Einführung in Matlab

Peter Junglas 12. 02. 2021

Inhaltsverzeichnis

Grundfunktionen

- Matlab:

  Umfangreiches Programmpaket für alle Arten numerischer Berechnungen

  Hersteller: The MathWorks Inc.

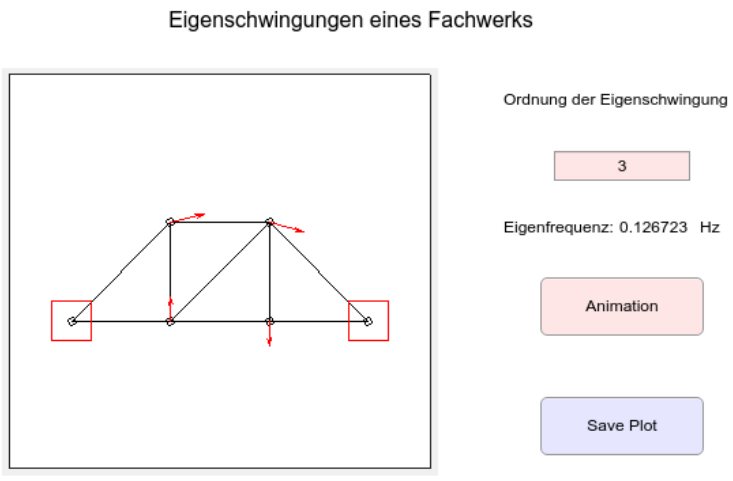
  viele Zusatzpakete für spezielle Anwendungen, auch von Drittfirmen

  bei uns (PC-Pool Diepholz) installiert

Paket	**Zweck**
Signal Processing Toolbox	Signalverarbeitung, Filter, ..
Image Processing Toolbox	Bildbearbeitung
Statistics and Machine Learning Toolbox	Statistische Beschreibung und Analyse von Daten
Simulink	graphisches Simulations-Tool
Simscape	Simulink-Blöcke für Physical Modeling
Stateflow	Simulink-Blöcke für Zustandsdiagramme
SimEvents	Simulink-Blöcke für Diskrete-Event-Simulation
Matlab/Simulink Coder	C-Programm erzeugen aus Simulink-Modellen und MATLAB-Code

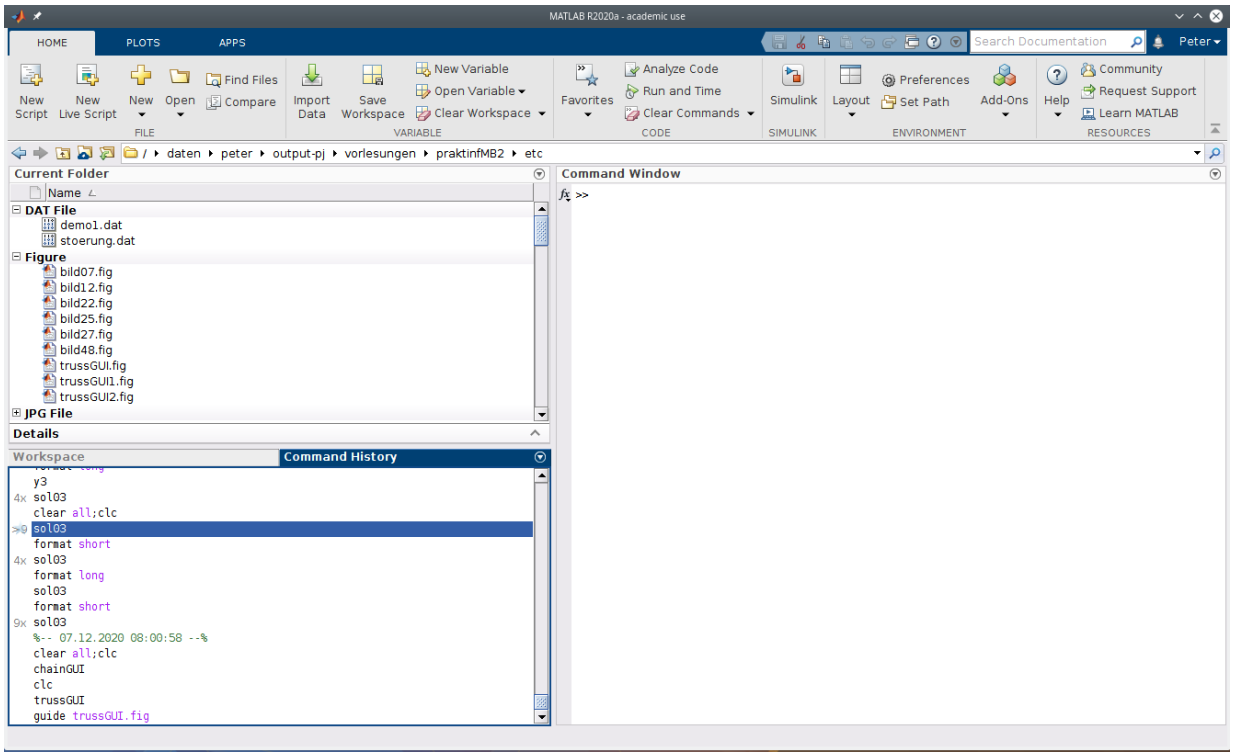
  Programmiersprache mit allen wichtigen Daten- und Programmstrukturen sowie Elementen zur objektorientierten Programmierung

  Matlab-Anwendungen
  - kleine Skripte
  - umfangreiche Programmpakete
  - Programme mit graphischer Bedieneroberfläche



- Bedienung:

  graphische Oberfläche mit mehreren Unter-Fenstern



Kommandofenster zur Eingabe von Matlab-Befehlen am Prompt >>

Fenster mit den aktuellen Variablen (**Workspace**)

Fenster mit den Dateien im aktuellen Verzeichnis (**Current Directory**)

Fenster mit den letzten Befehlen (**Command History**)

- aktivieren mit `Layout/Command History/Docked`

Fenster können beliebig angeordnet werden

- Hilfe:

  Informationen zu einer Funktion

  ```
  help NAME
  ```

  viele komplette Beispiele

  ```
  demo
  ```

  umfangreiche Komplett-Dokumentation

  ```
  doc
  ```

- Matlab als Taschenrechner:

  Zahlen und Operationen direkt eingeben

  ```
  >> 12.567*31.34/(17 + 4) - 1.88e1

  ans =

         -0.0452
  ```

  die üblichen Operatoren `+ - * / ^` ("hoch")

  alle gängigen Funktionen und viele mehr, z.B.

  - `sqrt`, `exp`, `log`, `log10`
  - `sin`, `cos`, `tan`, `asin`, `acos`, `atan`
  - `sinh`, `cosh`, `atanh`, ...

- `rem`, `floor`, `ceil`
- `pi`

Wiederholung mit "Pfeiltaste rauf"

- Genauigkeit:

  Anzeige
  - Standard: 4 Nachkommastellen bzw. e-Notation
  - alle Dezimalen mit

    ```
    format long
    ```

  normalerweise mit **double**-Zahlen
  - "fast" 16 Dezimalen
  - Exponent bis ±308

  daher rechnertypische Fehler

  ```
  >> 1 - 5*0.2

  ans =

       0

  >> 1 - 0.2 - 0.2 - 0.2 - 0.2 - 0.2

  ans =

       5.551115123125783e-17
  ```

- Variablen:

  "Speicher" mit Namen

  ```
  >> g = 9.81

  g =

      9.8100
  ```

  Namen aus Buchstaben, Zahlen und Unterstrich _

  Klein- und Großbuchstaben verschieden

  Unterdrücken der Ausgabe mit abschließendem ;

  ```
  >> T = 0.3;
  >> omega = 2*pi/T;
  >> t = 2.75;
  >> A = 1.3;
  >> x = A*sin(omega*t)

  x =

      1.1258
  ```

  Variable `ans` für unbenannte Ergebnisse

  Liste aller Variablen (Alternative zum Workspace-Fenster)

```
>> whos
  Name         Size          Bytes  Class

  T            1x1               8  double
  A            1x1               8  double
  ans          1x1               8  double
  b            1x1               8  double
  g            1x1               8  double
  omega        1x1               8  double
  t            1x1               8  double
  x            1x1               8  double
```

Löschen aller Werte (und Freigabe des Speichers) und Löschen des Kommandofensters mit

```
clear all; clc
```

- Zuweisungsoperator:

  Beispiel

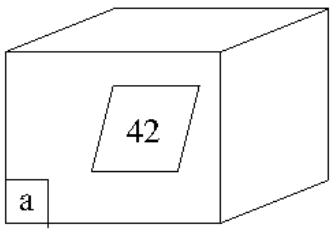
  ```
  a = 3;
  b = a;
  a = 5;
  b
  ```

  - was kommt heraus: 3 oder 5?

  Bedeutung von `a = b`
  - Gleichheitszeichen = bedeutet nicht "gleich"!
  - = heißt **Zuweisungsoperator**
  - "bestimme Wert von b und speichere ihn in a"

  Bedeutung von `a = b` in Matlab und Mathematik verschieden
  - in Matlab: Anweisung ("tue etwas")
  - in Mathematik: Aussage ("die sind gleich")

  unterscheide "Name von Variable" und "Wert von Variable"

  - damit Verhalten im Beispiel völlig klar

Vektoren und Plots

- Grundoperationen mit Vektoren:

  Folge von Zahlen, eingegeben als

```
>> a = [1, 2, 3, 4, 5, 6]

a =

     1     2     3     4     5     6
```

  einfache Addition, Subtraktion, Verknüpfung mit Zahlen

```
>> b = 2*a - 1

b =

     1     3     5     7     9    11
```

  viele Standardfunktionen arbeiten elementweise mit Vektoren

```
>> c = log(b)

c =

         0    1.0986    1.6094    1.9459    2.1972    2.3979
```

  elementweise Multiplikation, Division und Potenz mit `.*`, `./`, `.^`

```
>> a.^3

ans =

     1     8    27    64   125   216
```

- Vektor-Operationen:

  viele spezielle Funktionen, z.B.

`sum`	Summe aller Werte
`max`	größter Wert
`min`	kleinster Wert
`length`	Anzahl der Elemente
`mean`	Mittelwert
`std`	Standard-Abweichung
`sort`	sortierter Vektor

  Umwandlung von Zeilenvektor in Spaltenvektor (**Transposition**)

```
>> a'

ans =

     1
     2
     3
     4
     5
     6
```

Skalarprodukt = (Matrix-)Produkt von Zeilenvektor mit Spaltenvektor

```
>> b*a'

ans =

   161
```

- Teile von Vektoren auswählen:

  einzelnes Element mit Klammern

  ```
  >> a(3)

  ans =

       3
  ```

  Elemente `i` bis `j` mit `i:j`

  ```
  >> a(3:5)

  ans =

       3     4     5
  ```

  bis zum Schluss mit `end`

  ```
  >> a(4:end)

  ans =

       4     5     6
  ```

  nur jedes k-te Element mit `i:k:j`

  ```
  >>  a(1:2:6)

  ans =

       1     3     5
  ```

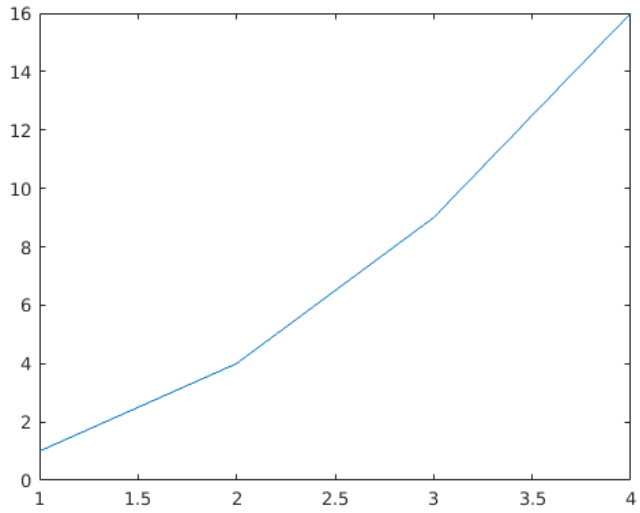
  `i:k:j` ist selbst ein Vektor (sogar mit Dezimalzahlen)

  ```
  >> 1:0.4:3

  ans =

      1.0000    1.4000    1.8000    2.2000    2.6000    3.0000
  ```

