablen `n` gespeichert.
- In einem Vektor `fact` der Dimension 1*xn* speichern Sie alle Fakultäten von 1 bis *n*. Führen Sie die Berechnung der Fakultäten „von Hand“ aus. (Anmerkung: Es gibt auch eine von Matlab zur Verfügung gestellte Funktion `factorial()` zur Berechnung der Fakultät.)
- Ist die Zahl *n* kleiner 1, so soll in der Variablen `error` der Wert 1 gespeichert werden, in der Variablen `success` der Wert 0 und in der Variablen `e_approx` der Wert -1.
- Ist die Zahl *n* größer oder gleich 1, so soll in der Variablen `error` der Wert 0 gespeichert werden und folgendes ausgeführt werden:
    - Der wahre Wert der Eulerschen Zahl *e* wird durch oben angegebene Summenformel in einer Schleife iterativ angenähert. Der Wert der Annäherung an *e* wird in der Variablen `e_approx` gespeichert.
    - Die Variable `deviation` enthält den Betrag der Abweichung von einer Iteration zur vorherigen.
- Die Berechnung der Eulerschen Zahl über die endliche Summe wird abgebrochen, wenn:
    - `deviation` kleiner oder gleich $1 \cdot 10^{-6}$ ist. Dann wird in der Variablen `success` der Wert 1 gespeichert.
    - Die Schleife *n* mal durchlaufen wurde, aber `deviation` größer als 1·10-6 ist. Dann wird in der Variablen `success` der Wert 0 gespeichert.

3.1.3 Lösung

```
------------------------------------------------------------------------
% Übungsaufgaben Matlab und Simulink
% Kapitel 3
% Aufgabe 1: Reihenentwicklung der Eulerschen Zahl (Skript)
% ------------------------------------------------------------------------

% --- ganze Zahl n definieren ---
n = 100;

% --- Vektor mit Fakultäten bestimmen ---
fact = ones(1,n);
for(k=2:1:n)
       fact(k) = fact(k-1)*k;
end

% --- Annäherung an die Zahl e ---
if(n>=1) % n ist gültig

       error = 0;
       success = 0;
       k = 0;
       tolerance = 1e-6;          % Näherungstoleranz / Abbruchkriterium
       deviation = inf;           % Abweichung
       e_approx = 1;

       while(k<n) % Schleife zur Berechnung von e
              k = k+1;
              e_approx_pre   = e_approx;
              e_approx       = e_approx + 1/fact(k);
              deviation      = abs(e_approx-e_approx_pre);
              if(deviation<=tolerance)          % Prüfung ob die Abweichung kleiner
                     success = 1;               % als die Toleranz ist
                     break
              end
       end
else % n ist ungültig
       error = 1;
       success = 0;
       e_approx = -1;
end
```

3.2 Reihenentwicklung der Eulerschen Zahl (Funktion)

3.2.1 Aufgabe

- Erweitern Sie ihr Skript aus Aufgabe 3.1 zu einer *Funktion* `approxEuler()` mit folgender Struktur, bzw. mit folgendem Funktionskopf:

  ```
  function [e_approx,success,error] = approxEuler(n,tolerance)
  ```

- Die funktionsargumente sind die ganze Zahl `n`, welche die Anzahl der maximalen Schleifendurchläufe angibt und die Variable `tolerance`, in der man das Abbruchkriterium (z.B. $1 \cdot 10^{-6}$) für den Betrag der Abweichung an die eulersche Zahl *e* angeben kann.
- Rückgabevariablen sind `e_approx`, `success` und `error` wie in Aufgabe 3.1 beschrieben.

3.2.2 Lösung

```
function [e_approx,success,error] = approxEuler(n,tolerance)
% ------------------------------------------------------------------------
% Übungsaufgaben Matlab und Simulink
% Kapitel 3
% Aufgabe 2: Reihenentwicklung der Eulerschen Zahl (Funktion)
% ------------------------------------------------------------------------

% --- Vektor mit Fakultäten bestimmen ---
fact = ones(1,n);
for(k=2:1:n)
	fact(k) = fact(k-1)*k;
end

% --- Annäherung an die Eulersche Zahl ---
if(n>=1)
	% --- Variablen sinnvolle Startwerte zuweisen ---
	error = 0;
	success = 0;
	k = 0;
	deviation = inf;
	e_approx = 1;

	while(k<n)
		k = k+1;
		e_approx_pre = e_approx;
		e_approx = e_approx + 1/fact(k);
		deviation = abs(e_approx-e_approx_pre);
		if(deviation<=tolerance)
			success = 1;
			break
		end
	end
else
	error = 1;
	success = 0;
	e_approx = -1;
end
```

3.3 Fibonacci-Folge und Goldener Schnitt

3.3.1 Hintergrund

Die Fibonacci-Folge (http://de.wikipedia.org/wiki/Fibonacci-Folge), benannt nach Leonardo Fibonacci, ist durch die Rekursion *Fn = Fn-1 + Fn-2* mit *F0 = F1 = 1* für n∈N gegeben. Also: *F* = 1, 1, 2, 3, 5, 8, 13, 21,…

Beim *Goldenen Schnitt* (http://de.wikipedia.org/wiki/Goldener_Schnitt) entsteht ein bestimmtes Verhältnis zwischen zwei Zahlen oder zwei Größen. Dieses Verhältnis ist die Goldene Zahl Φ und hat den Wert:

$$\Phi = \frac{1+\sqrt{5}}{2} \approx 1.618$$

3.3.2 Aufgabe

- Schreiben Sie eine Funktion mit der Sie die ersten *n* Fibonacci Zahlen berechnen können. Die *n* Fibonacci Zahlen sollen von der Funktion als Vektor `F` der Dimension 1x*n* zurückgegeben werden: `function [F] = fibonacci(n)`
- Schreiben sie ein Skript, in welchem die ersten 25 Fibonacci-Zahlen mit Hilfe der Funktion `fibonacci(n)` berechnet werden. Es soll grafisch dargestellt werden, dass der Quotient zweier aufeinanderfolgender Fibonacci-Zahlen *Fn / Fn-1* gegen den Grenzwert Φ konvergiert.

3.3.3 Lösung

3.3.3.1 Funktion

```
function [F] = fibonacci(n)
% ------------------------------------------------------------------------
% Übungsaufgaben Matlab und Simulink
% Kapitel 3
% Aufgabe 3: Fibonacci-Folge und Goldener Schnitt
% ------------------------------------------------------------------------
F = ones(1,n);      % Initialisieren eines Vektors der passenden Größe

for (k=3:1:n) % Fibonacci Zahlen berechnen, beginnend bei der dritten Zahl
       F(k) = F(k-1)+F(k-2);
end
```

3.3.3.2 Skript

```
% ------------------------------------------------------------------------
% Übungsaufgaben Matlab und Simulink
% Kapitel 3
% Aufgabe 3: Fibonacci-Folge und Goldener Schnitt
% ------------------------------------------------------------------------

n = 25;
F = fibonacci(n); % Berechnung der ersten 25 Fibonacci Zahlen

for(k=1:1:n-1)
       Fq(k) = F(k+1)/F(k); % Berechnung des Quotienten Fq
end
figure(1) % Darstellung des Quotienten und des Grenzwerts
       hold on
       plot([0 n],[1 1]*(1+sqrt(5))/2,'r','LineWidth',2)
       plot(Fq,'bx-','LineWidth',1)
       grid on
       axis([0 n 0 3])
       set(gca,'XTick',[0:1:n])
       set(gca,'YTick',[0:(1+sqrt(5))/2:3])
       title('Konvergenz der Fibonacci Folge','FontSize',14)
       xlabel('n','FontSize',14)
       legend('Verhältnis des goldenen Schnitts','Quotient der Fibonacci Folge')
```

3.4 Kreis zeichnen

3.4.1 Aufgabe

- Erstellen Sie eine Funktion mit folgendem Funktionskopf:
  `function [] = kreisZeichnen(Radius,Mittelpunkt)`
- Die Funktion soll folgendes leisten:

1. Es wird ein Figure erstellt.
2. Es wird ein roter Kreis mit dem als Funktionsargument übergebenen Radius und Mittelpunkt gezeichnet. `Radius` ist ein Skalar und `Mittelpunkt` ein Vektor mit zwei Elementen, den x- und der y-Koordinate des Kreismittelpunktes (Die Einheit der Angaben sei *cm*). Der Kreis soll über den `plot()`-Befehl gezeichnet werden und eine durchgehende Linie verbundene aus 1000 Einzelpunkten sein. Die Strichstärke soll 2 betragen.
3. Im Kreismittelpunkt wird ein einziger Kleiner roter Punkt gezeichnet.
4. Die x-Achse und die y-Achse soll in schwarz als durchgehende Linie dargestellt werden.
5. Die x-Achse und die y-Achse soll von -10 cm bis +10 cm dargestellt werden.

6. Die Skalierung der Achsen soll im Intervall von 2 cm erfolgen.
7. Es wird der Flächeninhalt `A` des Kreises im Command Window mit der Einheit cm² angezeigt.

Wird ihre Funktion also folgendermaßen aufgerufen, sollten Sie die untenstehende (oder ähnliche) Ausgabe erhalten:

```
>> kreisZeichnen(4,[-4 2])

A = 50.27 cm²
```

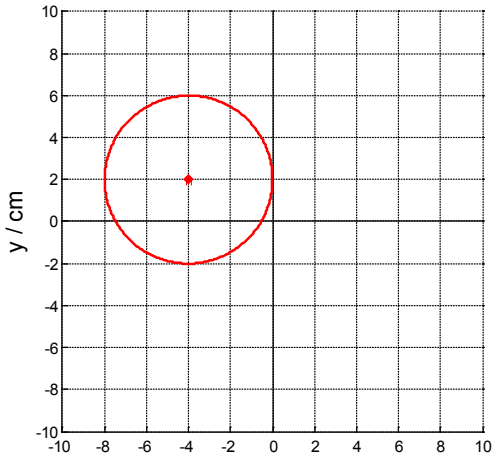


Abbildung 1: Kreis zeichnen

3.4.1.1 Hinweise

- mit dem Befehl `axis square` erhalten Sie eine quadratische Darstellung der Achsen, so dass ein Kreis also auch wie ein Kreis aussieht und keine elliptische Verzerrung stattfindet.

3.4.2 Lösung

```
function [] = kreisZeichnen(Radius,Mittelpunkt)
% ------------------------------------------------------------------------
% Übungsaufgaben Matlab und Simulink
% Kapitel 3
% Aufgabe 4: Kreis zeichnen
% ------------------------------------------------------------------------
\% --- Kreisberechnung ---
       phi = linspace(0,2*pi,1000);
       x = Radius*cos(phi);
       y = Radius*sin(phi);
       A = pi*Radius^2;

% --- Ausgabe des Flächeninhalts am Bildschirm ---
       fprintf('A = % 5.2f cm²\n',A)

% --- Zeichnen des Kreises ---
       figure(1)
       hold on
       plot([-10 10],[0 0],'k') % X-Achse zeichnen
       plot([0 0],[-10 10],'k') % Y-Achse zeichnen
       plot(x+Mittelpunkt(1),y+Mittelpunkt(2),'r','LineWidth',2) % Kreis zeichnen
       plot(Mittelpunkt(1),Mittelpunkt(2),'rx','LineWidth',5) % Kreismittelp. zeichnen
       axis([-10 10 -10 10]) % Min und Max Bereich der Achsen
       axis square % Qudratische Darstellung
       set(gca,'XTick',[-10:2:10]) % Skalierung der X-Achse
       set(gca,'YTick',[-10:2:10]) % Skalierung der Y-Achse
       grid on % Gitternetzlinie ein
       xlabel('x / cm','FontSize',14)
       ylabel('y / cm','FontSize',14)
```

3.5 Lotto

3.5.1 Aufgabe

Erstellen Sie die folgenden fünf Funktionen mit den angegebenen Funktionsköpfen:

```
1  function [lottoZahlen] = lottoZiehung()
```

Die Rückgabevariable `lottoZahlen` der Funktion soll ein Zeilenvektor der Dimension 1x6 sein, der gleichverteilte Zufallszahlen zwischen 1 und 49 enthält. Jede Zahl darf natürlich nur einmal im Vektor vorkommen.

```
2  function [flag] = lottoZahlenPruefen(lottoZahlen)
```

Das Argument `lottoZahlen` ist ein Zeilenvektor der Dimension 1x6. Die Funktion soll überprüfen,

a) ob der Vektor die richtige Dimension hat,
b) ob die Zahlen im Intervall [1, 49] liegen,
c) ob die Zahlen ganzzahlig sind und ob jede Zahl nur einmal im Vektor vorkommt.