- Plotten von Vektoren:

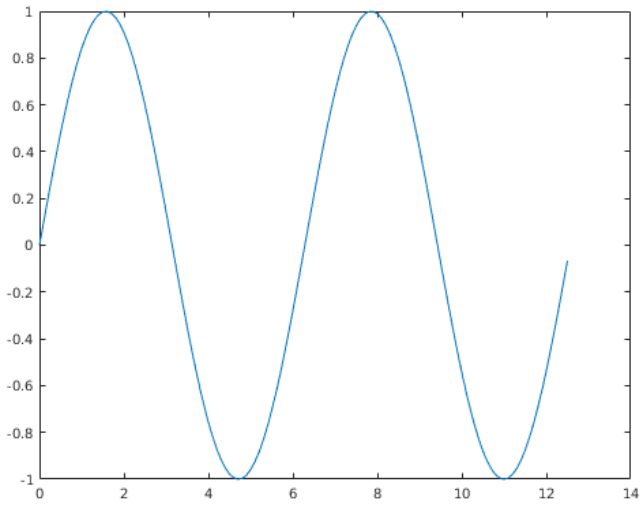
  `plot` verbindet Punkte, gegeben durch Vektoren ihrer x- und y-Komponenten

  ```
  plot([1, 2, 3, 4], [1, 4, 9, 16])
  ```



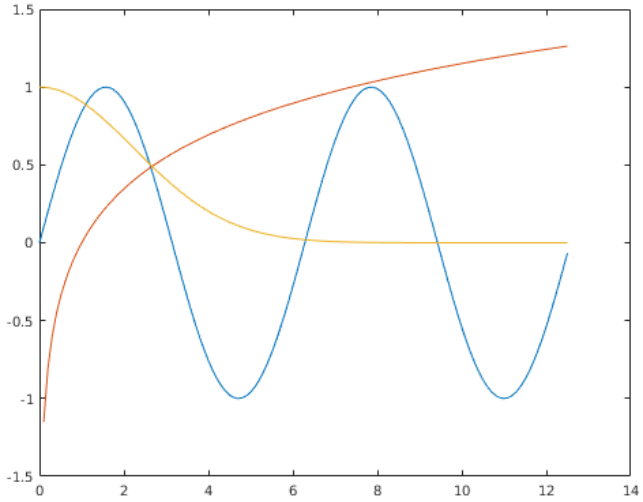
viele x-Werte → Plot einer Funktion

```
t = 0:0.1:4*pi;
y = sin(t);
plot(t,y)
```



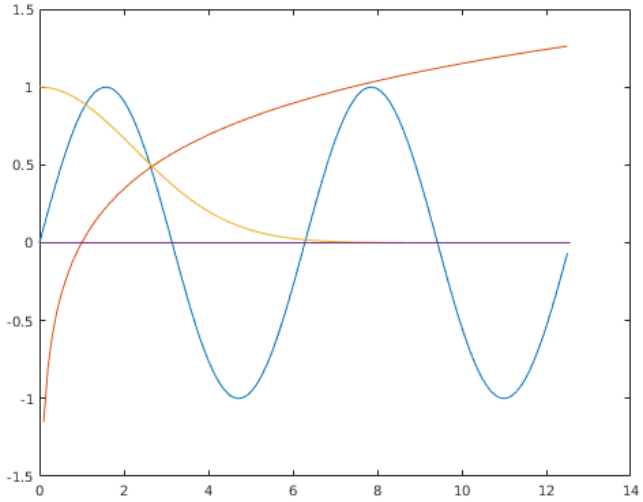
auch mehrere Vektoren möglich, jeweils als x- und y-Koordinaten

```
y2 = 0.5*log(t);
y3 = exp(-0.1*t.*t);
plot(t, y, t, y2, t, y3)
```



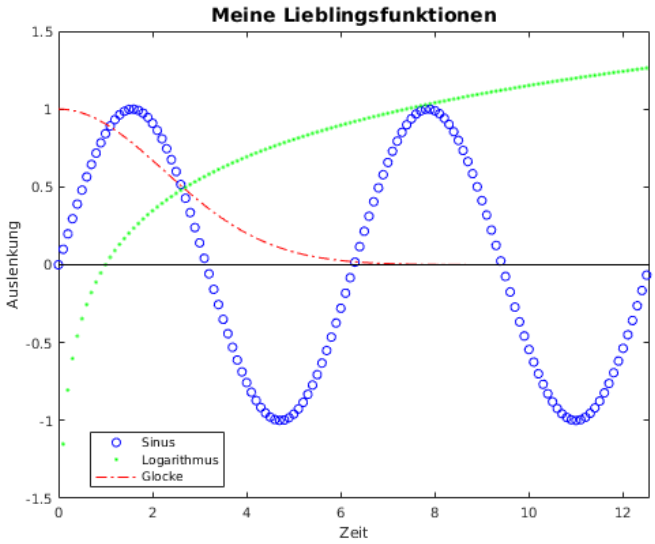
x-Achse als Datensatz mit zwei Punkten

```
plot(t, y, t, y2, t, y3, [0, 4*pi], [0, 0]);
```



zahllose Optionen und weitere Funktionen, s. `help plot`

```
plot(t, y, "bo", t, y2, "g.", t, y3, "r-.", [0, 4*pi], [0, 0], "k-")
title("Meine Lieblingsfunktionen", ...
      "FontSize", 14, "FontWeight", "bold")
xlabel("Zeit");
ylabel("Auslenkung");
xlim([0, 4*pi])
legend("Sinus", "Logarithmus", "Glocke", "Location", "best");
```



- Arbeiten mit Skripten

  Editor öffnen mit `File/New/M-File`

  Kommandos genauso eingeben wie im Command Window

  abspeichern als `NAME.m`, wobei `NAME` nur Buchstaben, Ziffern oder Unterstrich _ enthält und mit Buchstaben beginnt

  Beispiel zeichneFunktion.m

  ausführen mit Taste `<F5>` oder durch Eingabe von `NAME` im Command Window

- Aufgaben:

  Aufgabe 1

  Aufgabe 2

Erstellen eigener Funktionen

- Funktion:

  Teilprogramm, das als "Black Box" abläuft

  hat Eingangsgrößen (**Parameter**) und Ergebnisse

  Beispiel `y = sin(t)`
  - Parameter `t`
  - Ergebnis `y`
  - berechnet den Sinus des Parameters `t` (im Bogenmaß)

  Beispiel `h = plot(X, Y)`
  - Parameter `X`, `Y` (Vektoren mit x- bzw. y-Koordinaten)
  - Ergebniswert `h` (Zeiger auf die Graphik für nachfolgende Modifikationen)
  - zeichnet einen Streckenzug durch die Punkte (`X(i)`, `Y(i)`)

  Parameter
  - Anzahl und Typ normalerweise festgelegt
  - in Matlab auch Funktionen mit variabler Parameterzahl (z.B. `plot`)

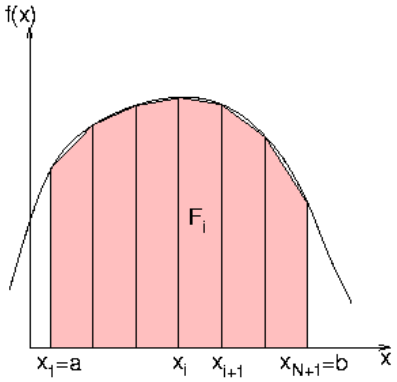
  Ergebniswert kann ignoriert werden
  - `plot(X,Y)`

  auch mehrere Ergebnise möglich
  - `[theta,rho] = cart2pol(x,y)`

- Berechnung bestimmter Integrale mit Trapezregel:

  sehr einfaches Verfahren zur numerischen Integration

  Prinzip
  - Unterteilung des Integrationsintervalls in N Teile
  - Funktion in jedem Intervall durch Strecke nähern
  - Fläche der sich ergebenden Trapeze addieren



  in Formeln
  - Zahl der Intervalle N vorgeben
  - Breite h der Intervalle

    h = (b - a)/N

  - Fläche F_i eines Trapezes

$$F_i = h\frac{(f(x_i) + f(x_{i+1}))}{2}$$

  - Integral damit

$$\int_a^b f(x)\,dx \approx \sum_{i=1}^{N} F_i$$
$$= h\left[\frac{1}{2}f(x_1) + \sum_{i=2}^{N} f(x_i) + \frac{1}{2}f(x_{N+1})\right]$$

Beispiel i.f.

$$I = \int_0^{\pi/2} \sin(x)\,dx$$

Berechnung von I in Matlab

```
>> a = 0;
>> b = pi/2;
>> N = 5;
>> h = (b-a)/N;
>> x = a:h:b;
>> y = sin(x);
>> I = h*(0.5*y(1) + sum(y(2:N)) + 0.5*y(N+1))

I =

    0.9918
```

ganz genauso für N = 100 ergibt (mit `format long`)

```
I = 0.99997943823961
```

- Eigene Funktionen in Matlab:

  als Skript (keine Funktion)
  - alle Kommandos in eine Datei trapez1.m
  - mit `trapez1` (Dateiname) aufrufen
  - zur Erklärung Kommentare einfügen (von `%` bis zum Zeilenende)
  - alle Variablen landen im (Basis-)Workspace
  - Problem: alle Parameter direkt in Datei ändern

  als Funktion
  - alle Kommandos in Datei trapez2.m
  - hat eigenen Workspace für ihre internen Variablen (`h`, `x`, `y`)
  - erste Zeile definiert Funktionsnamen und Argumente

  ```
  function trapez2(a, b, N)
  ```

  benutzen mit

  ```
  >> trapez2(0, pi/2, 1000)

  ans =

     0.99999979438323
  ```

  Kommentar am Anfang wird Hilfetext

  ```
  >> help trapez2

    Integration mit der Trapezregel
      Argumente
          Grenzen a, b
          Zahl der Intervalle N
      Ergebnis
  ```

```
Integral des Sinus von a bis b,
genaehert mit Trapezregel mit N Intervallen
```

andere Funktion als Sinus

- Funktion als neues Argument `func` (s. trapez.m)
- ersetze `sin(x)` durch `func(x)`
- aufrufen mit Funktionszeiger ("function handle"): `@FUNKTIONSNAME`

```
>> trapez(@cos, 0, pi, 100)

ans =

     8.370884395220717e-17
```

beliebige Funktion integrieren

- gewünschte Funktion als eigene Matlab-Funktion
- z.B. Fehlerintegral e^{-x^2} (glocke.m)

```
>> glocke(1)

ans =

   0.36787944117144
```

damit in `trapez`

```
>> trapez(@glocke, 0, 10, 1000)

ans =

   0.88622692545276
```

- Bessere Integrationsmethoden:

  Funktion durch Parabeln statt Geraden annähern (**Simpson-Methode**)

  Intervalle nicht gleich groß, sondern geschickt anpassen

  - unzählige Verfahren
  - kann auch mit schwierigen Integranden fertigwerden
  - nicht möglich bei Tabellenfunktionen

  Matlabfunktion `integral` berechnet Integral auf relative Genauigkeit `1e-6`

```
integral(@glocke, 0, 10)
```

- Aufgaben:

  Aufgabe 3

  Aufgabe 4

Matrizen

- Grundfunktionen:

  rechteckiges Zahlenschema, eingeben mit

```
>> a = [1 2 3; 4 5 6; 7 8 9; 10 11 12]

a =

     1     2     3
     4     5     6
     7     8     9
    10    11    12
```

  elementweise Funktionen wie bei Vektoren

  erster Index ist Zeilenindex

```
>> a(2,3)

ans =

     6
```

  Auswahlfunktionen wie bei Vektoren

```
>> a(3:4, 2:3)

ans =

     8     9
    11    12
```

  Auswahl ganzer Zeilen oder Spalten mit :

```
>> a(3,:)

ans =

     7     8     9

>> a(:,2)

ans =

     2
     5
     8
    11
```