Sind alle Bedingungen erfüllt, dann wird als Rückgabewert 1 geliefert, ansonsten 0.

```
3  function [anzahlRichtige] = lottoSpiel(lottoZahlen,lottoTipp)
```

Das Argument `lottoZahlen` ist der Rückgabewert der Funktion `lottoZiehung()` und `lottoTipp` ihr „Tippschein“, also ebenfalls ein entsprechender Vektor der Dimension 1x6. Von beiden Vektoren kann angenommen werden, dass diese gültige Werte enthalten (also beispielsweise durch die Funktion `lottoZahlenPruefen()` überprüft wurden). Als Rückgabewert wird die Anzahl der Übereinstimmungen in beiden Vektoren zurückgegeben.

```
4  function [flag,richtige] = lottoSerie(lottoTipp,n)
```

Die Funktion führt unter Verwendung der unter 1-3 erstellten Funktionen n Lottoziehungen durch. Der Rückgabewert richtige ist ein 4x1 Spaltenvektor. In ihm wird festgehalten wie oft man mit demselben Lottotipp erfolgreich war: Im ersten Element steht die Anzahl der „drei Richtigen", im zweiten die Anzahl der „vier Richtigen", im dritten die Anzahl der „fünf Richtigen" und im vierten die Anzahl „sechs Richtigen". In `flag` steht eine 1, wenn bei jedem Lottospiel alles mit rechten Dingen zugegangen ist: Wenn der Tippschein nicht korrekt ausgefüllt wurde oder die Lottoziehung einmal ungültige Zahlen liefern sollte, wird abgebrochen und in `flag` steht eine 0. Der Rückgabewert `richtige` ist dann eine leere 0x0 Matrix.

```
5   function [] = pruefeGleichverteilung(n)
```

Die Funktion soll zunächst *n* mal die Funktion `lottoZiehung()` aufrufen und sich die Häufigkeit (Summe) der jeweils gezogenen Zahlen merken. Das Ergebnis soll grafisch dargestellt werden. Nutzen Sie hierzu die die Balkendiagramm-Funktion `bar()`. Beschriften und Formatieren Sie die grafische Ausgabe sinnvoll.

3.5.2 Hinweise

- Speichern Sie jede Funktion in einer eigenen Datei, die den gleichen Namen wie die jeweilige Funktion trägt.
- Mit der Sortier-Funktion `sort()` können Sie einen Vektor aufsteigend sortieren.
- Zur Überprüfung, ob eine Zahl ganzzahlig ist, können Sie beispielsweise die Modulo-Funktion `mod()` verwenden. Diese gibt den Rest einer Division zurück.
- Informieren Sie sich gegebenenfalls mittels der Dokumentation über die Funktion `sort()`, `mod()` und `bar()`.

3.5.3 Analyse

- Rufen Sie die Funktion `pruefeGleichverteilung()` mit dem Argument `1e6` auf (→ 1 Million Lottoziehungen) und bewerten Sie anhand des Diagramms, ob alle Zahlen (mehr oder weniger) gleich oft gezogen werden, die Zahlen also wirklich gleichverteilt sind. Prüfen Sie insbesondere ob die Zahlen 1 und 49 gleich oft gezogen werden wie die anderen, oder ob signifikante Abweichungen erkennbar sind. Falls ja versuchen Sie ihre Funktion `lottoZiehung()` zu optimieren.
- Rufen Sie die Funktion `lottoSerie()` mit einem Lottotipp ihrer Wahl und dem Argument *n* = 1e6 auf (→ 1 Million Lottoziehungen) auf. Berechnen Sie anhand der Ergebnisse im Rückgabeargument `richtige` die Wahrscheinlichkeit eines Dreiers, eines Vierers, eines Fünfers und eines Sechsers. Vergleichen Sie diese Gewinnwahrscheinlichkeit mit der theoretischen Gewinnwahrscheinlichkeit (siehe http://de.wikipedia.org/wiki/Lotto). Wiederholen Sie mit 10 Millionen Ziehungen und 100 Millionen Ziehungen. Achtung: Je nachdem wie effizient Sie die einzelnen Funktionen Programmiert haben, kann dies mehrere Minuten/ Stunden in Anspruch nehmen.

3.5.4 Lösung

3.5.4.1 lottoZiehung()

```
function [lottoZahlen] = lottoZiehung()
% ------------------------------------------------------------------------
% Übungsaufgaben Matlab und Simulink
% Kapitel 3
% Aufgabe 5: Lotto
% ------------------------------------------------------------------------

repeat = 1;

while(repeat)

% --- Zufällige Ganzzahlen von 1 bis 49 generieren ---
      lottoZahlen = fix(49*rand(1,6))+1;

% --- Zufallszahlen sortieren ---
      lottoZahlen = sort(lottoZahlen);

% --- Überprüfen, ob jede Zahl nur einmal vorkommt
      for(k=1:1:5)
            if(lottoZahlen(k) == lottoZahlen(k+1))
                  repeat = 1;
                  break
            else
                  repeat = 0;
            end
      end
end
```

3.5.4.2 lottoZahlenPruefen()

```
function [flag] = lottoZahlenPruefen(lottoZahlen)
% ------------------------------------------------------------------------
% Übungsaufgaben Matlab und Simulink
% Kapitel 3
% Aufgabe 5: Lotto
% ------------------------------------------------------------------------
flag = 1;

% --- lottoZahlen aufsteigend sortieren ---
lottoZahlen = sort(lottoZahlen);

% --- 1. Dimension überprüfen ---
[m n] = (size(lottoZahlen));

if(m~=1 && n~=6)
	flag = 0;
end

% --- 2. Intervall überprüfen ---
if(lottoZahlen(1)<1 || lottoZahlen(6)>49)
	flag = 0;

end

% --- 3. Ganzzahligkeit überprüfen ---
if(sum(mod(lottoZahlen,1)))
	flag = 0;
end

% --- 4. prüfen ob nur jede Zahl einmal vorhanden ---
for(k=1:1:5)
	if(lottoZahlen(k) == lottoZahlen(k+1))
		flag = 0;
		break
	end
end
```

3.5.4.3 lottoSpiel()

```
function [anzahlRichtige] = lottoSpiel(lottoZahlen,lottoTipp)
% ------------------------------------------------------------------------
% Übungsaufgaben Matlab und Simulink
% Kapitel 3
% Aufgabe 5: Lotto
% ------------------------------------------------------------------------

% --- Zwei nullbesetzte 1x49 Vektoren erzeugen ---
      ind_lottoZahlen = zeros(1,49);
      ind_lottoTipp = zeros(1,49);

% --- Die Zahlen der Ziehung und des Tipps ---
% --- den entsprechenden Stellen zuordnen ---
      ind_lottoZahlen(lottoZahlen) = 1;
      ind_lottoTipp(lottoTipp) = 1;

% --- Treffer, wo Zahlen der Ziehung und ---
% --- Tippschein übereinstimmen ---
      identisch = and(ind_lottoZahlen,ind_lottoTipp);

% --- Die Anzahl der richtigen ermitteln ---
      anzahlRichtige = sum(identisch);
```

3.5.4.4 ottoSerie()

```
function [flag,richtige] = lottoSerie(lottoTipp,n)
% ------------------------------------------------------------------------
% Übungsaufgaben Matlab und Simulink
% Kapitel 3
% Aufgabe 5:Lotto
% ------------------------------------------------------------------------

if(lottoZahlenPruefen(lottoTipp)) % Prüfen, ob Tippschein korrekt

       flag = 1;
       richtige = zeros(1,4);

for(k=1:1:n) % n-Mal Lotto spielen
       lottoZahlen = lottoZiehung();
              if(lottoZahlenPruefen(lottoZahlen)) % Prüfen, ob Zahlen gültig
                     AnzahlRichtige = lottoSpiel(lottoZahlen,lottoTipp);
                     switch AnzahlRichtige
                            case 3
                                   richtige(1) = richtige(1)+1;
                            case 4
                                   richtige(2) = richtige(2)+1;
                            case 5
                                   richtige(3) = richtige(3)+1;
                            case 6
                                   richtige(4) = richtige(4)+1;
                     end
              else % Lottzahlen sind ungültig
                     flag = 0;
                     richtige = [];
                     break
              end
       end
else % LottoTipp ist ungültig
       flag = 0;
       richtige = [];
end
```

3.5.4.5 pruefeGleichverteilung()

```
function [] = pruefeGleichverteilung(n)
% ------------------------------------------------------------------------
% Übungsaufgaben Matlab und Simulink
% Kapitel 3
% Aufgabe 5: Lotto
% ------------------------------------------------------------------------

Z = [1:1:49];       % Vektor von 1 bis 49
S = zeros(1,49);    % Vektor, in dem die Häufigkeit der gezogenen Zahlen
                    % gespeichert wird
% --- n mal Lotto spielen und die Häufigkeit der Zahlen merken ---
      for(k=1:1:n)
            N = zeros(1,49);
            lottoZahlen = lottoZiehung();
            if(lottoZahlenPruefen(lottoZahlen))
                  N(lottoZahlen) = 1;
                  S = S+N;
            end
      end

% --- Ergebnis grafisch darstellen ---
      bar(Z,S,'r')
      set(1,'position',[50 200 1200 300])
      axis([1 49 0 max(S)*1.5])
      grid on
      set(gca,'Xtick',[1:1:49])
      xlabel('Lotto Zahl')
      ylabel('Häufigkeit')
```

3.6 Multiplikation in verschiedenen Zahlensystemen

3.6.1 Aufgabe

- Erzeugen Sie eine Funktion mit folgendem Funktionskopf:
  `function [result] = multiplication(A,B,typA,typB)`
- Die Funktion soll die beiden Skalare `A` und `B` miteinander multiplizieren und das Ergebnis in der Struktur Variablen `result` speichern. Die Zahlen `A` und `B` sollen dabei als Zeichenketten übergeben werden.
- `result` hat die Felder `dec, bin, oct`, und hex in denen das Ergebnis der Multiplikation jeweils in der Dezimal-, in der Binär- in der Oktal- und in der Hexadezimaldarstellung als Zeichenkette gespeichert wird.
- Zusätzlich hat `result` noch die Felder `A_dec` und `B_dec` in denen der Wert der Variablen `A` und `B` in der Dezimaldarstellung als numerischer Wert (Datentyp double) zurückgegeben wird.
- Über `typA` und `typB` kann man auswählen, ob die Zahlen `A` und `B` in der Dezimal-, in der Binär-, in der Oktal- oder in der Hexadezimaldarstellung vorliegen. `typA` und `typB` kann dabei eine der vier Zeichenketten sein: `'dec', 'bin', 'oct'` oder `'hex'`.

3.6.2 Hinweis

- Gehen Sie davon aus, dass in den vier übergebenen Argumenten plausible und zueinander passende Werte stehen, sie also keine Überprüfung der Argumente machen müssen.

3.6.3 Lösung

```
function [result] = multiplication(A,B,typA,typB)
% ------------------------------------------------------------------------
% Übungsaufgaben Matlab und Simulink
% Kapitel 3
% Aufgabe 6: Multiplikation in verschiedenen Zahlensystemen
% ------------------------------------------------------------------------

% --- A in einen numerischen Dezimalwert wandeln ---
switch typA
	case 'dec'
		A = str2num(A);
	case 'bin'
		A = bin2dec(A);
	case 'oct'

		A = base2dec(A,8);
	case 'hex'
		A = hex2dec(A);
end

% --- B in einen numerischen Dezimalwert wandeln ---
switch typB
	case 'dec'
		B = str2num(B);
	case 'bin'
		B = bin2dec(B);
	case 'oct'
		B = base2dec(B,8);
	case 'hex'
		B = hex2dec(B);
end

% --- Multiplikation durchführen ---
value = A*B;
```

```
% --- Rückgabe Structure ---
result.dec = num2str(value);
result.bin = dec2bin(value);
result.oct = dec2base(value,8);
result.hex = dec2hex(value);
result.A_dec = A;
result.B_dec = B;
```

4 Ausgewählte mathematische Anwendungen

4.1 Lineares Gleichungssystem

4.1.1 Aufgabe

Lösen Sie das lineare Gleichungssystem:

$$\begin{aligned} 3-p &= r \\ 4q &= -3p \\ -2p+3q+r &= 11 \end{aligned}$$

4.1.2 Lösung

Das Gleichungssystem kann auch folgendermaßen geschrieben werden:

$$\begin{aligned} -1p+0q-1r &= -3 \\ 3p+4q+0r &= 0 \\ -2p+3q+1r &= 11 \end{aligned}$$

Beziehungsweise in Matrizenform:

$$\underbrace{\begin{bmatrix} -1 & 0 & -1 \\ 3 & 4 & 0 \\ -2 & 3 & 1 \end{bmatrix}}_{A} \cdot \underbrace{\begin{bmatrix} p \\ q \\ r \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} -3 \\ 0 \\ 11 \end{bmatrix}}_{c}$$

```
% ------------------------------------------------------------------------
% Übungsaufgaben Matlab und Simulink
% Kapitel 4
% Aufgabe 1: Lineares Gleichungsystem
% ------------------------------------------------------------------------

A = [-1,0,1;3,4,0;-2,3,1]; % Gleichungssystem definieren
c = [-3;0;11];

x = A\c;                   % Gleichungssystem lösen

p = x(1);                  % Lösungevektor den entspr. Variablen zuordnen
q = x(2);
r = x(3);

% --- Kontrolle ---
A*x                        % A*x = c
```

4.2 Mallorca

4.2.1 Aufgabe

- Nachfolgendes Diagramm listet die klimatischen Durchschnittswerte der Insel Mallorca auf (Quelle: http://de.wikipedia.org/wiki/Mallorca)
- Erstellen Sie eine Datei *mallorca_klima.mat*, in welcher Sie in einer entsprechenden Strukturvariablen die Klimawerte abspeichern.
- Erzeugen Sie ein Balkendiagramm, in welchem die monatliche Durchschnittstemperatur aufgetragen wird.
- Approximieren Sie diese Daten durch ein Polynom fünfter Ordnung und zeichnen Sie dieses ebenfalls in das Diagramm ein. Das Polynom soll dabei an 365 Stellen (= Tagen) ausgewertet werden.
- Beschriften Sie das Diagramm sinnvoll.

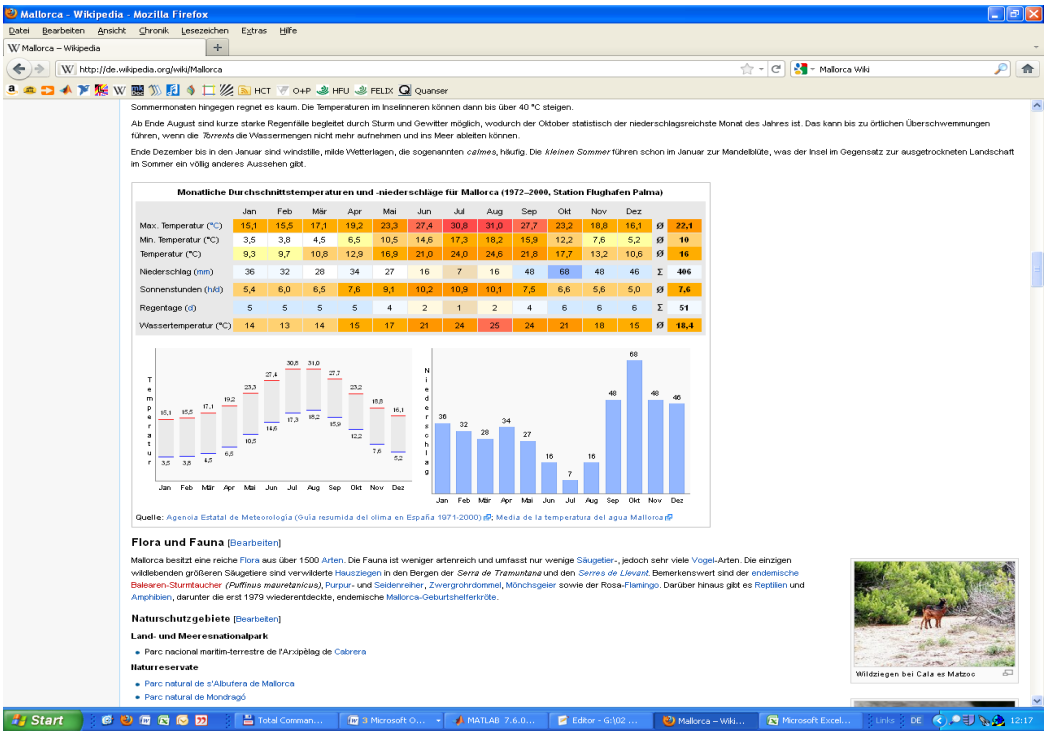


Abbildung 2: Klimadaten Mallorca

4.2.2 Hinweise

- Sie können die horizontale Achse mit Zeichenketten beschriften indem sie auf die `XTickLabel` Eigenschaft der Achse zugreifen.
- Beispiel: `set(gca,'XTickLabel',{'s1','s2',…,'sn'})`
- Sie sollten also ein Diagramm ähnlich dem nachfolgenden erhalten:

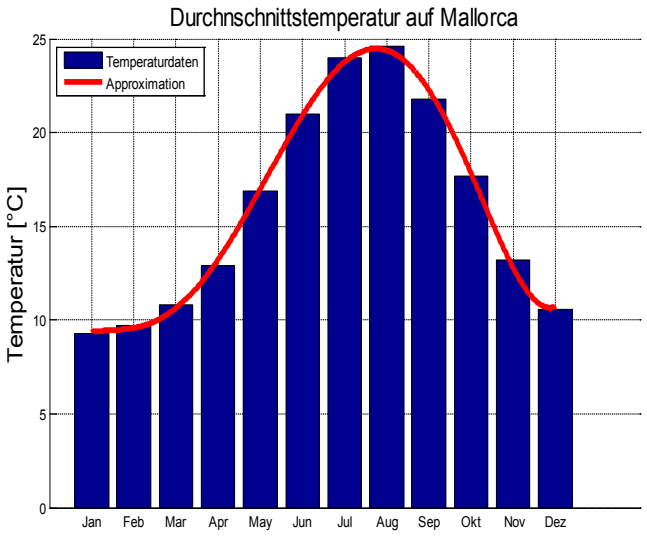


Abbildung 3: Durchschnittstemperatur Mallorca

4.2.3 Analyse

- An welchem Tag des Jahres ist im Schnitt die höchste Temperatur zu erwarten?
- Wie hoch ist diese?

4.2.4 Lösung

4.2.4.1 MAT Datei

```
% ------------------------------------------------------------------------
% Übungsaufgaben Matlab und Simulink
% Kapitel 4
% Aufgabe 2: Mallorca
% ------------------------------------------------------------------------

% --- Daten erzeugen ----------------------------------------------------

Mallorca.Monat.Name = {,Jan','Feb','Mar','Apr','May','Jun',...,Jul','Aug','Sep','Okt','N
                      ov','Dez'};

Mallorca.Monat.Nummer = [1:1:12];

Mallorca.Temperatur.Max = [15.1 15.5 17.1 19.2 23.3 27.4...30.8 31.0 27.7 23.2 18.8 16.1];

Mallorca.Temperatur.Min = [3.5 3.8 4.5 6.5 10.5 14.6 17.3 18.2 15.9 12.2 7.6 5.2];

Mallorca.Temperatur.Mittel = [9.3 9.7 10.8 12.9 16.9 21.0 24.0 24.6 21.8 17.7 13.2 10.6];

Mallorca.Temperatur.Wasser = [14 13 14 15 17 21 24 25 24 21 18 15];

% --- Daten in MAT Datei „mallorca_klima.mat“ speichern -------------------

save mallorca_klima
```

4.2.4.2 Auswertung der Klimadaten

```
% ------------------------------------------------------------------------
% Übungsaufgaben Matlab und Simulink
% Kapitel 4
% Aufgabe 2: Mallorca
% ------------------------------------------------------------------------

% --- Daten laden ---
load mallorca_klima

% --- Plynomapproximation der Durchnschnittstemperatur ---
n = 5; % Polynomordnung
p = polyfit(Mallorca.Monat.Nummer,Mallorca.Temperatur.Mittel,n);
tage= linspace(1,12,365); % Vektor mit 365 Elemneten
T_p = polyval(p,tage); % Auswertung des Polynoms an 365 Tagen

% --- Zeichnen ---
figure(1)
       set(1,'position',[100 100 800 500])
       hold on
       bar(Mallorca.Monat.Nummer,Mallorca.Temperatur.Mittel) % Balkendiagramm der Daten
       plot(tage,T_p,'r','LineWidth',4); % Polynom
       grid on

% --- Beschriften ---
       ylabel(,Temperatur [°C]','FontSize',16)
       set(gca,'XTick',Mallorca.Monat.Nummer)
       set(gca,'XTickLabel',Mallorca.Monat.Name)
       title('Durchnschnittstemperatur auf Mallorca','FontSize',16)
       legend('Temperaturdaten','Approximation',2);
```

4.3 Numerische Differentiation und Integration

4.3.1 Aufgabe

- Erstellen Sie eine Funktion mit folgendem Funktionskopf:
  ```
  function [result] = diffAndInt(y_x,x,plotTyp,k)
  ```

- Das Argument `y_x` ist eine Zeichenkette, die von der `eval()` Funktion ausgewertet werden kann, also ein funktionaler Ausdruck in Abhängigkeit einer oder mehrerer Variablen. Wir wollen uns hier auf den Fall *einer Variablen* beschränken.

- Das Argument `x` ist ein *numerischer Vektor* mit streng monoton steigenden Elementen. Das Argument `y_x` soll von der `eval()` Funktion „entlang" dieses Vektors `x` ausgewertet werden.

- Der Rückgabewert `result` ist eine Struktur mit folgenden Feldern: `x`, `y`, `y_I`, `y_D` und `A`. Erklärung der Felder:
  - Das Feld `x` ist identisch mit dem Argument `x`.
  - Das Feld `y` ist die numerische Auswertung von `y_x` entlang `x`.
  - Das Feld `y_I` ist das numerische Integral von `y` entlang `x`
  - Das Feld `y_D` ist die numerische Ableitung von `y` entlang `x`.
  - Im Feld `A` steht das ausgewertete numerische Integral der Funktion `y` vom ersten bis zum letzten Element von `x`, also die Fläche unter der Funktion `x`. `A` ist somit ein Skalar.

- Darüber hinaus sollen die drei Funktionsverläufe `y, y_I` und `y_D` grafisch dargestellt werden. Die Art der Darstellung kann mittels `plotTyp` ausgewählt werden:
    - Steht in `plotTyp` eine 1 wird ein figure mit der Nummer `k` mit 400 Pixeln Breite und 800 Pixeln Höhe erzeugt. Jede Funktion wird mit `subplot()` in blau in ein *eigenes Koordinatensytsem* gezeichnet. Gitternetzlinien sollen dargestellt werden und jedes Koordinatensystem soll einen entsprechenden Titel haben: „*Integral Y(x)*“, „*Funktion y(x)*“, oder „*Ableitung y'(x)*“.
    - Steht in `plotTyp` eine 2 wird ein figure mit der Nummer `k` mit 600 Pixeln Breite und 600 Pixeln Höhe erzeugt. Alle drei Funktionen werden in *ein Koordinatensystem* gezeichnet in den Farben blau, grün und rot. Gitternetzlinien sollen daregstellt werden und die Grafik soll eine Legende mit den Beschriftungen „*Integral Y(x)*“, „*Funktion y(x)*“, und „*Ableitung y'(x)*“ enthalten.
    - Steht in `plotTyp` ein anderer Wert, so wird nichts gezeichnet.