  Multiplikation von Matrizen untereinander und mit Vektoren

```
>> a = [1 2 3; 4 5 6; 7 8 9];
>> b = [0 1 2; -1 0 1; -2 -1 0];
>> v = [2 0 -1]'

v =

     2
     0
```

```
     -1

>> a*v

ans =

     -1
      2
      5

>> a*b

ans =

     -8     -2      4
    -17     -2     13
    -26     -2     22
```

Vektorfunktionen auf Matrizen liefern Vektoren

```
>> sum(a)

ans =

     12    15    18
```

- Einige Matrixfunktionen:

  Transposition

```
>> a'

ans =

      1      4      7
      2      5      8
      3      6      9
```

  Matrizen aus Nullen oder aus Einsen

```
>> zeros(2,2)

ans =

      0      0
      0      0

>> ones(2,2)

ans =

      1      1
      1      1
```

  Matrizen aus Zufallszahlen zwischen 0 und 1

```
>> rand(3,3)

ans =

    0.7680    0.7889    0.2140
```

```
0.9708    0.4387    0.6435
0.9901    0.4983    0.3200
```

- Ein- und Ausgabe:

  alle aktuellen Variablen abspeichern (in Datei `matlab.mat`)

  ```
  save
  ```

  wieder einladen mit

  ```
  load
  ```

  abspeichern der Matrix `a` in ASCII-Format in Datei `juhu.dat`

  ```
  save("juhu.dat", "a", "-ascii")
  ```

  einladen

  ```
  load("juhu.dat")
  ```

  liefert Ergebnis in Matrix `juhu`

  ASCII-Datei für `load` darf nur eine Matrix enthalten, sonst andere Dateifunktionen verwenden

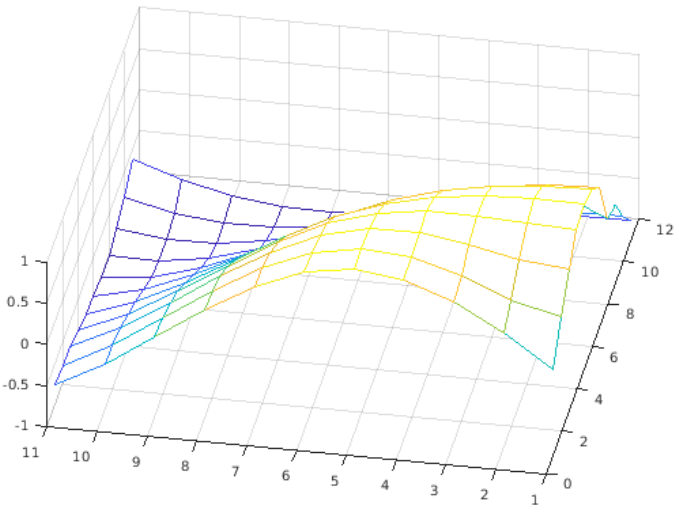
- Darstellung von Matrizen:

  Demo-Matrix aus demo1.dat laden

  ```
  demo1 = load("demo1.dat")
  ```

  Darstellung als 3d-Gitter

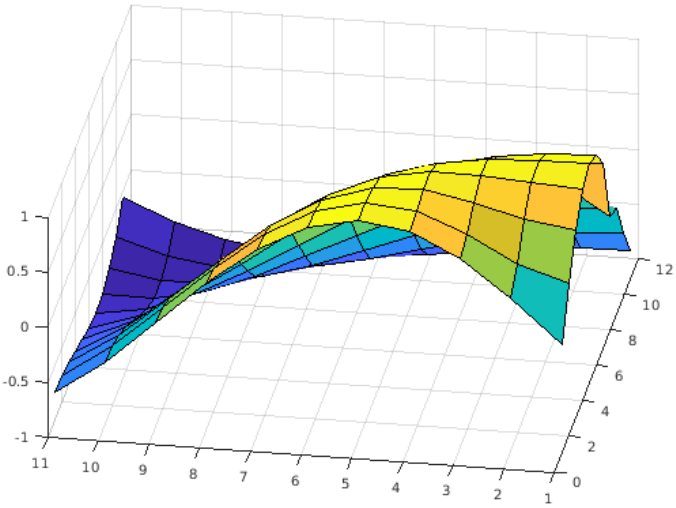
  ```
  mesh(demo1)
  ```



  Änderung der Ansicht durch Lupen- und Drehfunktionen

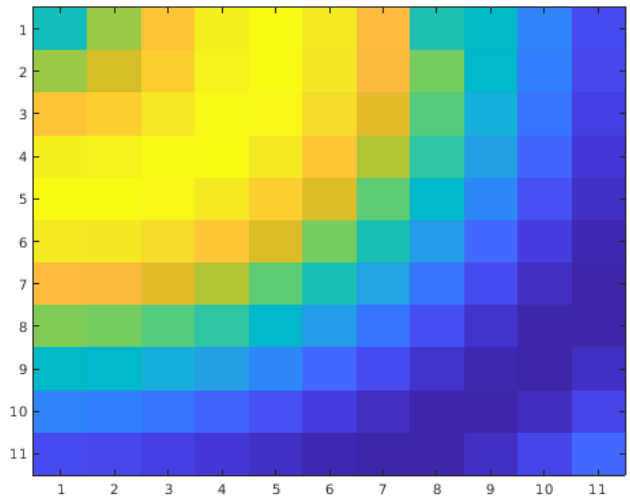
  Darstellung als Oberfläche

  ```
  surf(demo1)
  ```



Darstellung als Bild

```
imagesc(demo1)
```



Farben gemäß aktueller Farbtabelle

- Darstellung zweidimensionaler Funktionen:

  Beispielfunktion

  $$f(x,y) = \sin x^2 + \cos y$$

  soll graphisch dargestellt werden

  Funktion in Matlabdatei demo2dfunc.m gegeben

  Bereiche für x und y in Vektoren vorgegeben

  ```
  x = -3:0.1:3;
  y = -5:0.1:5;
  ```

  benötigt werden nun Werte als Matrix

  $$Z = \begin{pmatrix} f(x_1,y_1) & f(x_2,y_1) & \dots & f(x_n,y_1) \\ f(x_1,y_2) & f(x_2,y_2) & \dots & f(x_n,y_2) \\ \vdots & \vdots & \vdots & \vdots \\ f(x_1,y_m) & f(x_2,y_m) & \dots & f(x_n,y_m) \end{pmatrix}$$

  daher zunächst Hilfsmatrizen X, Y bestimmen

  ```
  [X,Y] = meshgrid(x,y);
  ```

- erzeugt Matrizen

$$X = \begin{pmatrix} x_1 & x_2 & \dots & x_n \\ x_1 & x_2 & \dots & x_n \\ \vdots & \vdots & \vdots & \vdots \\ x_1 & x_2 & \dots & x_n \end{pmatrix} \quad Y = \begin{pmatrix} y_1 & y_1 & \dots & y_1 \\ y_2 & y_2 & \dots & y_2 \\ \vdots & \vdots & \vdots & \vdots \\ y_m & y_m & \dots & y_m \end{pmatrix}$$

Funktionswerte als Matrix jetzt einfach mit

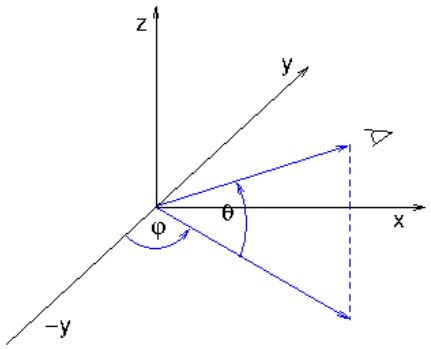
```
Z = demo2dfunc(X, Y);
```

3d-Darstellung z. B. mit

```
mesh(X, Y, Z)
```

Anpassen des Blickpunkts (**Viewpoint**)

- interaktiv oder mit `view(phi, theta)`
- `phi`: Winkel in Grad in der x-y-Ebene, zur negativen y-Achse
- `theta`: Winkel in Grad zur z-Achse, bezogen auf die xy-Ebene



- Standardwerte: phi = -37.5°, theta = 30°
- hier: `view(-10, 40)`

Achsenbeschriftung

```
xlabel("x","FontSize",14);
ylabel("y","FontSize",14);
zlabel("z","FontSize",14);
```

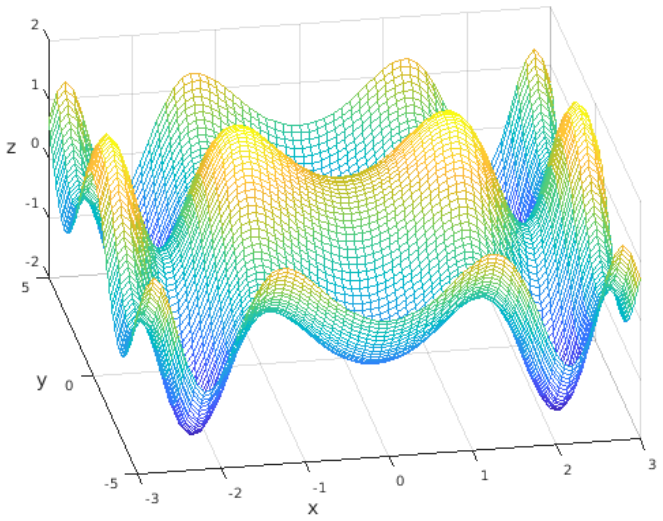
Drehen der Beschriftung der z-Achse

```
hz = get(gca(),"ZLabel");
set(hz,"Rotation", 0.0);
```

Verschiebung der Beschriftung der y-Achse

```
hy = get(gca(),"YLabel");
get(hy,"Position") % liefert aktuellen Wert -10.8636 -68.9438 20.0268
set(hy,"Position", [-10.3 -64 20]);
```

Ergebnis



- Aufgaben:
  - Aufgabe 5
  - Aufgabe 6

Lösen von Gleichungen

- Lineare Gleichungssysteme:

  betrachten Beispiel

$$\begin{aligned} 2x - y + 3z &= 1 \\ 4x - 2y + z &= 5 \\ 3y + z &= -3 \end{aligned}$$

  in Matrixform

  A x = r

  wobei

$$A = \begin{pmatrix} 2 & -1 & 3 \\ 4 & -2 & 1 \\ 0 & 3 & 1 \end{pmatrix} \qquad r = \begin{pmatrix} 1 \\ 5 \\ -3 \end{pmatrix}$$

- Lösung in Matlab:

  für das Beispielsystem

```
>> A = [2 -1 3; 4 -2 1; 0 3 1]

A =

     2    -1     3
     4    -2     1
     0     3     1

>> r = [1 5 -3]'

r =

     1
     5
    -3

>> x = A \ r

x =

    1.0000
   -0.8000
   -0.6000
```

  Probe:

```
>> A*x - r

ans =

   1.0e-15 *

   -0.3331
         0
         0
```

  und genauso auch z.B. für ein 1000x1000-System

```
A=rand(1000);
r = rand(1000,1);
x = A \ r;
```

Lösung braucht etwa 0.05 s auf 2.6 Gz-PC

Probe

```
>> error = max(abs(A*x-r))

error =

   2.0761e-13
```

- Überbestimmte Systeme:

  mehr Gleichungen als Unbekannte

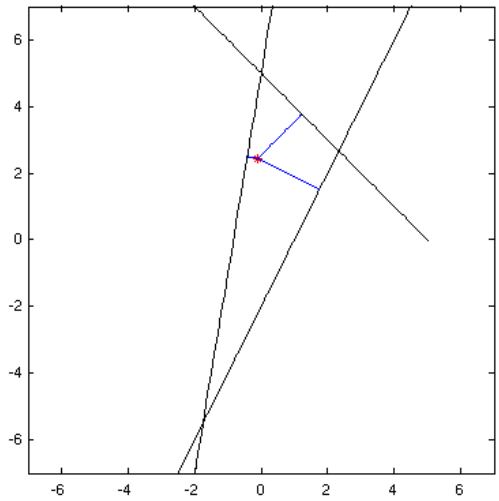
  i.a. keine Lösung

  gesucht: Lösung mit geringstem Fehler

  in Matlab auch einfach mit

  ```
  x = A \ r;
  ```

  graphisch bei 2 Unbekannten
  - Gleichung ax + by = r ist Gerade
  - mehr als 2 Geraden schneiden sich (i.a.) nicht in einem Punkt
  - gesuchter Punkt hat geringste Summe der Abstände