4.3.2 Hinweise

- Wird die Funktion `diffAndInt()` folgendermaßen aufgerufen, sollten Sie das nachstehende Ergebnis erhalten.

```
>> [result] = diffAndInt(,2*x',[0:0.01:10],1,1)

result =

       x: [1x1001 double]
       y: [1x1001 double]
       y_I: [1x1001 double]
       y_D: [1x1001 double]
       A: 100
```

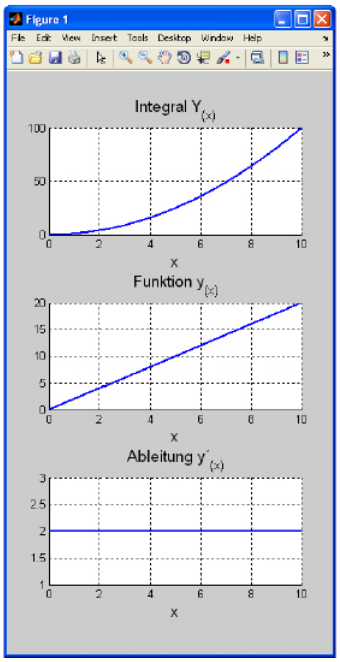


Abbildung 4: Numerische Differentiation und Integration

4.3.3 Lösung

```
function [result] = diffAndInt(y_x,x,plotTyp,n)
% ------------------------------------------------------------------------
% Übungsaufgaben Matlab und Simulink
% Kapitel 4
% Aufgabe 3: Numerische Differentiation und Integration
% ------------------------------------------------------------------------

% --- Integrieren und Differenzieren ---
result.x = x;
result.y = eval(y_x,x);
result.y_I = cumtrapz(x,result.y);
result.y_D = gradient(result.y,x);
result.A = trapz(x,result.y);

% --- Funktionen zeichnen ---
figure(n)
clf
       switch plotTyp
              case 1
                     set(n,'position',[50 50 400 800])
                            subplot(311)
                                   hold on
                                   plot(x,result.y_I,'b','LineWidth',2)
                                   grid on
                                   title('Integral Y_{(x)}','FontSize',14)
                                   xlabel('x','FontSize',14)
                            subplot(312)
                                   hold on
                                   plot(x,result.y,'b','LineWidth',2)
                                   grid on
                                   title('Funktion y_{(x)}','FontSize',14)
                                   xlabel('x','FontSize',14)
                            subplot(313)
                                   hold on
                                   plot(x,result.y_D,'b','LineWidth',2)
                                   grid on
                                   title('Ableitung y´_{(x)}','FontSize',14)
                                   xlabel('x','FontSize',14)
case 2
              set(n,'position',[50 50 600 600])
```

```
            hold on
            plot(x,result.y_I,'b','LineWidth',2)
            plot(x,result.y,'r','LineWidth',2)
            plot(x,result.y_D,'g','LineWidth',2)
            grid on
            xlabel('x','FontSize',14)
            legend('Integral Y_{(x)}','Funktion y_{(x)}','Ableitung y´_{(x)}')
        otherwise
end
```

4.4 Gleitender Mittelwert

4.4.1 Vorbereitung

Rufen Sie die in der vorigen Aufgabe erstellte Funktion `diffAndInt()` in den beiden folgenden Varianten auf:

```
x = [0:0.001:4*pi];
[result1] = diffAndInt('sin(x).^2',x,1,1)
[result2] = diffAndInt('sin(x).^2+0.03*randn(1,length(x))',x,1,2)
```

Beim ersten Aufruf wird die Funktion $y = sin^2(x)$ im Bereich $x = 0$ *bis* 4π ausgewertet, und beim zweiten Aufrufmal dieselbe Funktion, diesmal aber mit einem *normalverteilten Rauschen* überlagert. Was können Sie bezüglich der Qualität von numerischer Integration und numerischer Differentiation sagen, wenn *verrauschte Signale* vorliegen?

4.4.2 Hintergrund

Sie sollen ein Verfahren implementieren, um verrauschte Signale zu glätten, bzw. zu filtern. Als Filteralgorithmus soll der *Gleitende Mittelwert* (*engl.: moving average, siehe* http://de.wikipedia.org/wiki/Gleitender_Mittelwert) implementiert werden. Der gleitende Mittelwert ist ein *Tiefpassfilter*, welcher vor allem hohe Frequenzen (Rauschen) dämpft. Es gibt verschieden Ausführungsformen, beispielsweise der *einfache einseitige* oder der *einfache zentrierte* gleitende Mittelwert. Es soll hier der einfache einseitige gleitende Mittelwert verwendet werden.

Der einfache einseitige gleitende Mittelwert berechnet zu *jedem Element* eines Datenvektors *ein einziges gefiltertes Element*. Das gefilterte Element ist dabei einfach der arithmetische Mittelwert der n letzen Datenpunkte. Der Algorithmus „schiebt" sich also quasi durch den Datenvektor und berechnet an jeder Stelle einen neuen Mittelwert aus n Punkten. Bei diesem *einseitigen* Mittelwert ist dabei zu achten, dass dieses „Schieberegister" erst nach n Datenelementen komplett gefüllt ist und der arithmetische Mittelwert eigentlich erst dann „sinnvoll" angewendet werden kann. Die unbesetzten Elemente des Schieberegisters sollen deshalb mit *Nullen* initialisiert werden.

4.4.3 Aufgabe

- Erstellen Sie eine Funktion mit folgendem Funktionskopf:
  ```
  function [y_filtered] = movingAverage(y_noise,n)
  ```
  Diese soll den einfachen einseitigen gleitenden Mittelwert implementieren. `y_noise` ist der ungefilterte/ verrauschte Datenvektor und `y_filtered` der gefilterte Datenvektor. Die Länge des Schiebregisters soll `n` Datenpunkte betragen.

4.4.4 Analyse

- Verwenden Sie als Datenvektor `y_nosie` das verrauschte Signal `result2.y` aus der Vorbereitungsaufgabe. Rufen Sie die Funktion `movingAverage` mit folgenden Parameterwerten für *n* auf und plotten Sie das ungefilterte und das gefilterte Signal übereinander:

n = 5	n = 10	n = 20
n = 50	n = 100	n = 200
n = 500	n = 1000	n = 2000

- Welche Effekte können Sie beobachten?

4.4.5 Hinweise

- Es gibt verschiedene Möglichkeiten den Algorithmus mehr oder weniger elegant bzw. mehr oder weniger aufwendig zu implementieren.
- Sie können das Problem angehen wie Sie mögen, es kann jedoch sehr elegant über einen *Ringspeicher* gelöst werden. Informieren Sie sich gegebenenfalls darüber, was ein Ringspeicher ist (z.B.: http://de.wikipedia.org/wiki/Digitaler_Ringspeicher).
- Hierbei könnte Ihnen vielleicht auch die Modulo-Funktion `mod()` von Nutzen sein.

4.4.6 Lösung

```
function [y_filtered] = movingAverage(y_noise,n)
% ------------------------------------------------------------------------
% Übungsaufgaben Matlab und Simulink
% Kapitel 4
% Aufgabe 4: Gleitender Mittelwert
% ------------------------------------------------------------------------

y_noise = y_noise(:);            % Sicherstellen, dass y_noise als Spaltenvektor vorliegt
y_reg = zeros(n,1);              % Schieberegister der Länge n. mit Nullen initialisiert
for(k=1:1:length(y_noise))
       r = mod(k-1,n)+1;         % Index des Schieberegisters als Ringspeicher
       y_reg(r) = y_noise(k);
       y_filtered(k) = mean(y_reg);
end

figure(1)
       hold on
       plot(y_noise,'b','LineWidth',1)
       plot(y_filtered,'r','LineWidth',1)
       grid on
```

4.5 Galtonbrett

4.5.1 Hintergrund

Ein *Galtonbrett* (nach Francis Galton, siehe http://de.wikipedia.org/wiki/Galtonbrett) ist ein mechanisches Modell zur Demonstration und Veranschaulichung der Binomialverteilung, einer Wahrscheinlichkeitsverteilung, die in vielen Zufallsexperimenten eine Rolle spielt. Das Galtonbrett besteht aus einer regelmäßigen Anordnung von Hindernissen, an denen eine von oben eingeworfene Kugel jeweils mit der *gleichen Wahrscheinlichkeit* nach links oder rechts abprallen kann. Nach dem Passieren der Hindernisse werden die Kugeln in Fächern aufgefangen, um dort gezählt zu werden.

In der Natur entspricht das zum Beispiel der Rauschverteilung eines elektrischen Signals, verursacht durch sehr viele sehr kleine Störsignale. Ein grundlegendes mathematisches Gesetz, der zentrale Grenzwertsatz, garantiert, dass eine sehr beliebig zusammengesetzte Verteilung solcher sehr kleinen und sehr zahlreichen Einzelstörungen gegen die Gaußsche Glockenform konvergiert. Sind die Voraussetzungen für eine solche Rauschverteilung erfüllt, spricht man von *gaußschem Rauschen*. Bei einer endlichen Zahl von Störungen, wie beim Galtonbrett, erhält man die Binomialverteilung, die im Grenzwert vieler Störungen und vieler Fächer ebenfalls gegen die Gaußsche Glockenform konvergiert.

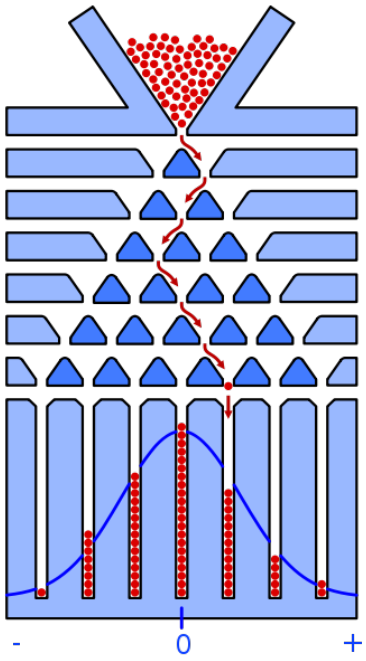


Abbildung 5: Galtonbrett

4.5.2 Aufgabe

- Erstellen Sie eine Funktion, welche ein Galtonbrett simuliert:
  `function [verteilung] = galtonbrett(ebenen,kugeln)`
- Über das Argument ebenen können Sie die Anzahl der Hindernisebenen festlegen, über kugeln die Anzahl an `Kugeln`, die in das Brett geworfen werden.
- Trifft eine (imaginäre) Kugel auf ein Hindernis, so soll diese mit gleicher Wahrscheinlichkeit nach „links“ oder nach „rechts“ fallen, bis diese schließlich in einem Gefäß landet. Im Vektor `verteilung` werden die Kugeln in jedem Gefäß gezählt. Darüber hinaus soll die Funktion am Ende ein Histogramm der Verteilung erstellen, um die Anzahl der Kugeln in den jeweiligen Gefäßen grafisch darzustellen.

4.5.3 Lösung

```
function [gefaess] = galtonbrett(ebenen,kugeln)
% ------------------------------------------------------------------------
% Übungsaufgaben Matlab und Simulink
% Kapitel 4
% Aufgabe 5: Galtonbrett
% ------------------------------------------------------------------------

gefaess(1,:)= [1:1:ebenen+1];     % Vektor mit der Gefaessnummer
gefaess(2,:)= zeros(1,ebenen+1); % Vektor mit Anzahl der Kugeln im Gefäß
verteilung = zeros(1,ebenen+1)'; % Vektor zum speichern der Kugelverteilung

for(k=1:1:kugeln)
      kugelposition = 0;  % --- aktuellen Kugelposition in der jew. Hindernisebene
                          % --- Kann positiv oder negativ sein
      for(e=1:1:ebenen)
            if(rand()>.5) % --- Zufallszahl > 0.5 --> Kugel fällt nach rechts
                  kugelposition = kugelposition+1;
            else          % --- Zufallszahl < 0.5 --> Kugel fällt nach links
                  kugelposition = kugelposition-1;
            end
      end
      gefaess_nr  =  ceil(kugelposition/2)  +  floor(ebenen/2)  +  1;  % Kugelposition  in
"Gefaessnummer" umrechnen
      gefaess(2,gefaess_nr) = gefaess(2,gefaess_nr)+1;
      verteilung(k) = gefaess_nr; % Die aktuelle Kugelposition in der Gesamtverteilung
aufnehmen
end

% --- Zeichnung ---
figure(1)
      hist(verteilung,gefaess(1,:))
      grid on
      title('Galtonbrett','FontSize',16)
      xlabel('Gefaess','FontSize',16)
      ylabel('Anzahl Kugeln','FontSize',16)
      axis([gefaess(1,1) gefaess(1,end) 0 ceil(max(gefaess(2,:))/100)*100])
```

4.6 NTC-Kennlinie 1

4.6.1 Hintergrund

Der ohmsche Widerstand eines NTC Sensorelementes hängt stark von Temperatur ab, es kann somit als Messfühler zur elektrischen Temperaturmessung eingesetzt werden. Je größer die Umgebungstemperatur, desto kleiner ist der Widerstandswert des NTC (Heißleiter). Nachfolgend ist die Kennlinie eines NTC Temperatursensors im Temperaturbereich von 0°C bis 80°C als Wertetabelle mit der Schrittweite 1°C angegeben

T / °C	R / Ω
0	32650
1	31028
2	29494
3	28048
4	26680
5	25388
6	24166
7	23010
8	21916
9	20880
10	19900
11	18968
12	18086
13	17252
14	16458
15	15708
16	14996
17	14320
18	13678
19	13068

T / °C	R / Ω
20	12490
21	11940
22	11418
23	10922
24	10450
25	10000
26	9572
27	9164
28	8778
29	8408
30	8058
31	7722
32	7402
33	7098
34	6808
35	6532
36	6268
37	6016
38	5776
39	5546

T / °C	R / Ω
40	5326
41	5118
42	4918
43	4726
44	4544
45	4368
46	4202
47	4042
48	3890
49	3742
50	3604
51	3468
52	3340
53	3218
54	3100
55	2986
56	2878
57	2774
58	2674
59	2580

T / °C	R / Ω
60	2488
61	2400
62	2316
63	2236
64	2158
65	2084
66	2012
67	1943
68	1877
69	1813
70	1752
71	1694
72	1637
73	1583
74	1531
75	1481
76	1433
77	1387
78	1342
79	1299
80	1258

4.6.2 Aufgabe

- Schreiben Sie eine Funktion mit folgendem Funktionskopf:
  `function [p] = NTC_approximation(R_max)`
- Die Funktion führt eine *Polynomapproximation* der Kennlinie durch. Dabei soll beginnend bei einem Polynom nullter Ordnung, die Ordnung des Approximationspolynoms schrittweise erhöht werden, bis der Betrag der Abweichung zwischen den Originaldaten der Kennlinie und dem Approximationspolynom maximal den Wert `R_max` beträgt. Die Funktion liefert das entsprechende Polynom als Vektor `p` zurück.
- Darüber hinaus soll die Funktion einen Plot erstellen. Im `subplot(121)` wird die Kennlinie als blaue Punktefolge gezeichnet. Die Approximationsfunktion wird als durchgehende Linie in rot darüber gezeichnet. Erstellen Sie zudem eine Legende.
- Im `subplot(122)` stellen Sie die *Sensitivität* des Sensorelements in Ω / °C dar. Die Sensitivität ist die Ableitung der Widerstand-Temperatur-Kennlinie nach der Temperatur. Stellen Sie die Funktion ebenfalls als durchgehende rote Linie dar und beschriften Sie die Achsen entsprechend.

4.6.3 Hinweis

Wird die Funktion mit dem Parameter `R_max = 100` aufgerufen, so sollte der Plot also ungefähr so aussehen:

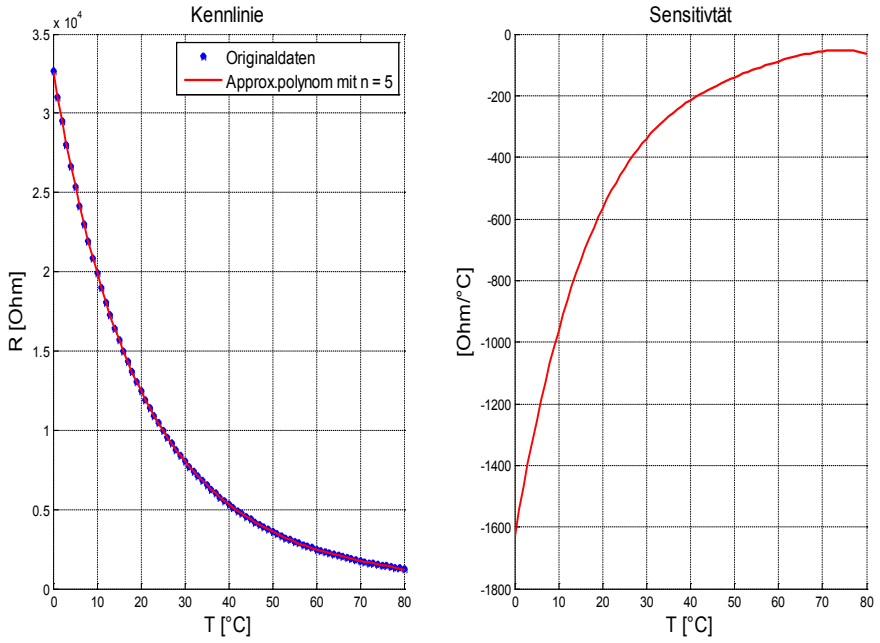


Abbildung 6: NTC-Kennlinie

4.6.4 Lösung

```
function [p] = NTC_approximation(R_max)
% ------------------------------------------------------------------------
% Übungsaufgaben Matlab und Simulink
% Kapitel 4
% Aufgabe 1: NTC-Kennlinie 1
% ------------------------------------------------------------------------
close all

% --- Die Daten in der Excel-Tabelle 'NTC_Kennlinie.xls' abgespeichert ---
[numeric,text,raw] = xlsread('NTC_Kennlinie.xls');

T = numeric(:,1);
Rn= numeric(:,2);

string_x = text{1};
string_y = text{2};

% --- Geeignetes Polynom finden ---
for(n=1:1:10);
       p = polyfit(T,Rn,n);
       Rp= polyval(p,T);
       if(max(abs(Rn-Rp))<abs(R_max))
              break
       end
end
% --- Sensitivität berechnen ---
p_diff = polyder(p);
s = polyval(p_diff,T);
```

```
% --- Plot erstellen ---
figure(1)
set(1,'position',[100 250 1100 600])
        subplot(121)
                hold on
                plot(T,Rn,'bx','LineWidth',3)
                plot(T,Rp,'r','LineWidth',2)
                grid on
                title('Kennlinie','FontSize',14)
                h = legend('Originaldaten',['Approx.polynom mit n = ',num2str(n)]);
                set(h,'FontSize',12)
                xlabel(string_x,'FontSize',14)
                ylabel(string_y,'FontSize',14)
        subplot(122)
                hold on
                plot(T,s,'r','LineWidth',2)
                grid on
                title('Sensitivtät','FontSize',14)
                xlabel(string_x,'FontSize',14)
                ylabel('[Ohm/°C]','FontSize',14)
```

4.7 NTC-Kennlinie 2

4.7.1 Aufgabe

- Erstellen Sie eine Funktion mit folgendem Funktionskopf:
  `function [T,exceed] = NTC_sensor(R)`
- Die Funktion berechnet zu einem beliebigen Widerstandswert `R` (Gleitkommazahl) zwischen *R0* und *R80* den korrespondierenden Temperaturwert `T`. Verwenden Sie das Verfahren der *Linearen Interpolation* um zwischen den Stützpunkten der Wertetabelle zu interpolieren.
- Liegt der Wert `R` außerhalb des außerhalb der Wertetabelle wird entweder `T = 0` oder `T = 80` zurückgeliefert und der Rückgabewert `exceed` zu `-1` bzw. `+1` gesetzt. Ist der Widerstandswert im zulässigen Bereich wird `exceed = 0` zurückgegeben.

4.7.2 Hinweis

- Die lineare Interpolation erfolgt nach folgender Formel:

$$y_A = y_1 + \frac{y_2 - y_1}{x_2 - x_1} \cdot (x_A - x_1)$$

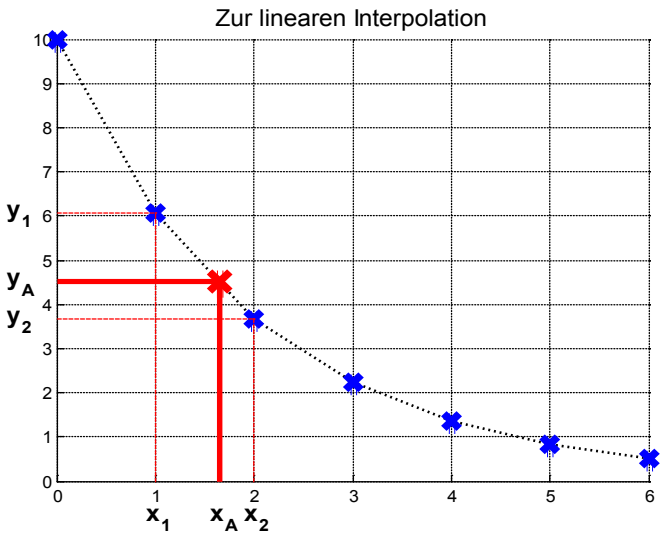


Abbildung 7: Lineare Interpolation

4.7.3 Lösung

```
function [T,exceed] = NTC_sensor(R)
% ------------------------------------------------------------------------
% Übungsaufgaben Matlab und Simulink
% Kapitel 4
% Aufgabe 7: NTC-Kennlinie 2
% ------------------------------------------------------------------------

% --- Kennlinie einlesen ---
      [numeric,text,raw] = xlsread('NTC_Kennlinie.xls');

% --- Temperaturvektor "t" und Widerstandsvektor "r" ---
      t = numeric(:,1);
      r = numeric(:,2);

% --- Minimal- und Maximalwerte ---
      R_max = r(1);
      R_min = r(end);
      T_max = t(end);
      T_min = t(1);

% --- Rückgabewerte bestimmen ---
if(R>R_max)          % Widerstandswert zu groß -> Temperatur kleiner 0°C
      exceed = -1;
      T = T_min;
elseif(R<R_min)      % Widerstandswert zu klein -> Temperatur größer 80°C
      exceed = +1;
      T = T_max;
else                 % Temperatur zwischen 0°C und 80°C
      exceed = 0;
      index = find(r>=R,1,'last');
      R1 = r(index);
      R2 = r(index+1);
      T1 = t(index);
      T2 = t(index+1);
      T = T1 + (T2-T1)/(R2-R1)*(R-R1); % Lineare Interpolation
end
```

4.8 Produkt- und Quotientenregel

4.8.1 Aufgabe

- Bestätigen Sie mit der Symbolic Math Toolbox die Produkt- und die Quotientenregel:

Produktregel: $$\left(f_{(x)} \cdot g_{(x)}\right)' = f_{(x)}' \cdot g_{(x)} + f_{(x)} \cdot g_{(x)}'$$

Quotientenregel: $$\left(\frac{f_{(x)}}{g_{(x)}}\right)' = \frac{f_{(x)}' \cdot g_{(x)} - f_{(x)} \cdot g_{(x)}'}{g_{(x)}^2}$$

4.8.2 Lösung

```
% ------------------------------------------------------------------------
% Übungsaufgaben Matlab und Simulink
% Kapitel 4
% Aufgabe 8: Produkt- und Quotientenregel
% ------------------------------------------------------------------------

syms f g x
f = sym('f(x)');
g = sym('g(x)');

produktregel = diff(f*g,x)
quotientenregel = diff(f/g,x)
```

4.9 Schnittpunkte von Funktionen

4.9.1 Aufgabe

- Ermitteln Sie mit der Symbolic Math Toolbox die Schnittpunkte der quadratischen Parabel $f_{1(x)}$ mit den Geraden $g_{(x)}$ und die Schnittpunkte der kubischen Parabel $f_{2(x)}$ mit der Geraden $g_{(x)}$.

$$f_{1(x)} = x^2 + 3x$$

$$f_{2(x)} = x^3$$

$$g_{(x)} = 2x + 5$$

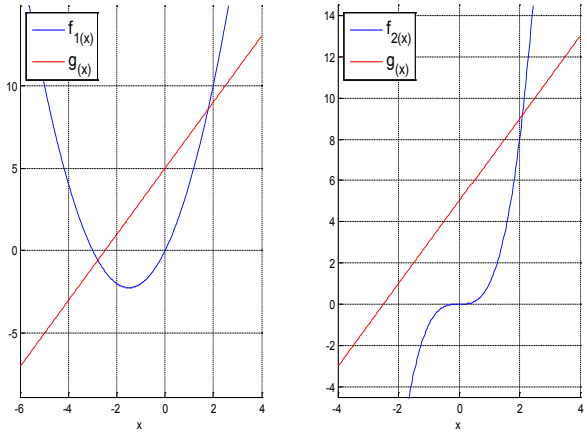


Abbildung 8: Schnittpunkte von Funktionen

4.9.2 Analyse

- Wie viele Schnittpunkte erhalten Sie jeweils formal mathematisch?
- Wie sind diese geometrisch zu interpretieren?