  Beispiel
  - 3 Geradengleichungen

$$\begin{aligned} y &= 6x + 5 \\ y &= -x + 5 \\ y &= 2x - 2 \end{aligned}$$

  - geschrieben als lineares System

$$\begin{aligned} 6x - y &= -5 \\ -x - y &= -5 \\ 2x - y &= 2 \end{aligned}$$

  - also in Matlab

```
A = [6 -1; -1 -1 ; 2 -1];
r = [-5 -5 2]';
x0 = A \ r
```

- Ergebnis: `x0 = [-0.0946 2.4459]'`

- Nullstellen von Polynomen:

  Beispiel 2. Grades

  $2x^2 - 2x - 24 = 0$

  in Matlab Polynom als Vektor der Koeffizienten

  ```
  poly1 = [2 -2 -24];
  ```

  alle Lösungen mit der Funktion `roots`

  ```
  >> roots(poly1)

  ans =

       4
      -3
  ```

  geht auch bei komplexen Lösungen, etwa $x^2 + 1 = 0$

  ```
  >> poly2 = [1 0 1];
  >> roots(poly2)

  ans =

          0 + 1.0000i
          0 - 1.0000i
  ```

  klappt auch bei höherer Ordnung

  ```
  >> poly3 = [1    -8    20   -10   -25    22];
  >> roots(poly3)

  ans =

     3.7059
    -1.0781
     2.1861 + 0.8527i
     2.1861 - 0.8527i
     1.0000
  ```

- Lösung "beliebiger" Gleichungen:

  Beispiel: Löse die Gleichung

  $$e^{-0.05x^2} = \sin(x)$$

  Umformulierung: Finde Nullstellen der Funktion

  $$f(x) = \sin(x) - e^{-0.05x^2}$$

  Funktion `fzero` braucht als Argumente Funktion und Startwert

  Funktion direkt hinschreiben als "anonyme Funktion"

  ```
  >> F = @(x) sin(x) - exp(-0.05*x.^2)
  ```

```
F =

    @(x)sin(x)-exp(-0.05*x.^2)
```

irgendeinen (?) Startwert nehmen und ab geht's

```
>> x0 = 0;
>> x = fzero(F, x0)

x =

    1.1968
```

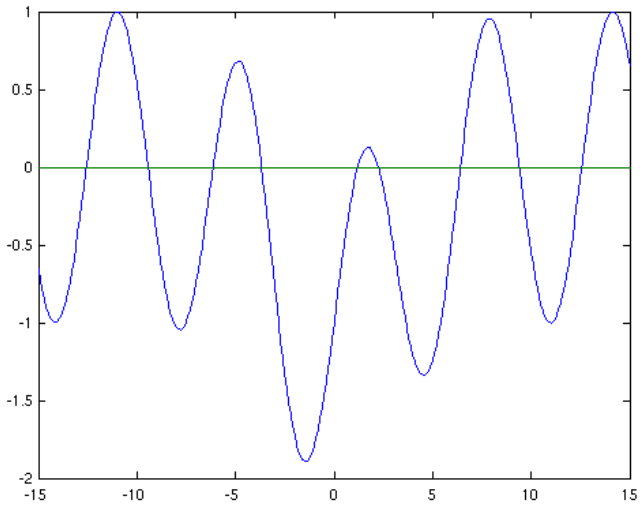
andere Startwerte liefern andere Lösungen

x0	x
0	1.1968
1	1.1968
2	2.2537
3	2.2537
4	2.2537
5	6.4116

wieviele Lösungen gibt es überhaupt?

graphisch Überblick gewinnen

```
x = -15:0.1:15;
plot(x, F(x), [-15 15], [0 0])
```



damit Struktur der Lösungsmenge und gute Startwerte für `fzero`

am besten mit Suchintervall [x0,x1]

```
x = fzero(F, [x0,x1])
```

- Voraussetzungen: f ist stetig und Vorzeichen von `F(x0)` und `F(x1)` verschieden
- genau eine Lösung im Intervall → `xzero` findet sie
- mehrere Lösungen im Intervall → `xzero` findet eine

- Aufgaben:

  Aufgabe 7

Integration von Differentialgleichungen

- Beispiele für Differentialgleichungen:

  radioaktiver Zerfall
  - Anzahl N der Kerne mit Zerfallskonstante λ

  $$\frac{dN}{dt} = -\lambda N$$

  radioaktive Zerfallskette
  - Teilchensorte 1 zerfällt in Sorte 2 mit λ_1
  - Sorte 2 zerfällt weiter mit λ_2

  $$\begin{aligned} \frac{dN_1}{dt} &= -\lambda_1 N_1 \\ \frac{dN_2}{dt} &= \lambda_1 N_1 - \lambda_2 N_2 \end{aligned}$$

  angeregte Schwingung mit viskoser Reibung
  - Masse m schwingt an Feder mit Federkonstante c
  - Reibung ist proportional zur Geschwindigkeit v
  - äußere zeitabhängige Erregerkraft F(t)

  $$m\ddot{x} + b\dot{x} + cx = A\cos(\omega t)$$

- Lösungsverfahren von Euler:

  einfachstes (und ungenauestes) Verfahren

  Grundidee
  - Ableitung ersetzen durch (endliche) Differenzen

  $$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

  - ergibt normale Gleichungen für f

  im Beispiel "radioaktiver Zerfall"

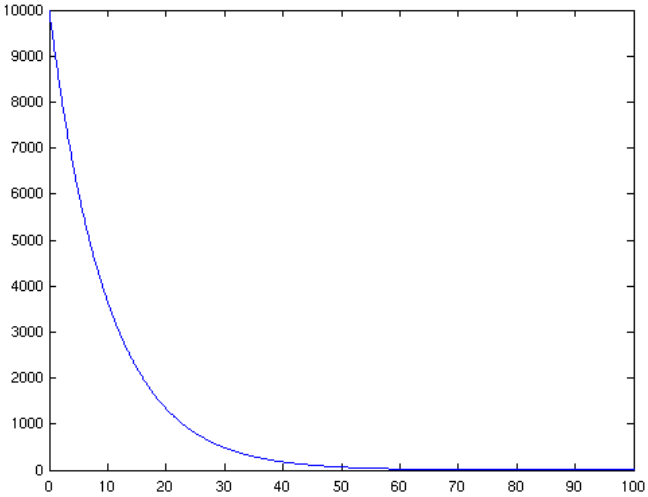
  $$\begin{aligned} \frac{N(t+h) - N(t)}{h} &= -\lambda N(t) \\ \Rightarrow \quad N(t+h) &= (1 - \lambda h) N(t) \end{aligned}$$

  - bei gegebenem Startwert N(0) schrittweise lösen
  - hier sogar direkt auflösbar

  $$N(nh) = (1 - \lambda h)^n N(0)$$

  in Matlab

```
N0 = 10000;          % Anfangsmenge
lambda = 0.1;        % Zerfallskonstante
h = 0.1;             % Schrittweite
n = 0:1000;          % 1000 Schritte
t = h*n;             % Zeitwerte
N = N0*(1-lambda*h).^n;   % Werte fuer N(t)
plot(t, N);
```



- Lösungsverfahren in Matlab:

  Ausgangspunkt immer Grundform

  y' = **f**(t, **y**)

  viele sehr weitentwickelte Verfahren (**ode solver**)

  einige Solver für schwierige (**steife**) Differentialgleichungen

  Auswahl des Solvers u.a. abhängig von
  - Genauigkeitsforderung
  - "Glattheit" der Gleichungen
  - Schwierigkeit der Gleichung (steif oder nicht steif)

  Daumenregel
  - erste Wahl: `ode45`
  - falls der versagt oder zu lange braucht: `ode15s`
  - wenn der auch nicht klappt: Experten fragen

- Vorgehensweise am Beispiel "Radioaktiver Zerfall"

  Gleichungen schreiben als y' = f(t, y)

  $$y' = -\lambda y$$

  rechte Seite als Matlab-Funktion definieren (radioaktiv.m)

  Solver aufrufen

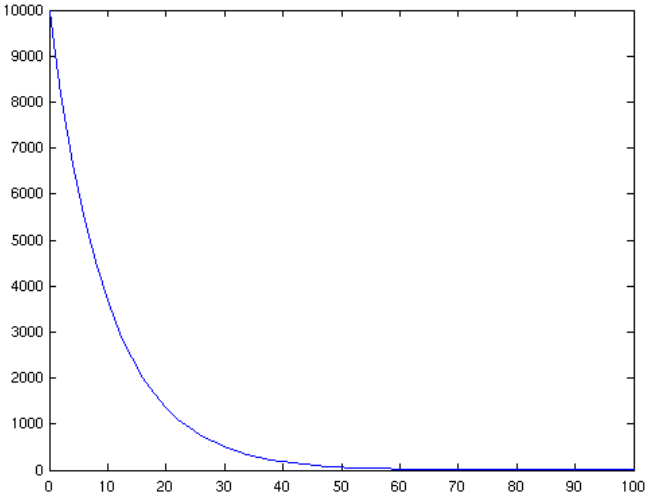
  ```
  [t,y] = ode45(@radioaktiv,[0 100],[10000]);
  ```

  Parameter
  - betrachtetes Zeitintervall `[0 100]`
  - Anfangswert `[10000]`

  Ergebniswerte `y` und zugehörige Zeiten `t`

  Ergebnis in gewohnter Form plotten

  ```
  plot(t, y);
  ```



- Beispiel "Zerfallskette":

  System von Differentialgleichungen

  wie vorher, aber alle Größen als Vektoren

  Gleichungen als y' = f(t, y)

$$\begin{aligned} y_1' &= -\lambda_1 y_1 \\ y_2' &= \lambda_1 y_1 - \lambda_2 y_2 \end{aligned}$$

  rechte Seiten als Matlab-Funktion radiokette.m

  Anfangsbedingungen z.B.

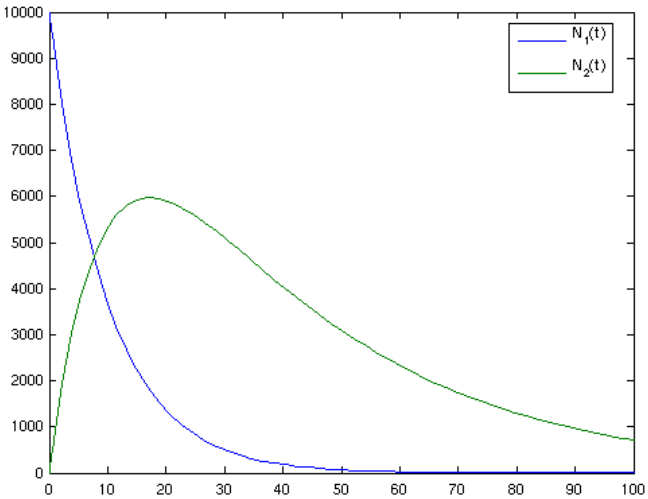
  - 10000 Teilchen von Sorte 1
  - keine Teilchen von Sorte 2

  Lösen

```
[t,y] = ode45(@radiokette,[0 100],[10000 0]);
```

  Spalten 1, 2 von `y` enthalten Zeitwerte von y_1, y_2, daher plotten mit

```
plot(t, y(:,1), t, y(:,2))
legend('N_1(t)', 'N_2(t)', 0);
```



- Beispiel "Angeregte Schwingung mit viskoser Reibung"

  Problem: Gleichung zweiter Ordnung

Trick: Geschwindigkeit $\dot{x}$ als zweite Variable einführen

damit Umschreiben in 2 Gleichungen 1. Ordnung

$$\begin{aligned} \mathbf{y} &= \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} := \begin{pmatrix} x \\ \dot{x} \end{pmatrix} \\ \Rightarrow \quad \mathbf{y}' &= \begin{pmatrix} y_1' \\ y_2' \end{pmatrix} = \begin{pmatrix} \dot{x} \\ \ddot{x} \end{pmatrix} = \begin{pmatrix} \dot{x} \\ -\frac{b}{m}\dot{x} - \frac{c}{m}x + \frac{A}{m}\cos(\omega t) \end{pmatrix} \\ &= \begin{pmatrix} y_2 \\ -\frac{b}{m}y_2 - \frac{c}{m}y_1 + \frac{A}{m}\cos(\omega t) \end{pmatrix} \end{aligned}$$

rechte Seiten als Matlab-Funktion erzwungen.m

Anfangsbedingungen z.B. Masse in Ruhe am Ursprung

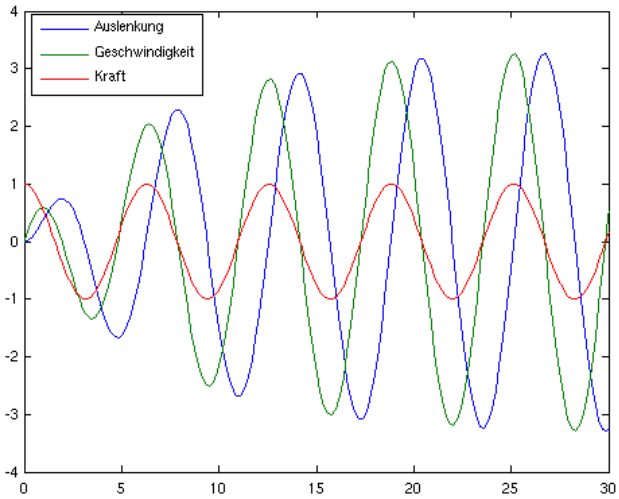
Lösen

```
[t, y] = ode45(@erzwungen, [0 30], [0 0]);
```

Darstellung

- zweite Spalte ist Geschwindigkeit
- zusätzlich äußere Anregung plotten

```
F = cos(t);
plot(t, y(:,1), t, y(:,2), t, F)
legend('Auslenkung', 'Geschwindigkeit', 'Kraft', 'Location', 'NorthWest');
```



- Übergabe von Parametern:

  Problem:

  - Parameter (m, b, c, A, omega) fest in Funktion `erzwungen`
  - sollen direkt angebbar sein

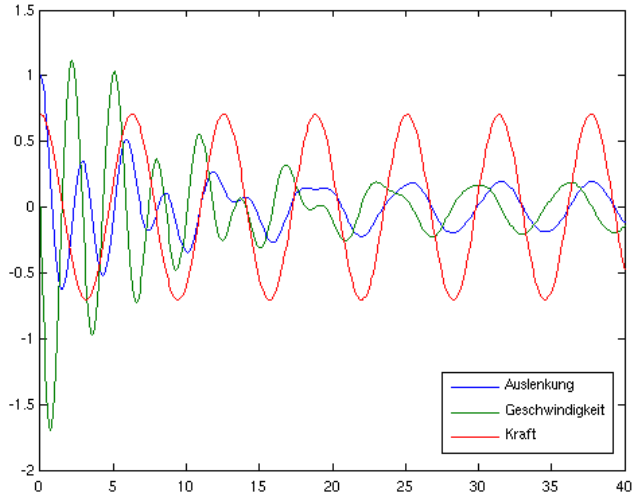
  Parameter als zusätzliche Argumente der Funktion (erzwungenp.m)

  Erzeugen einer Hilfsfunktion `myfunc`, die nur von t und y abhängt

  ```
  myfunc = @(t, y) erzwungenp(t, y, 1 , 0.286, 4.68, 0.706, 1);
  ```

  Aufruf des Solvers mit der Hilfsfunktion

  ```
  [t, y] = ode45(myfunc, [0 40], [1 0]);
  ```



- Aufgaben:

  Aufgabe 8

Anwendungs-Beispiele

- Wärmezufuhr bei temperaturabhängiger Wärmekapazität
- Versuchsauswertung eines Zugversuchs

Wärmezufuhr bei temperaturabhängiger Wärmekapazität

- Aufgabenstellung (vgl. Thermodynamik 1, Aufgabe 11):

  Ein Kohlendioxid-Strom von $\dot{m} = 5.668 \cdot 10^{-3}$ kg/s wird reversibel von der Anfangstemperatur t_1 = 70.0 °C mit einer Heizleistung von $\dot{Q}_{12}$ = 3.225 kW isobar erhitzt. Welche Temperatur wird dabei erreicht?

  Daten der Wärmekapazität in cps.dat

- Lösungsverfahren:

  grundlegende Gleichung aus der Thermodynamik

  $$\dot{Q}_{12} = \dot{m}\,\overline{c_p}|_{t_1}^{t_2}\,(t_2 - t_1)$$

  Umformen in Bestimmung einer Nullstelle

  $$f(t) := \dot{Q}_{12} - \dot{m}\,\overline{c_p}|_{t_1}^{t}\,(t - t_1) = 0$$

  Lösen mit `fzero` und geeignetem Intervall

- Berechnung der spezifischen Wärmekapazität:

  Tabelle enthält molare mittlere Wärmekapazitäten $\overline{C_{m,p}}|_{0°C}^{t}$

  Daten einlesen

  ```
  data = load("cps.dat");
  tD = data(:,1);          % in °C
  cpD = data(:,3)/M;       % in J/(kg K)
  ```

  Daten werden dabei mit der Molmasse M gleich von molar in spezifisch umgerechnet

  zu gegebener Temperatur t Wert aus Tabellenwerten interpolieren

  ```
  cp_t = interp1(tD, cpD, t);
  ```

  daraus gesuchte Werte für ein Intervall berechnen

  $$\overline{c_p}|_{T1}^{T2} = \frac{(T_2 - T_0)\overline{c_p}|_{T0}^{T2} - (T_1 - T_0)\overline{c_p}|_{T0}^{T1}}{T_2 - T_1}$$

  in Matlab einfach

  ```
  cp12 = (t2*cp_t2 - t1*cp_t1)/(t2 - t1);
  ```

- Sicheres Startintervall für `fzero`:

  untere Grenze

  - auf jeden Fall ist $t_{min} = t_1$ zu klein, außerdem gilt

    $$\begin{aligned} f(t_{min}) &= \dot{Q}_{12} - \dot{m}\,\overline{c_p}|_{t_1}^{t_{min}}\,(t_{min} - t_1) \\ &= \dot{Q}_{12} \\ &> 0 \end{aligned}$$

  - Problem: Matlab berechnet 0/0
  - Lösung: sehr kleinen Wert `epsi` zu t_{min} addieren

  Abschätzung für obere Grenze

  - kleinste auftretende Wärmekapazität als konstanten Wert $c_{p,min}$ nehmen
  - damit Endtemperatur abschätzen

    $$t_{max} = t_1 + \dot{Q}_{12}/(\dot{m}\,c_{p,min})$$

  - → t_{max} auf jeden Fall zu groß

- außerdem ist

$$\begin{aligned} f(t_{max}) &= \dot{Q}_{12} - \dot{m}\,\overline{c_p}\big|_{t_1}^{t_{max}}\,(t_{max} - t_1) \\ &= \dot{Q}_{12} - \dot{m}\,\overline{c_p}\big|_{t_1}^{t_{max}}\,\frac{\dot{Q}_{12}}{\dot{m}\,c_{p,min}} \\ &= \dot{Q}_{12}\left(1 - \frac{\overline{c_p}\big|_{t_1}^{t_{max}}}{c_{p,min}}\right) \\ &< 0 \end{aligned}$$

mit Intervall `[tmin, tmax]` klappt `fzero` sicher

Ergebnis: t_2 = 602.98 °C

komplettes Skript `computeEndTemperature.m`

Versuchsauswertung eines Zugversuchs

- Aufgabenstellung:

  Zugversuch mit 6 identischen Proben (kreisförmiger Querschnitt)

  jeweils Kräfte und Längenänderungen gemessen

  Messergebnisse abgespeichert in tensileTest.dat

  Spannungs-Dehnungs-Diagramme darstellen

  daraus E-Modul des Werkstoffs ermitteln

- Einlesen der Daten:

  Aufbau der Messdatei

```
# allgemeine Daten
 60.00     #  L0 [mm]
  4.00     #  d [mm]
# Zugversuche
# jeweils Delta L [mm] F [kN]
# Zugversuch 1
  0.000   0.000
  0.010   0.417
     ...
  6.640   9.549
  6.650   9.541
# Zugversuch 2
  0.000   0.000
  0.010   0.430
     ...
  6.680   9.282
  6.690   9.410
```

  Problem
  - Datensätze nicht gleich lang → können nicht in Matrix abgespeichert werden
  - Standardtrick: mit NaN's auffüllen

  zentrale Routine `textscan`
  - Formatstring beschreibt erwartete Daten
  - liest jeweils soweit, wie das Muster passt
  - Kommentare können überlesen werden

  Sammeln von Daten in **Cell-Array**
  - verhält sich grundsätzlich wie ein normales Array
  - Unterschied: Felder können ganz unterschiedliche Daten enthalten, auch komplette Arrays oder weitere Cell-Arrays
  - Zugriff mit `A{}` statt mit `A()`

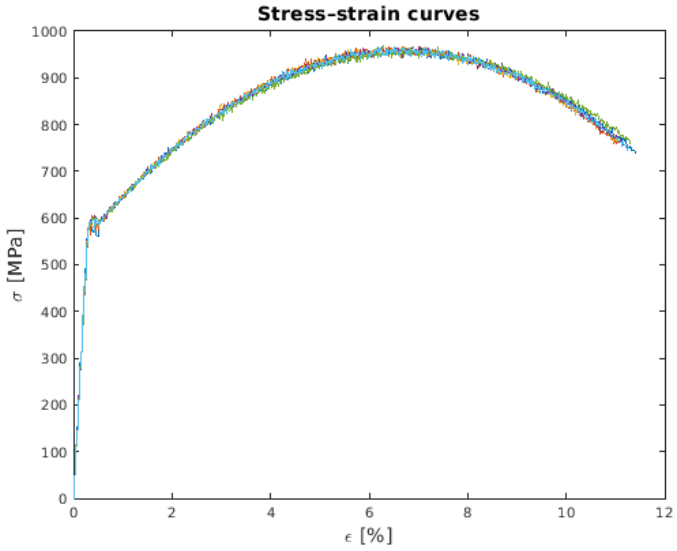
  Grundoperation für Dateibearbeitung
  - `fopen`, `fclose`, `fgetl`, `feof`

  komplette Funktion `readTensileData.m`

- Daten darstellen und statistisch auswerten:

  komplette Funktion `tensileTest.m`

  in Spannungen und Dehnungen umrechnen und graphisch darstellen

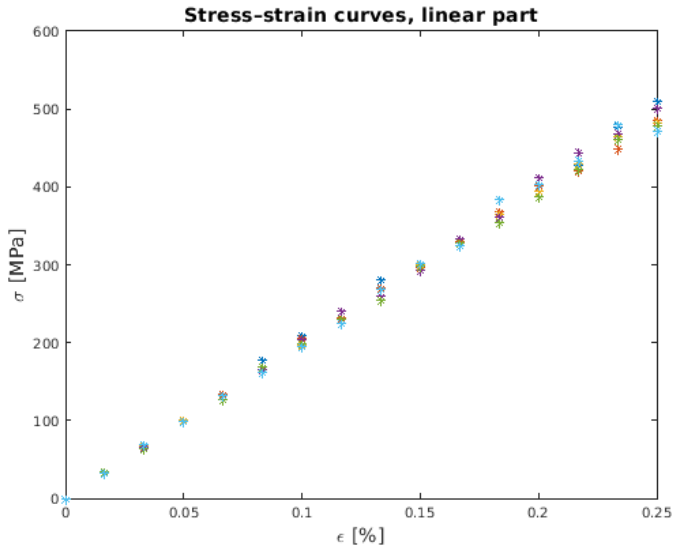


linearen Anteil ausfiltern

- Problem: wie weit ist die Kurve (nahezu) linear?
- hier einfach aus Graph ablesen: bis ε = 0.25%

```
epsMax = 0.25;
idxMax = find(epsMat(:,1) <= epsMax, 1, "last");
epsLin = epsMat(1:idxMax, 1);
sigLin = sigMat(1:idxMax, :);
```