4.9.3 Lösung

```
% ------------------------------------------------------------------------
% Übungsaufgaben Matlab und Simulink
% Kapitel 4
% Aufgabe 9: Schnittpunkte von Funktionen
% ------------------------------------------------------------------------

clear
clc
close all

% --- Smbolische Funktionen definieren ---
syms x
f1 = x^2+3*x;
f2 = x^3;
g = 2*x+5;

% --- Schnittpunkte berechnen ---
s1 = eval(solve(f1-g))
s2 = eval(solve(f2-g))

% --- Plot ---
figure(1)
set(1,'position',[50 200,900,500])
       subplot(121)
              hold on
              ezplot(f1,[-6 4])
              hand = ezplot(g,[-6 4]);
              set(hand,'color',[1 0 0]);
              grid on
              title(,')
              hand = legend(,f_{1(x)}','g_{(x)}',2);
              set(hand,'FontSize',14)
       subplot(122)
              hold on
              ezplot(f2)
              hand = ezplot(g,[-4 4]);
              set(hand,'color',[1 0 0]);
              grid on
              title(,')
              hand = legend('f_{2(x)}','g_{(x)}',2);
       set(hand,'FontSize',14)
```

4.10 Endliche Reihen

4.10.1 Aufgabe

- Beweisen Sie mit der Symbolic Math Toolbox,

a) dass die Summe der ersten n ungeraden natürlichen Zahlen gleich n^2 ist,

b) dass die Summe der Quadrate der ersten n ungeraden natürlichen Zahlen gleich $4/3 \cdot n^3 - 1/3 \cdot n$ ist.

4.10.2 Lösung

```
% ------------------------------------------------------------------------
% Übungsaufgaben Matlab und Simulink
% Kapitel 4
% Aufgabe 10: Endliche Reihen
% ------------------------------------------------------------------------

clear
clc
close all

syms k n

% --- a) Summe der ersten n ungeraden Zahlen ---
s1 = symsum((2*k-1),1,n);
s1 = simplify(s1);

% --- b) Summe der Quadrate der ersten n ungeraden Zahlen ---
s2 = symsum((2*k-1)^2,1,n);
s2 = simplify(s2);
```

4.11 Gebrochen Rationale Funktion

4.11.1 Aufgabe

Gegeben sei folgende gebrochen rationale Funktion:

$$f_{(x)} = \frac{p_{(x)}}{q_{(x)}} = \frac{2x^3 + 20x^2 + 64x + 64}{x^4 + 8x^3 + 18x^2 - 27}$$

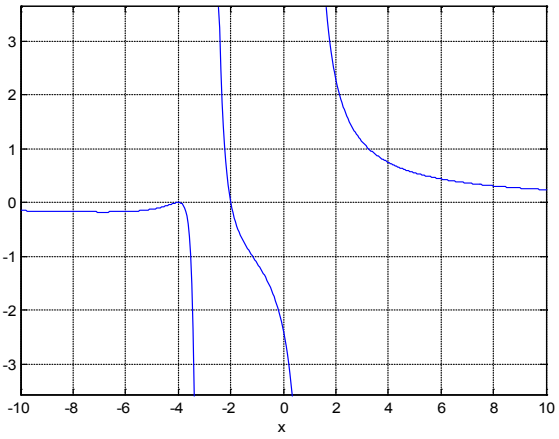


Abbildung 9: Gebrochen rationale Funktion

a) Bestimmen Sie den Funktionswert von *f(x)* an der Stelle $x = 4$.
b) Bestimmen Sie die Fläche unter der Funktion für mit $a = +2$ als untere und $b = +8$ als obere Integrationsgrenze.
c) Bestimmen Sie Nullstellen des Zählerpolynoms *p(x)* und die Nullstellen des Nennerpolynoms *q(x)* (= Polstellen).
d) Überprüfen Sie (automatisiert), ob die Funktion *mehrfache Polstellen* hat. Speichern Sie in einem Zeilenvektor `polstellen` nur die *unterschiedlichen Polstellen*.
e) Erweitern Sie den Vektor `polstellen` um zwei Zeilen. In der zweiten Zeile steht der linksseitige Grenzwert der jeweiligen Polstelle und in der dritten Zeile der rechtsseitige Grenzwert der jeweiligen Polstelle.
f) Bestimmen Sie den Grenzwert der Funktion für $x \to +\infty$ und $x \to -\infty$.
g) Plotten Sie die Ableitung der Funktion, die Funktion selbst und deren Stammfunktion in Bereich von $x = -10$ bis $x = +10$.

4.11.2 Hinweis

- Sehen Sie sich gegebenenfalls die Dokumentation zum Befehl `limit()` an.

4.11.3 Lösung

```
% ------------------------------------------------------------------------
% Übungsaufgaben Matlab und Simulink
% Kapitel 4
% Aufgabe 11: Gebrochen rationale Funktion
% ------------------------------------------------------------------------

clear
clc
close all

syms f p q x

% --- Funktion f(x) definieren ---
p = 2*x^3 + 20*x^2 + 64*x + 64;
q = 1*x^4 + 8*x^3 + 18*x^2 -27;
f = p/q;
% --------------------------------

% --- a) Funktionswert ---
f4 = subs(f,x,4)          % Funktionswert an der Stelle x = 4

% --- b) Fläche unter der Kurve ---
A = eval(int(f,x,2,8))    % Bestimmtes Integral a=-2, b=+8
```

```
% --- c) Nullstellen bestimmen ---
zeros_p = solve(p,x)       % Zählerpolnom
zeros_q = solve(q,x)       % Nennerpolynom

% --- d) unterschieldiche Polstellen bestimmen ---
i = 1;
polstellen(i) = zeros_q(i);
for(k=1:1:length(zeros_q)-1)
       if(zeros_q(k+1)~=zeros_q(k))
              i=i+1;
              polstellen(i) = zeros_q(k+1);
       end
end

% --- e) Grenzwerte an den Polstellen ---
for(k=1:1:length(polstellen))
       polstellen(2,k) = limit(f,x,polstellen(1,k),'left');
       polstellen(3,k) = limit(f,x,polstellen(1,k),'right');
end
polstellen
% --- f) Grenzwerte für x -> unendlich bestimmen ---
lim_inf_pos = limit(f,x,inf)            % Grenzwert x -> + unendlich
lim_inf_neg = limit(f,x,-inf)           % Grenzwert x -> - unendlich

% --- g) Plot der Funktion ---
df = diff(f,x);                         % f'(x) Ableitung
F = int(f,x);                           % F(x) Stammfunktion
figure(1)
set(1,'position',[50 250 1200 500])
       subplot(131)
              ezplot(df,[-10 10])
              title('f´_{(x)}')
              grid on
       subplot(132)
              ezplot(f,[-10 10])
              title('f_{(x)}')
              grid on
       subplot(133)
              ezplot(F,[-10 10])
              title('F_{(x)}')
              grid on
```

5 Differentialgleichungen

5.1 Populationsdynamik nach Lotka-Volterra

5.1.1 Hintergrund

Mit dem *Lotka-Volterra Modell* soll die Populationsdynamik von Füchsen und Hasen in einem abgeschlossenen Ökosystem untersucht werden. Diese Populationsdynamik kann dabei durch folgendes nichtlineares Differentialgleichungssystem (Anfangswertproblem) beschrieben werden:

$$\dot{N}_{1(t)} = +\alpha_1 \cdot N_{1(t)} - \beta_1 \cdot N_{1(t)} \cdot N_{2(t)}$$

$$\dot{N}_{2(t)} = +\beta_2 \cdot N_{1(t)} \cdot N_{2(t)} - \alpha_2 \cdot N_{2(t)}$$

N_1 entspricht der Beutepopulation Hasen und N_2 der Räuberpopulation Füchse.

5.1.2 Aufgabe

- Simulieren Sie mittels `ode45()` die Populationsdynamik im Zeitraum von fünf Jahren `(timespan = [0 5])`.
- Stellen Sie das Ergebnis grafisch dar.
- Die Anfangspopulationen bestehen aus 12 Hasen und 4 Füchsen.
- Verwenden Sie die folgenden Systemparameter:

  $\alpha 1 = 5$ pro Jahr

  $\alpha 2 = 5$ pro Jahr

  $\beta 1 = 0.5$ pro Fuchs und Jahr

  $\beta 2 = 0.5$ pro Hase und Jahr

5.1.3 Lösung

5.1.3.1 Implementierung der DGL

```
function [dN] = lotka_volterra(t,N,a1,a2,b1,b2)
% ------------------------------------------------------------------------
% Übungsaufgaben Matlab und Simulink
% Kapitel 5
% Aufgabe 1: Populationsdynamik nach Lotka-Volterra
% ------------------------------------------------------------------------

dN = zeros(2,1);    % Lösungsvektor als Spaltenvektor initialisieren
dN(1) = +a1*N(1)-b1*N(1)*N(2);   % 1. Gleichung des DGL-Systems
dN(2) = +b2*N(1)*N(2)-a2*N(2);   % 2. Gleichung des DGL-Systems
```

5.1.3.2 Lösung der DGL

```
% ------------------------------------------------------------------------
% Übungsaufgaben Matlab und Simulink
% Kapitel 5
% Aufgabe 1: Populationsdynamik nach Lotka-Volterra
% ------------------------------------------------------------------------

clear
clc
close all

% --- Systemparameter ---
      a1 = 5;
      a2 = 5;
      b1 = .5;
      b2 = .5;

% --- Start- und Endzeitpunkt ---
      timespan = [0 5];   % [Jahre]

% --- Anfangswerte ---
      IC = [12 4];        % [Hasen Füchse]

% --- Aufruf des Lösungsalgorithmus ---
      [t,y] = ode45(@lotka_volterra,timespan,IC,[],a1,a2,b1,b2);

% --- Ergebnis plot ---
figure(1)
      hold on
      plot(t,y(:,1),'b','LineWidth',2)
      plot(t,y(:,2),'r','LineWidth',2)
      plot([t(1) t(end)],[0 0],'k--','LineWidth',2)
      grid on
      axis([t(1) t(end) 0 25])
      title('Lotka-Volterra Model','FontSize',16)
      xlabel('t / s','FontSize',16)
      ylabel('N_{(t)}','FontSize',16)
      h = legend('Hasen','Füchse',4);
      set(h,'FontSize',12)
```

5.2 SIR-Modell

5.2.1 Hintergrund

Als SIR-Modell (*Susceptible-Infected-Recovered-Model,* siehe: http://de.wikipedia.org/wiki/SIR-Modell) bezeichnet man in der mathematischen Epidemiologie, einem Teilgebiet der theoretischen Biologie, einen klassischen Ansatz zur Beschreibung der Ausbreitung von ansteckenden Krankheiten mit Immunitätsbildung. Dabei wird eine in ihrer Gesamtgröße N als konstant veranschlagte Population aufgeteilt in

1. *gesunde Individuen*, die krank werden können (susceptible individuals S),
2. *bereits erkrankte und ansteckende Individuen* (infectious individuals I) und
3. *bereits immunisierte* (oder verstorbene) *Individuen* (resistant individuals R).