- Ergebnis

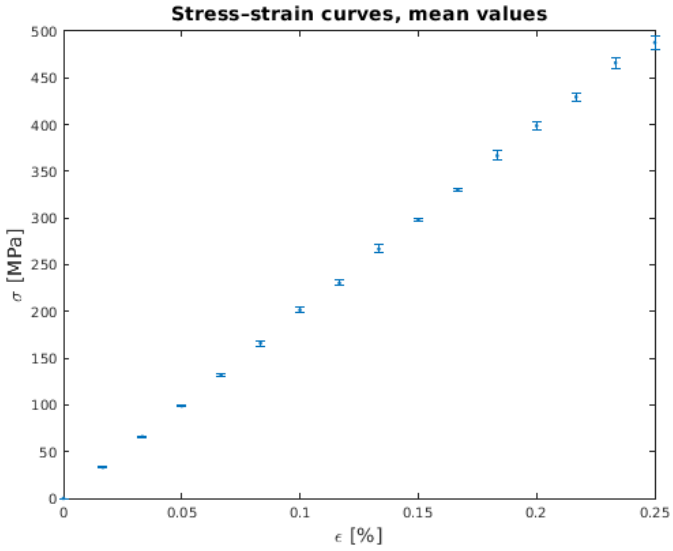


statistisch zusammenfassen

- `mean` und `std` fassen spaltenweise zusammen
- Datensätze hier horizontal, also transponieren
- NaNs auslassen

```
sigMean = mean(sigLin', "omitnan")';
sigDevs = std(sigLin', "omitnan")';
```

- Mittelwerte mit Fehlerbalken



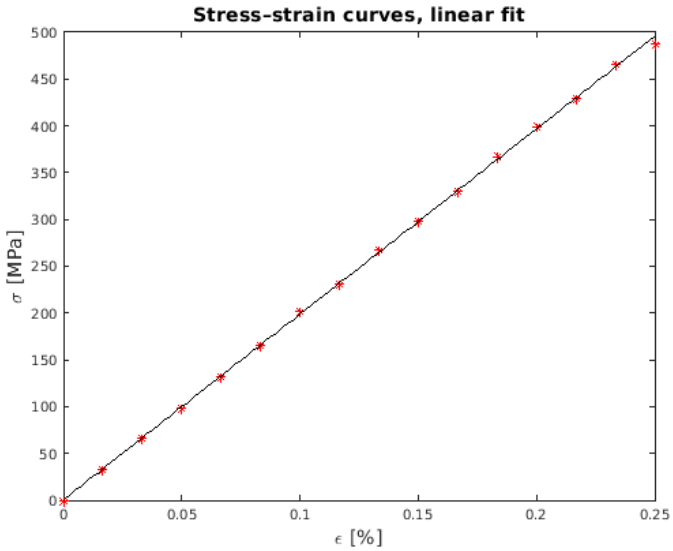
- Ausgleichsgerade und E-Modul:

  Ausgleichsgerade ohne Berücksichtigung der Fehlerbalken

  `polyfit` liefert
  - y = 1979.8 x + 0.8395
  - mit x ≙ ε [%], y ≙ σ [MPa]

  damit E = 198.0 GPa

  Ausgleichsgerade plotten mit `polyval`

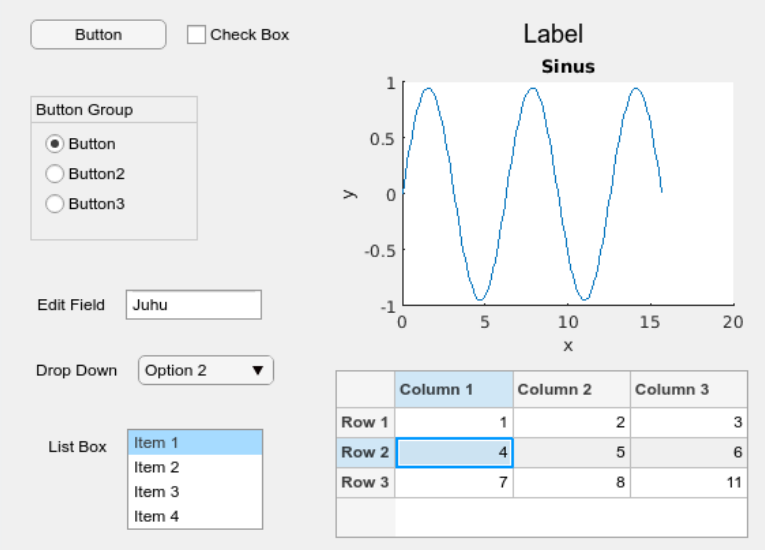


  Bestimmung der Genauigkeit von E
  - Berücksichtigen der Standardabweichungen
  - Berechnen mit statistischer Regressionsanalyse

Erstellen graphischer Oberflächen

- Graphische Bedienungs-Oberfläche (**GUI**):

  graphische Anzeige in einem oder mehreren Fenstern zur interaktiven Arbeit mit einem Programm



  Eingabe-Elemente
  - Push-Button
  - Check-Box
  - Radio-Button
  - Textfeld
  - Drop-Down-Menü
  - Listbox
  - Tabelle

  Ausgabe-Elemente
  - Textfeld
  - Graphik
  - Tabelle

  zu jedem Eingabe-Element gehört eine Ausführungsfunktion (**Callback**)

  Funktionsweise
  - Programm erzeugt beim Start Fenster mit GUI-Elementen
  - wartet auf Eingabe
  - ruft bei Eingabe die Callback-Funktion des aktivierten Eingabe-Elements auf
  - diese erzeugt (u.a.) Ausgabewerte für die Ausgabe-Elemente

  ereignis-gesteuerte Programmierung
  - Benutzer löst eine Aktion aus (Klick auf Button, Eingabe von Werten)
  - Eingabe-Element löst ein Event aus
  - dem Event zugeordnete Callback-Funktion wird ausgelöst

- Vorhandene Funktionen zum Einbauen in die Oberfläche:

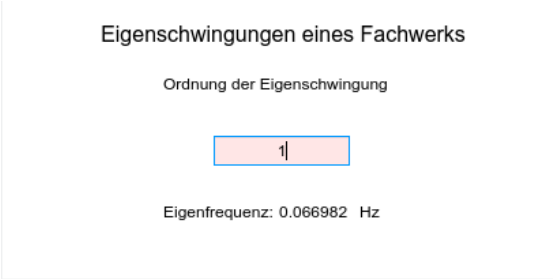
```
truss = loadTruss("bruecke");
[M, C] = createMatrices(truss);
[Phi, freq] = computeEigenvalues(M, C);
plotTruss(axes, truss);
plotMode(axes, Phi(:,1), truss);
```

```
F = plotModeAnimation(axes, xe, truss);
```

- Beispielprogramm `trussGUI1`:

  Aufgabenstellung
  - Berechnung der Eigenfrequenzen des Beispiel-Fachwerks
  - Eingabe der Ordnung
  - Ausgabe der Frequenz

  gewünschte GUI

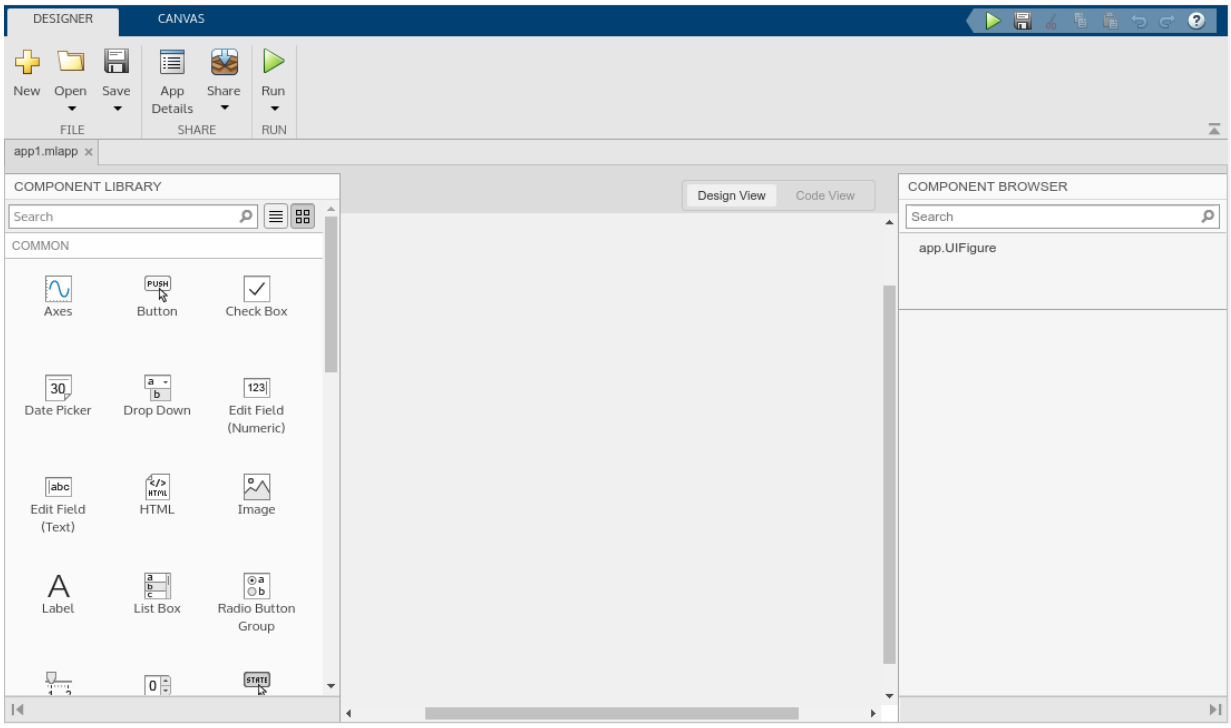


  Erstellen der GUI
  - hier mit interactivem Werkzeug **App Designer**
  - alternativ auch komplette GUI-Erstellung mit M-File-Programmierung möglich

- Starten von App Designer:

  `appdesigner` am Matlab-Prompt eingeben

  Auswahl von `Blank App` → App-Designer-Fenster erscheint



  Fenstergröße verkleinern ("Anfasser" unten rechts)

- Einfügen der benötigten Elemente:

  Komponenten aus der `Component Library` (Panel links) einfügen

  für die Überschrift `Label`

  weiteres `Label` für Hinweistext

  zur Eingabe der Ordnung `Edit Field (Numeric)` mit `Strg` → kein zusätzliches Textfeld

  drei weitere `Label` nebeneinander für Hinweistext, Ausgabe des Zahlenwerts und Einheit

abspeichern unter `trussGUI1.mlapp`

- Anpassen der Komponenten:

  `Component Browser` (Panel rechts)
  - zeigt oben Liste aller Komponenten
  - können über Kontext-Menü (Rechts-Klick) umgenannt oder gelöscht werden
  - nenne `Label` um in `Label1`

  Klick auf eine Komponente (oder im Design-Fenster)
  - `Component Browser/Inspector` zeigt Liste aller Parameter mit ihren Werten
  - können direkt editiert werden

  Parameter für Überschrift (`Label1`)

Eigenschaft	alter Wert	neuer Wert
`FontSize`	`12`	`16`
`Text`	`Label`	`Eigenschwingungen eines Fachwerks`
`BackgroundColor`	hellgrau	z.B. weiß (`[1.0 1.0 1.0]`)

  - Darstellungsfenster (Größe und Position) anpassen

  analog für die weiteren Elemente

Element	Eigenschaft	alter Wert	neuer Wert
`Label2`	`Text`	`Label2`	`Ordnung der Eigenschwingung`
`Label2`	`BackgroundColor`	hellgrau	weiß
`Label3`	`Text`	`Label3`	`Eigenfrequenz:`
`Label3`	`BackgroundColor`	hellgrau	weiß
`Label4`	`Text`	`Label4`	`-1.234567`
`Label4`	`BackgroundColor`	hellgrau	weiß
`Label5`	`Text`	`Label5`	`Hz`
`Label5`	`BackgroundColor`	hellgrau	weiß
`EditField`	`Value`	`0`	`1`
`EditField`	`HorizontalAlignment`	rechts	mittig
`EditField`	`BackgroundColor`	weiß	rosa (`1.0, 0.9, 0.9`)

  - Namen der Label ändern sich automatisch, evtl. manuell umnennen

  allgemeine Eigenschaften
  - durch Klick auf den Hintergrund oder Auswahl von `UIFigure`
  - `Color` von hellgrau auf weiß

  Hinweise zum Fine-Tuning
  - `Show grid/Snap to Grid` ggf. aktivieren
  - Position und Größe (z.B. `100,100,405,205`) evtl. direkt eintragen bei Parameter `Position`

  abspeichern mit `Save`

  Programm starten mit `Run` im App-Designer oder `trussGUI1` von der Kommandozeile
  - fertige Oberfläche erscheint
  - Wert der Ordnung kann geändert werden - aber nur als gültiges Zahlenformat
  - `<ENTER>` im geänderten Textfeld bewirkt (natürlich!) nichts

- Einbau der Callback-Funktion:

  Bei Eingabe eines Wertes (mit `<ENTER>` bestätigt) soll die entsprechende Eigenfrequenz angezeigt werden

Callback-Funktion des Textfelds `EditField` anlegen

- `Kontext-Menü von app.EditField/Callbacks/Add ValueChangedFcn callback`
- Funktion `EditFieldValueChanged(app, event)` wird erzeugt
- automatisch umgeschaltet auf `Code View`
- weiß hinterlegter Code: kann (muss!) manuell eingegeben werden

Parameter von `EditFieldValueChanged`

`app`	Objekt `app` mit allen UI-Komponenten
`event`	event-spezifische Informationen

eingegebenen Wert holen (als `double`)

```
modeNr = app.EditField.Value;
```

Eigenfrequenz `fe` berechnen, zunächst Dummy-Berechnung

```
fe = modeNr/12;
```

`fe` zur Ausgabe in eine Zeichenkette mit 6 Nachkommastellen wandeln

```
sFreq = sprintf("%8.6f", fe);
```

Text von Textfeld `Label4` auf diesen Wert setzen

```
app.Label4.Text = sFreq;
```

kompletter Inhalt der Callback-Funktion (incl. Kontrolle der Eingabe)

```
truss = loadTruss("bruecke");
[M, C] = createMatrices(truss);
[Phi, freq] = computeEigenvalues(M, C);

modeNr = app.EditField.Value;
if (1 <= modeNr) && (modeNr <= 2*truss.N)
   fe = freq(modeNr);
   sFreq = sprintf("%8.6f", fe); % Eigenfrequenz als String
   app.Label4.Text = sFreq;
end
```

Testen → klappt!

- Verbessern von `trussGUI1`:

  Probleme
  - zeigt am Anfang sinnlosen Wert
  - berechnet bei jeder Eingabe Fachwerk und seine Eigenfrequenzen neu

  Lösungsidee
  - beim Programmstart wird eine Startup-Funktion automatisch ausgeführt
  - darin Berechnung der Eigenfrequenzen
  - dort auch Wert zu `modeNr` = 1 anzeigen

  Startup-Funktion anlegen
  - `Kontext-Menü von UIFigure/Callbacks/Add StartupFcn callback`

  folgenden Code einfügen

```
truss = loadTruss("bruecke");
[M, C] = createMatrices(truss);
[Phi, freq] = computeEigenvalues(M, C);

% Setze Frequenz für Startwert 1
fe = freq(1);
```

```
sFreq = sprintf("%8.6f", fe);
app.Label4.Text = sFreq;
```

automatisch erzeugte Zeilen folgen (nicht löschen!)

Test klappt, Anfangswert wird richtig angezeigt

nun in `EditFieldValueChanged` Zeilen 1-3 löschen (Berechnung von `truss` und `freq`)

→ Eingabe neuer Werte klappt nicht mehr

Ursache:

in `startupFcn` definierte Werte (`truss`, `freq`) sind in `EditFieldValueChanged` nicht bekannt!

- Übergeben von Benutzer-Informationen:

  app-weite Daten (**Properties**) anlegen

  - im Code Browser von `Callbacks` auf `Properties` wechseln
  - `+ Private Property` → neue Property namens `app.Property`
  - mit Kontextmenü umnennen zu `app.Truss`
  - weitere Property `app.Freq` anlegen
  - Beschreibung der Properties hinzufügen

  ```
  Truss % Fachwerk-Struktur
  Freq  % Vektor der Eigenfrequenzen
  ```

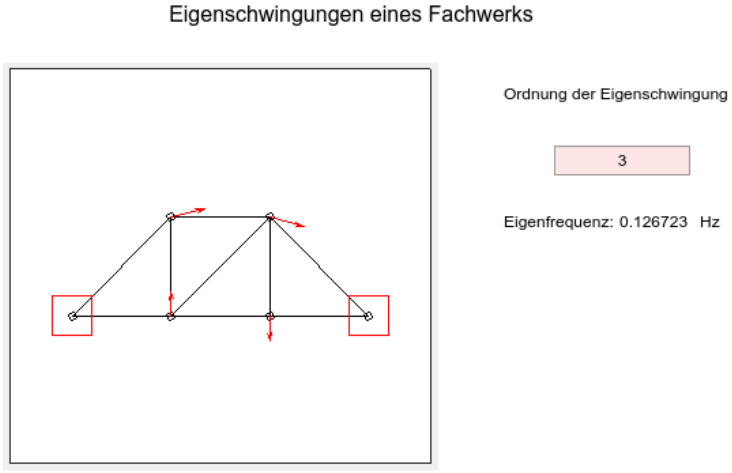
  `startupFcn` und `EditFieldValueChanged` anpassen

  - ersetze `truss` durch `app.Truss`
  - ersetze `freq` durch `app.Freq`

  komplettes Programm: trussGUI1.mlapp

- GUI mit Plotbereich:

  Ziel: Erweitern von `trussGUI` um Anzeige der Eigenvektoren



  Vorbereitungen mit App Designer

  - `trussGUI1.mlapp` öffnen
  - als `trussGUI2.mlapp` speichern
  - Layout-Fläche vergrößern (auf 600 x 400 Pixel)
  - Überschrift mittig anordnen
  - restliche Texte nach rechts schieben

  Plotbereich

  - Achsenkreuz (`Axes`) hinzufügen

- Größe (`Position`): 320 x 300 Pixel
- Strings für `Title`, `XLabel` und `YLabel` entfernen
- `Box Styling`: Checkbox `Box` aktivieren

Plot-Kommandos in `startupFcn` am Ende hinzufügen

```
plotTruss(app.UIAxes, app.Truss)
plotMode(app.UIAxes, Phi(:,1), app.Truss);
```

- starten → klappt

plotten nach Mode-Änderung

- Eigenvektoren als neue Property `app.Phi`
- in `startupFcn` `Phi` durch `app.Phi` ersetzen
- Rechnung in `EditFieldValueChanged` ergänzen um

```
plotTruss(app.UIAxes, app.Truss)
plotMode(app.UIAxes, app.Phi(:,modeNr), app.Truss);
```

- Testen → klappt

Toolbar ausschalten

- Maus im Plot → Toolbar erscheint

- in der `startupFcn` ausschalten

```
app.UIAxes.Toolbar.Visible = "off";
```

Ergebnis in trussGUI2.mlapp

- Animation der Eigenschwingung hinzufügen:

  mit App Designer `trussGUI2` öffnen und als `trussGUI` speichern

  Button hinzufügen

  - Beschriftung auf `Animation` ändern
  - Größe und Farbe anpassen

  Callback `ButtonPushedFcn` hinzufügen

  `AnimationButtonPushed` füllen mit

```
modeNr = app.EditField.Value;
if (1 <= modeNr) && (modeNr <= 2*app.Truss.N)
  xe = app.Phi(:, modeNr);
  plotModeAnimation(app.UIAxes, xe, app.Truss, 4);
  plotTruss(app.UIAxes, app.Truss)
  plotMode(app.UIAxes, xe, app.Truss);
end
```

- Abspeichern erzeugter Plots:

  Problem: Zugriff auf GUI-Figure und -Achsen von außen nicht möglich

  Lösung: füge Button "Save Plot" incl. Callback hinzu

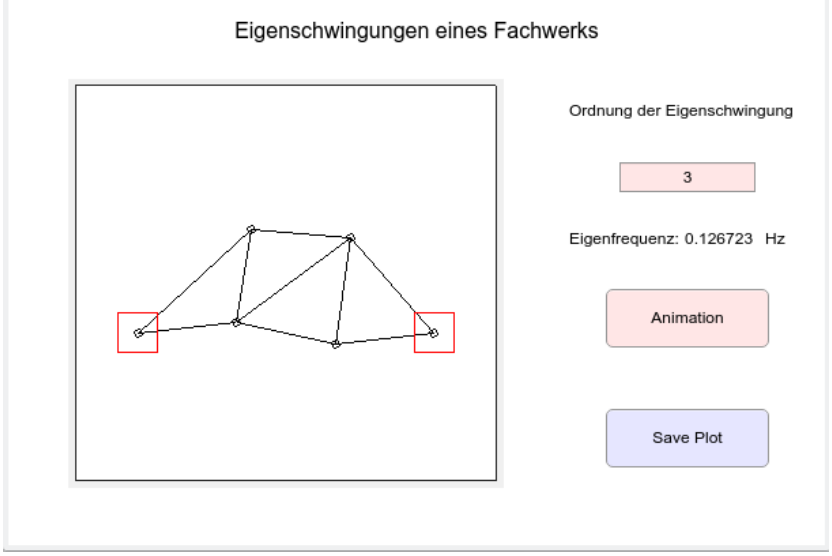
  weiteres Problem

  - `getframe`, `saveas` funktionieren nicht in App Designer
  - Alternative ab Version 2020a: `exportgraphics`

  Inhalt von `SavePlotButtonPushed`

```
exportgraphics(app.UIAxes, "plot.png");
```

fertige Version in trussGUI.mlapp



Aufgaben

Schreiben Sie zu jeder Aufgabe eine Matlab-Funktion `aufgabeNR()` (NR = Nummer der Aufgabe), die die komplette Fragestellung löst bzw. die zu erstellenden Funktionen an einfachen Daten testet.

Aufgabe 1

- Untersuchen Sie das Phänomen der Schwebung, indem Sie Kurven plotten für die Funktionen

  $y1 = \sin(\omega_1 t)$

  $y2 = \sin(\omega_2 t)$

  und ihre Summe $s = y1 + y2$ für die Werte

  $\omega_1 = 0.8 \cdot 1/s$

  $\omega_2 = 0.85 \cdot 1/s$

  Bestimmen Sie die Schwebungsfrequenz durch Ablesen aus dem Plot.

Aufgabe 2

- Der Wert für die spezifische Wärmekapazität c_p von Wasser bei $T_1 = 26.5$ °C ist gesucht.

  a. Die Datei ex02.dat enthält Ergebnisse von Messungen von $c_p(T)$. Auf einen Temperaturwert (in °C) folgt dabei immer der dazugehörige c_p-Wert (in kJ/(kg K)). Fitten Sie eine Gerade, eine quadratische Parabel und eine Parabel 4. Grades an die Messwerte und ermitteln Sie so Näherungswerte für $c_p(T_1)$.

  b. Plotten Sie die Messwerte zusammen mit den Fit-Kurven. Welche erscheint Ihnen am vertrauenswürdigsten?

- Hinweis: Die folgenden Funktionen könnten hilfreich sein:

  `load`, `polyfit`, `polyval`

Aufgabe 3

- Die Geschwindigkeit einer Flüssigkeit, die turbulent durch ein Rohr mit Radius r_0 strömt, lässt sich beschreiben durch

$$w(r) = w_{max} \cdot \left(1 - \frac{r}{r_0}\right)^{\frac{1}{n}}$$

  wobei der Parameter n von der Flüssigkeit und der Rohrbeschaffenheit abhängt.