Die Inkubationszeit wird dabei als vernachlässigbar kurz betrachtet. Die Ausbreitung der betrachteten Krankheit wird in Form von gewöhnlichen Differentialgleichungen mit einer *Erkrankungsrate* c und einer *Gesundungsrate* w formuliert:

$$\dot{S}_{(t)} = -c \cdot I_{(t)} \cdot S_{(t)}$$
$$\dot{I}_{(t)} = +c \cdot I_{(t)} \cdot S_{(t)} - w \cdot I_{(t)}$$
$$\dot{R}_{(t)} = +w \cdot I_{(t)}$$

5.2.2 Aufgabe

- Simulieren Sie basierend auf dem SIR-Modell mittels `ode45()` die Ausbreitung einer ansteckenden Krankheit über drei Monate hinweg `(timespan = [0 3])` in einer Stadt mit 100.000 gesunden Menschen, die von einer einzigen infizierten Person besucht wird. Gehen Sie davon aus, dass aufgrund einer sofort ausgerufenen Quarantäne niemand das Stadtgebiet betreten oder verlassen kann.
- Stellen Sie das Ergebnis grafisch dar.
- Verwenden Sie die folgenden Parameter und Anfangsbedingungen:

$c = 0.0002$ pro Monat und Mensch

$w = 10$ pro Monat

$S_0 = 10^5$ Menschen

$I_0 = 1$ Menschen

$R_0 = 0$ Menschen

5.2.3 Lösung

5.2.3.1 Implementierung der DGL

```
function [dy] = sir(t,y,c,w)
% ------------------------------------------------------------------------
% Übungsaufgaben Matlab und Simulink
% Kapitel 5
% Aufgabe 2: SIR-Modell
% ------------------------------------------------------------------------

dy = zeros(3,1); % Lösungsvektor als Spaltenvektor initialisieren

dy(1) = -c*y(2)*y(1);            % S'(t) (1. Gleichung des DGL-Systems)
dy(2) = +c*y(2)*y(1)-w*y(2);     % I'(t) (2. Gleichung des DGL-Systems)
dy(3) = +w*y(2);                 % R'(t) (3. Gleichung des DGL-Systems)
```

5.2.3.2 Lösung der DGL

```
% ------------------------------------------------------------------------
% Übungsaufgaben Matlab und Simulink
% Kapitel 5
% Aufgabe 2: SIR-Modell
% ------------------------------------------------------------------------
clear
clc
close all

% --- Systemparameter ---
c = .0002;    % Erkrankungsrate
w = 10;       % Gesundungsrate

% --- Start- und Endzeitpunkt ---
timespan = [0 3]; % [Monate]

% --- Anfangswerte ---
IC = [1e5 1 0];     % [Gesunde(S) Erkrankte(I) Immunisierte(R)]

% --- Aufruf des Lösungsalgorithmus ---
[t,y] = ode45(@sir,timespan,IC,[],c,w);

% --- Ergebnis plot ---
figure(1)
       hold on
       plot(t,y(:,1),'r','LineWidth',2)
       plot(t,y(:,2),'g','LineWidth',2)
       plot(t,y(:,3),'b','LineWidth',2)
       plot(t,y(:,1)+y(:,2)+y(:,3),'k','LineWidth',1)
       plot([t(1) t(end)],[0 0],'k--','LineWidth',2)
       grid on
       axis([t(1) t(end) 0 1.3e5])
       title('SIR-Modell Grippe','FontSize',16)
       xlabel('t [Monate]','FontSize',16)
       h = legend('Gesunde Menschen','Kranke Menschen','Immunisierte Menschen');
       set(h,'FontSize',12)
```

5.3 Explizites Euler Cauchy-Verfahren

5.3.1 Hintergrund

In dieser Aufgabe sollen Sie ihren eigenen Solver zur Lösung gewöhnlicher Differentialgleichungen nach dem *expliziten Euler-Cauchy* Verfahren schreiben. Das explizite Euler-Cauchy Verfahren ist durch folgenden rekursiven Algorithmus gegeben:

$$y_{k+1} = y_k + \dot{y}_k \cdot \Delta t$$

Hierbei ist Δt die *konstante* Schrittweite des Solvers. Sie können den Solver so entwerfen, dass dieser ganze *Differentialgleichungssysteme* erster Ordnung löst, für diese Aufgabe reicht es jedoch aus, wenn der Solver die nachfolgende einzelne Differentialgleichung lösen kann:

$$\dot{y}_{(t)} = -a \cdot y_{(t)}$$

Die analytische Lösung dieser DGL mit der Anfangsbedingung *y0* ist durch folgende Funktion gegeben:

$$y_{(t)} = y_0 \cdot e^{-a \cdot t}$$

5.3.2 Aufgabe

- Schreiben Sie eine Funktion `ode()`, welche obige Differentialgleichung implementiert. `a` ist dabei ein *positiver reeller Skalar* (Anmerkung: geht mit zwei Zeilen Code). De Funktionskopf soll folgendermaßen aussehen:

```
function [dy] = ode(y,a)
```

- Schreiben Sie die Solver Funktion `ode_solver()`, welche das Euler-Cauchy Verfahren implementiert. Der Funktionskopf soll folgendermaßen aussehen.

```
function [t,y] = ode_solver(fh,dt,tmax,IC,par)
```

`fh` ist ein Function Handle der zu lösenden Differentialgleichung.
`dt` ist die konstante Schrittweite, mit welcher der Solver arbeitet.
`tmax` ist der Endzeitpunkt der Lösung.
`IC` sind die zur DGL gehörenden Anfangsbedingungen. In diesem Fall also *y0*.
`par` sind die zur DGL gehörenden Parameter. In diesem Fall also lediglich der Skalar *a*.

- Schreiben Sie ein Skript File `dgl_loesen`, das

a) ein *Function Handle* zur Differentialgleichung `ode()` generiert.
b) den Endzeitpunkt der Lösung `tmax`, die Anfangsbedingungen `IC`, die Parameter der DGL `par` und die Schrittweite des Solvers `dt` festlegt.
c) die Differentialgleichung `ode()` mittels `ode_solver()` löst.
d) die *numerisch berechnete Lösungsfunktion* grafisch in `rot` darstellt.
e) Generieren Sie darüber hinaus einen zweiten Zeitvektor `t_ana = [0:dt/1000:tmax]` und werten Sie die *analytische Lösungsfunktion* an diesen Zeitpunkten aus. Zeichnen Sie die analytische Lösungsfunktion *in blau* in dasselbe Diagramm.

- Verwenden Sie folgende Parameter:

```
tmax= 10
IC = 1
par = 1
dt = 0.0001
```

5.3.3 Analyse

- Führen Sie ihr Skript File dgl_loesen nacheinander mit folgenden Schrittweiten Δt aus:

$\Delta t = 0.001$	$\Delta t = 1$	$\Delta t = 3$
$\Delta t = 0.01$	$\Delta t = 1.5$	$\Delta t = 5$
$\Delta t = 0.1$	$\Delta t = 2$	$\Delta t = 10$

- Wie verhält sich die numerische Lösung im Vergleich zu der analytischen (= korrekten) Lösung mit den verschiedenen Schrittweiten des Solvers *qualitativ* und *quantitativ*? Wie hängt also die sogenannte *numerische Stabilität* des Lösungsalgorithmus von der Schrittweite Δt ab?

5.3.4 Lösung

5.3.4.1 Implementierung der DGL

```
function [dy] = ode(y,a)
% ------------------------------------------------------------------------
% Übungsaufgaben Matlab und Simulink
% Kapitel 5
% Aufgabe 3: Explizites Euler-Cauchy Verfahren
% ------------------------------------------------------------------------

            dy = -a*y;
```

5.3.4.2 Solverfunktion

```
function [t,y] = ode_solver(fh,dt,tmax,IC,par)
% ------------------------------------------------------------------------
% Übungsaufgaben Matlab und Simulink
% Kapitel 5
% Aufgabe 3: Explizites Euler-Cauchy Verfahren
% ------------------------------------------------------------------------

t = [0:dt:tmax]';          % Zeitvektor von 0 bis tmax, Schrittweite dt
m = length(t);             % Lösungsmatrix hat m Zeilen
n = length(IC);            % Lösungsmatrix hat n Spalten
y = zeros(m,n);            % Lösungsmatrix initialisieren
y(1,:) = IC;               % Erste Zeile der Lösung = Anfangsbedingungen

% --- Die Lösung der DGL rekursiv nach dem Euler-Cauchy Verf. berechnen ---
for(k = 1:1:length(t)-1)
      dy = fh(y(k),par);
      y(k+1) = y(k) + dy*dt;
end
```

5.3.4.3 Lösung der DGL

```
% ------------------------------------------------------------------------
% Übungsaufgaben Matlab und Simulink
% Kapitel 5
% Aufgabe 3: Explizites Euler-Cauchy Verfahren
% ------------------------------------------------------------------------
clear
clc
```

```
close all

% --- Function Handle der gewünschten DGL-Funktion ---
fh = @ode;

% --- Solvereinstellungen, Parameter und Anfangsbedingungen ---
dt = 1e-3;   % konstante Schrittweite des solvers
tmax= 10;    % Endzeitpunkt der Lösung
IC = 1;      % Anfangsbedingungen der DGL
par = 1;     % Parameter der DGL

% --- Solver zur numerischen Lösung der DGL aufrufen ---
[t_num,y_num] = ode_solver(fh,dt,tmax,IC,par);

% --- Analytische Lösungsfunktion ---
t_ana = [0:dt/1000:tmax];
y_ana = IC*exp(-par*t_ana);

% --- Plot der Lösung ---
figure(1)
       hold on
       plot(t_num,y_num,'r','LineWidth',2)
       plot(t_ana,y_ana,'b','LineWidth',1)
       grid on
       xlabel('t','FontSize',16)
       ylabel('y','FontSize',16)
       legend('numerisch','analytisch')
```

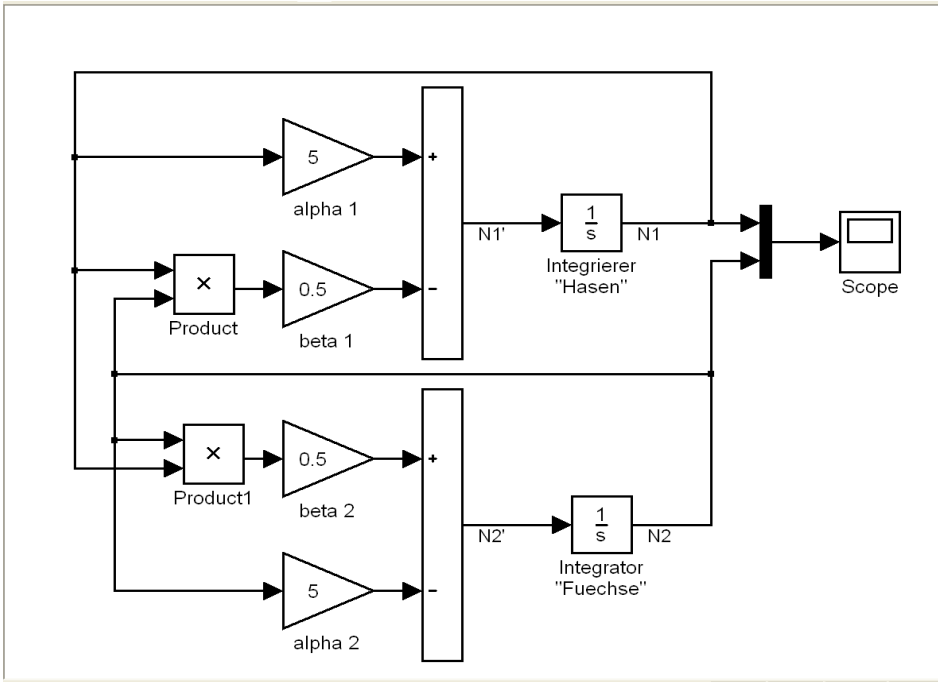
6 Simulink

6.1 Populationsdynamik nach Lotka-Volterra

6.1.1 Aufgabe

- Implementieren Sie das Lotka-Volterra Modell in Simulink. Verwenden Sie die Systemparameter und Anfangsbedingungen aus Aufgabe 5.1.

6.1.2 Lösung



6.1.3 Anmerkung

- Die Anfangsbedingungen (*Initial Conditions*) sind in den beiden Integrieren festzulegen.

6.2 Gleichstrommotor

6.2.1 Hintergrund

Eine Gleichstromnebenschlussmaschine (GNM) ist ein elektrischer Gleichstromantrieb, welcher über die Ankerspannung $U_A(t)$ angesteuert wird. Abhängig von $U_A(t)$ und vom an der Motorwelle angreifenden Lastmoment $M_L(t)$ stellt sich eine bestimmte Drehzahl $\Omega(t)$ ein. Die Ankerspannung und das Lastmoment sind also *Eingangssignale*, die Drehzahl das interessierende *Ausgangssignal*.

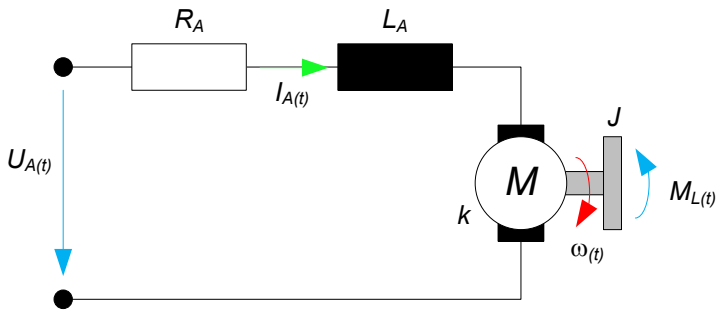


Abbildung 10: Gleichstrommotor

Das dynamische Verhalten einer GNM kann näherungsweise durch die beiden folgenden linearen Differentialgleichungen beschrieben werden:

$$\left(\frac{\partial}{\partial t}\omega_{(t)}\right) = \frac{1}{J}\cdot\left(k\cdot I_{A(t)} - M_{L(t)}\right)$$

$$\left(\frac{\partial}{\partial t} I_{A(t)}\right) = \frac{1}{L_A}\cdot\left(U_{A(t)} - k\cdot\omega_{(t)}\right) - \frac{R_A}{L_A}\cdot I_{A(t)}$$

Die Systemparameter sind durch folgende Zahlenwerte gegeben:

Symbol	Bedeutung	Wert	Einheit
R_A	Ankerwiderstand	2.0	Ω
L_A	Ankerinduktivität	0.1	H
k	Drehmomentkonstante	1.5	Nm/A
J	Trägheitsmoment an der Motorwelle	0.05	kgm^2

Die im Modell vorkommenden Signale haben die folgenden Einheiten:

Symbol	Bedeutung	Typ	Einheit
U_A	Ankerspannung	Eingangssignal	V
I_A	Ankerstrom	Inneres Signal	A
M_L	Lastmoment	Eingangssignal	Nm
ω	Drehgeschwindigkeit	Ausgangssignal	rad/s

6.2.2 Aufgabe

- Erstellen Sie ein Simulink Modell des Gleichstromantriebs. *Maskieren* Sie ihr Modell, so dass Sie es mit den Systemparametern parametrieren können. Erstellen Sie gegebenenfalls zunächst ein Blockschaltbild des Systems auf Papier.

6.2.3 Analyse

- Simulieren Sie das Modell mit einer konstanten Ankerspannung von U_A = 48 V und M_L = 0Nm als Eingangssignale. Betrachten Sie das Simulationsergebnis über einen *Scope* Block.

a) Welche stationäre Drehzahl wird nach dem Einschwingverhalten erreicht?

b) Nach welcher Zeit wird ungefähr die stationäre Drehzahl erreicht, ändert sich also nicht mehr signifikant?

- Untersuchen Sie den Einfluss aller vier Systemparameter auf das Systemverhalten, indem Sie für alle vier Systemparameter nacheinander größere (↑) und kleinere Werte (↓)wählen. Welchen Einfluss hat ein konstantes äußeres Lastmoment auf das

a) *Stationäre Verhalten?*
Ist die stationär erreichte Drehzahl also größer (↑), kleiner (↓) oder bleibt sie gleich (-)?

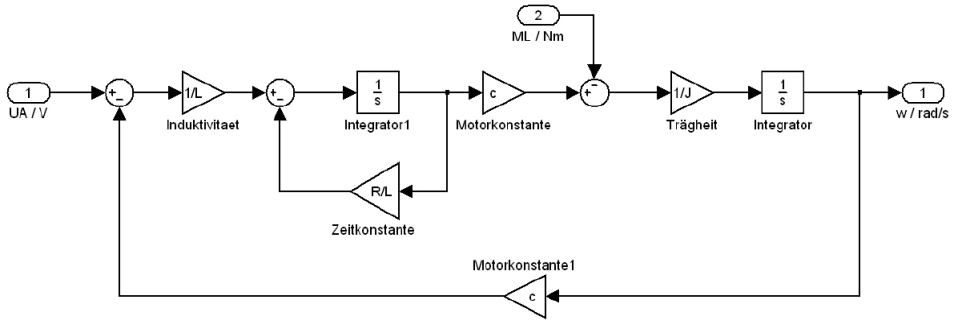
b) *Dynamische Verhalten?*
Erfolgt also die Systemreaktion schneller (↑), langsamer (↓) oder bleibt sie gleich (-)?

- Füllen Sie hierzu die nachfolgende Tabelle aus:

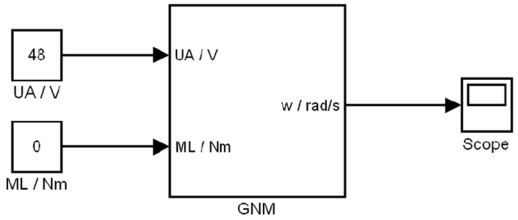
Parameter	Ankerwiderstand		Ankerinduktivität		Drehmomentkonstante		Trägheitsmoment	
	$R_A \uparrow$	$R_A \downarrow$	$L_A \uparrow$	$L_A \downarrow$	$k \uparrow$	$k \downarrow$	$J \uparrow$	$J \downarrow$
stationäre Drehzahl								
dynamisches Verhalten								

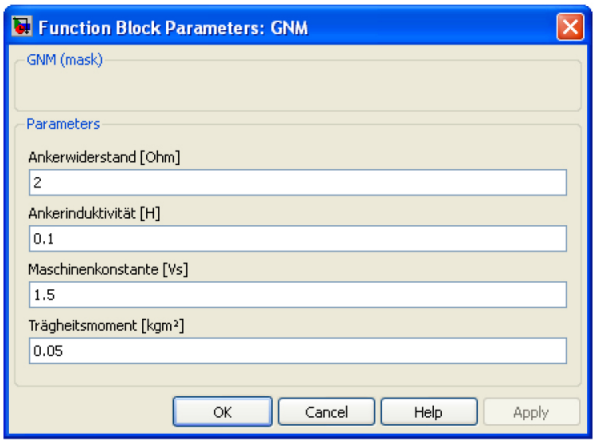
6.2.4 Lösung

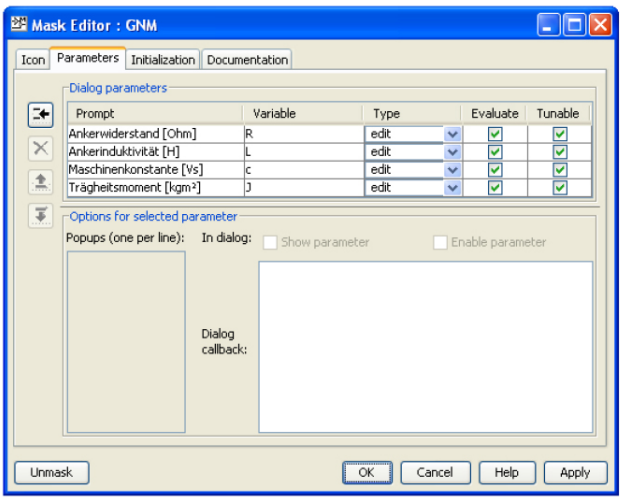
6.2.4.1 Implementierung des Differentialgleichungssystems



6.2.5 Subsystem mit Maskierung





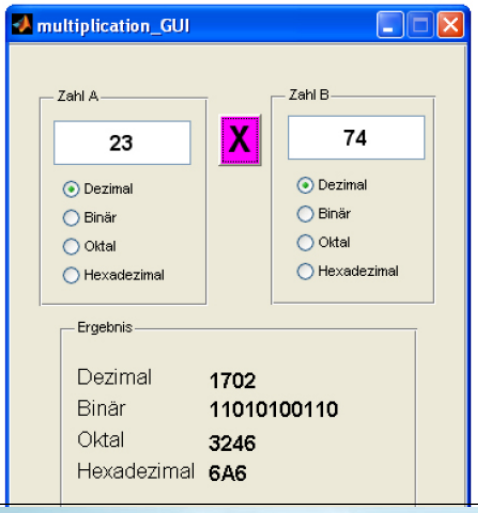


7 Graphical User Interfaces

7.1 Multiplikation GUI

7.1.1 Aufgabe

Schreiben Sie zur Funktion `multiplication()` aus Aufgabe 3.6 eine grafische Benutzeroberfläche `multiplication_GUI`. Das Ergebnis könnte ungefähr wie folgt aussehen, wobei durch drücken des lila farbenen *Pushbuttons* die Multiplikation der Zahlen *A* und *B* erfolgt:



7.1.2 Lösung

```
function varargout = multiplication_GUI(varargin)
% ------------------------------------------------------------------------
% Übungsaufgaben Matlab und Simulink
% Kapitel 7
% Aufgabe 1: Multiplikation GUI
% ------------------------------------------------------------------------
gui_Singleton = 1;
gui_State = struct('gui_Name', mfilename, ...
                   'gui_Singleton', gui_Singleton, ...
                   'gui_OpeningFcn', @multiplication_GUI_OpeningFcn, ...
                   'gui_OutputFcn', @multiplication_GUI_OutputFcn, ...
                   'gui_LayoutFcn', [] , ...
                   'gui_Callback', []);
if nargin && ischar(varargin{1})
       gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
       [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
       gui_mainfcn(gui_State, varargin{:});
end

% ------------------------------------------------------------------------
function multiplication_GUI_OpeningFcn(hObject, eventdata, handles, varargin)
       handles.output = hObject;
       guidata(hObject, handles);
% ------------------------------------------------------------------------
function varargout = multiplication_GUI_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;

% ------------------------------------------------------------------------
function edit_B_Callback(hObject, eventdata, handles)
function edit_B_CreateFcn(hObject, eventdata, handles)
       if ispc && isequal(get(hObject,'BackgroundColor'),...
                     get(0,'defaultUicontrolBackgroundColor'))
              set(hObject,'BackgroundColor','white');
       end
% ------------------------------------------------------------------------
```

```
function edit_A_Callback(hObject, eventdata, handles)
function edit_A_CreateFcn(hObject, eventdata, handles)
       if ispc && isequal(get(hObject,'BackgroundColor'),...
                     get(0,'defaultUicontrolBackgroundColor'))
              set(hObject,'BackgroundColor','white');
       end
% ########################################################################
% ### CALLBACKFUNKTION DES PUSBUTTONS ####################################
% ########################################################################
function push_mult_Callback(hObject, eventdata, handles)
       A = get(handles.edit_A,'String');              % Eingabefeld A auslesen
       B = get(handles.edit_B,'String');              % Eingabefeld B auslesen

       Adec = get(handles.rb_Adec,'Value');           % Zahlensystem A und B ermitteln
       Abin = get(handles.rb_Abin,'Value');
       Aokt = get(handles.rb_Aokt,'Value');
       Ahex = get(handles.rb_Ahex,'Value');

       Bdec = get(handles.rb_Bdec,'Value');
       Bbin = get(handles.rb_Bbin,'Value');
       Bokt = get(handles.rb_Bokt,'Value');
       Bhex = get(handles.rb_Bhex,'Value');

       A_choice = [Adec Abin Aokt Ahex];
       B_choice = [Bdec Bbin Bokt Bhex];

       for(k=1:1:4)
              if(A_choice(k) == 1)
                     switch k
                            case 1
                                   typA = 'dec';
                            case 2
                                   typA = 'bin';
                            case 3
                                   typA = 'okt';
                            case 4
                                   typA = 'hex';
                     end
              end
       if(B_choice(k) == 1)
              switch k
                     case 1
```

```
                    typB = 'dec';
                case 2
                    typB = 'bin';
                case 3
                    typB = 'okt';
                case 4
                    typB = 'hex';
            end
        end
end

[result] = multiplication(A,B,typA,typB);      % Funktion multiplikation
                                               % aus 3.6 aufrufen


set(handles.res_dec,'String',result.dec);      % Ergebnisfelder beschreiben
set(handles.res_bin,'String',result.bin);
set(handles.res_okt,'String',result.okt);
set(handles.res_hex,'String',result.hex);
```

7.1.3 Anmerkung

Die *Tags* der entsprechenden UI-Komponenten sind im zugehörigen Figure File über GUIDE natürlich entsprechend oben gewählter Namenskonvention zu bezeichnen.