- Die mittlere Geschwindigkeit erhält man dann durch

$$\overline{w} = \frac{2}{r_0^2} \int_0^{r_0} r w(r) dr$$

- Plotten Sie w(r) über den Rohrdurchmesser für die Werte n = 6, 7, 8, 9 und bestimmen Sie jeweils $\overline{w}$. Wählen Sie dazu die Werte r_0 = 5 cm und w_{max} = 2.5 m/s.

- Hinweis: Lösen Sie die Aufgabe zunächst für n = 6 fest und verwenden Sie Hilfsfunktionen, die nur vom Radius r abhängen.

Aufgabe 4

- Um die Leistung eines Verbrennungsmotors zu bestimmen, wurden für einen Zyklus des Kreisprozesses der Druck bei verschiedenen Kolbenständen (Volumina) gemessen. Die Tabelle ex04.dat enthält diese Daten in der Form von drei Spalten mit Werten

  V [cm^3] p_{unten}[bar] p_{oben} [bar]

  wobei p_{unten} bzw. p_{oben} jeweils der untere bzw. obere Druck beim Volumen V sind.

  a. Stellen Sie den Kreisprozess graphisch im pV-Diagramm dar.

  b. Bestimmen Sie die Leistung (für den einen gemessenen Zylinder), wenn die Drehzahl bei der Messung n = 3000 U/min betrug.

- Hinweise:
  - Die Kreisprozess-Arbeit ist gegeben durch die umschlossene Fläche

    $$W = \oint p dV$$

    Die Leistung ist dann

    P = W n
  - Zur Integration kann die Funktion `trapz` verwendet werden.

Aufgabe 5

- Erstellen Sie einen dreidimensionalen Plot der Funktion

$$f(x,y) = (x+y)^2 + \frac{1}{5}y^3 e^{-|x+y|}$$

und speichern Sie ihn vom Skript aus als Datei `"manta.png"`.

Aufgabe 6

- Vier Peilstationen befinden sich an den Orten

  x1 = (2.7, 2.5)

  x2 = (36.1, 5.5)

  x3 = (28.4, 31.9)

  x4 = (4.0, 24.6)

  Sie messen jeweils die Richtung zu einem Störsender und geben dafür folgende Werte an (Abweichung in Grad zur Nordrichtung = positive y-Richtung, gegen den Uhrzeigersinn):

  α_1 = -64.5°

  α_2 = 62.0°

  α_3 = 147.6°

  α_4 = -122.7°

- Erstellen Sie eine Funktion `senderOrt(peilOrte, winkel)`, die aus den Positionen der Peilstationen (als 4x2-Matrix) und den Winkeln (als Vektor) die wahrscheinlichste Position des Störsenders ermittelt.
- Hinweis: Stellen Sie zunächst die 4 Geradengleichungen auf, die sich aus den Informationen ergeben, und bringen Sie sie in die Standardform

  A x = b

  Dieses unbestimmte System (4 Gleichungen für 2 Unbekannte) lösen Sie in Matlab einfach mit

  x = A \ b

Aufgabe 7

- Finden Sie alle Lösungen der Gleichung

  $$\tan(x) = x \ln(0.1\, x)$$

  im Intervall [0, 10].

Aufgabe 8

- Berechnen Sie die Bahn x(t) eines Körpers, der unter Reibungseinfluss frei fällt. Seine Bewegungsgleichung lautet

  $m \ddot{x} = -m g + F_R(x)$

  mit

  - $F_R(x) = -b_1 \dot{x}$ (laminare Reibung)
  - $F_R(x) = -b_2 \dot{x}^2 \, \mathrm{sign}(\dot{x})$ (turbulente Reibung)

- Plotten Sie beide Kurven in einem Diagramm, zusammen mit der Bahn bei reibungslosem freien Fall. Plotten Sie ebenso die drei Geschwindigkeitskurven.

- Werte:

  $m = 75$ kg

  $g = 9.81$ m/s^2

  $b_1 = 5$ kg/s

  $b_2 = 0.1$ kg/m

  $x_0 = 5000$ m

  $v_0 = 0$

Anhang

Literatur

1. Angermann, Beuschel, Rau, Wohlfarth: Matlab - Simulink- Stateflow
   Oldenbourg, 10. Aufl 2020, ISBN: 978-3110641073

2. W. D. Pietruszka, M. Glöckler: MATLAB und Simulink in der Ingenieurpraxis,
   Springer Vieweg, 5. Aufl. 2021, ISBN: 978-3658297398

3. B. R. Hunt, R. L. Lipsman et. al.: A Guide to MATLAB
   Cambridge University Press, 3. Aufl. 2014, ISBN: 978-1107662223

4. U. Stein: Programmieren mit MATLAB
   Carl Hanser Verlag, 6. Aufl. 2017, ISBN: 978-3446448643

5. S. Attaway: MATLAB: A Practical Introduction to Programming and Problem Solving
   Butterworth-Heinemann, 5. Aufl. 2018, ISBN: 978-0128154793

Matlab-Beispiele

- computeEigenvalues.m
- computeEndTemperature.m
- createMatrices.m
- demo2dfunc.m
- erzwungen.m
- erzwungenp.m
- glocke.m
- loadTruss.m
- plotMode.m
- plotModeAnimation.m
- plotTruss.m
- radioaktiv.m
- radiokette.m
- readTensileData.m
- tensileTest.m
- trapez1.m
- trapez2.m
- trapez.m
- zeichneFunktion.m

computeEigenvalues.m

```
function [Phi, freq] = computeEigenvalues(M, C)
% berechnet die Eigenfrequenzen freq(i) und Eigenvektoren Phi(:,i)
% Eigenvektoren sind auf Länge 1 normiert
% Parameter
%    M     Massenmatrix
%    C     Steifigkeitsmatrix

[Phi, om2] = eig(C,M);
freq = sqrt(diag(om2))/(2*pi);

% Eigenvektoren auf Länge 1 normieren
norms = sqrt(diag(Phi'*Phi));
Phi = Phi*diag(1./norms);
```

createMatrices.m

```
function [Mass, C] = createMatrices(truss)
% berechnet die Massen- und Steifigkeitsmatrix des gegebenen Fachwerks
% Indizes i, j von 1 .. M
% Index k von 1 .. N
% Indizes a, b  Werte 1, 2  (= x, y)

M = size(truss.A,1);
N = truss.N;

% Verbindungsvektoren e(i,j,a)
e0 = zeros(M,M,2);
for I=1:M
  for J = 1:(I-1)
    % oben rechts
    e0(I,J,:) = truss.x0(:,I) - truss.x0(:,J);
    e0(I,J,:) = e0(I,J,:)/norm(squeeze(e0(I,J,:)));
    % unten links durch Antisymmetrie
    e0(J,I,:) = -e0(I,J,:);
  end
end

%
% Steifigkeitsmatrix C
%
% Doppelindex j a in Einfachindex m
%   m = 2*(k-1) + a
% also: 1x, 1y, 2x, 2y, 3x, 3y, ...
C = zeros(2*N,2*N);

% Diagonalelemente
for K = 1:N
  for J = 1:M
    C(2*K-1,2*K-1) = C(2*K-1,2*K-1) + truss.A(K,J)*e0(K,J,1)^2;  % C(kx,kx)
    C(2*K-1,2*K  ) = C(2*K-1,2*K  ) + truss.A(K,J)*e0(K,J,1)*e0(K,J,2); % C(kx,ky)
    C(2*K  ,2*K  ) = C(2*K  ,2*K  ) + truss.A(K,J)*e0(K,J,2)^2;  % C(ky,ky)
  end
  C(2*K,2*K-1) = C(2*K-1,2*K);  % C(ky,kx) = C(kx,ky)
end

% Nicht-Diagonalelemente
for K = 1:N
  for J = 1:(K-1)
     % oben rechts
     C(2*K-1,2*J-1) = -truss.A(K,J)*e0(K,J,1)^2;  % C(kx,jx)
     C(2*K-1,2*J  ) = -truss.A(K,J)*e0(K,J,1)*e0(K,J,2); % C(kx,jy)
     C(2*K  ,2*J-1) = C(2*K-1,2*J  );   % C(ky,jx)
     C(2*K  ,2*J  ) = -truss.A(K,J)*e0(K,J,2)^2;  % C(ky,jy)
     % unten links durch Symmetrie
     C(2*J-1,2*K-1) = C(2*K-1,2*J-1);
     C(2*J  ,2*K-1) = C(2*K-1,2*J  );
     C(2*J-1,2*K  ) = C(2*K  ,2*J-1);
     C(2*J  ,2*K  ) = C(2*K  ,2*J  );
  end
end
C = truss.c * C;

Mass = truss.m*eye(2*N,2*N);
```

computeEndTemperature.m

```
function computeEndTemperature()
% solution of exercise 11
  mdot = 5.668e-3;        % in kg/s
  Q12dot = 3225;          % in W
  t1 = 70;                % in °C
  M = 44.01e-3;           % in kg/mol, für CO2

  % read data
  data = load("cps.dat");
  tD = data(:,1);         % in °C
  cpD = data(:,3)/M;      % in J/(kg K)

  % equation as zero of a function
  f = @(t) Q12dot - mdot*cpQuer(t1, t, tD, cpD).*(t - t1);

  % find interval surrounding solution
  epsi = 1e-5;
  tMin = t1 + epsi;
  cpmin = min(cpD);
  tMax = t1 + Q12dot/(mdot*cpmin);

  t2 = fzero(f, [tMin, tMax]);
  fprintf("Endtemperatur t2 = %9.4f\n", t2)

end

%-----------------------------------------------------------------------
function y = cpQuer(t1, t2, tD, cpD)
  % berechnet die mittlere spezifische Wärmekapazität von t1 bis t2
  % t1        Starttemperatur in °C
  % t2        Starttemperatur in °C
  % tD        Tabellenwerte für t
  % cpD       Tabellenwerte für cp
  % y         Wärmekapazität in J/(kg K)

  cp_t1 = interp1(tD, cpD, t1);
  cp_t2 = interp1(tD, cpD, t2);
  y = (t2.*cp_t2 - t1.*cp_t1)./(t2 - t1);  % T0 = 0 °C
end
```

demo2dfunc.m

```
function z = demo2dfunc(x,y)
% Beispielfunktion fuer Kapitel 4
z = sin(x.^2) + cos(y);
```

erzwungen.m

```
function dy = erzwungen(t, y)
% rechte Seite der DGL bei erzwungener Schwingung

% Parameter
m = 1.0;
b = 0.3;
c = 1.0;
A = 1.0;
omega = 1.0;

dy1 = y(2);
dy2 = -b/m*y(2) - c/m*y(1) + A/m*cos(omega*t);
dy = [dy1; dy2];
```

erzwungenp.m

```
function dy = erzwungenp(t, y, m, b, c, A, omega)
% rechte Seite der DGL bei erzwungener Schwingung
% Parameter:
% m       Masse
% b       Reibungskoeffizient
% c       Federkonstante
% A       Amplitude der äußeren Kraft
% omega   Kreisfrequenz der äußeren Kraft

dy1 = y(2);
dy2 = -b/m*y(2) - c/m*y(1) + A/m*cos(omega*t);
dy = [dy1; dy2];
```

glocke.m

```
function y = glocke(x)
% Glockenfunktion e^(-x^2)
y = exp(-x.^2);
```

loadTruss.m

```
function truss = loadTruss(filename)
% lädt eine Fachwerkstruktur
% Parameter:
%    filename  Name der Datei (ohne Endung .truss)
% Felder von truss:
%    x0        Gleichgewichtskoordinaten aller Punkte
%    A         Verbindungsmatrix (symmetrisch, auf der Diagonalen 0)
%    N         Zahl der beweglichen Massen
%    m         Größe einer Masse
%    c         Steifigkeit eines Balkens

fid = fopen(filename + ".truss", "r");
if (fid == -1)
  fprintf("Datei %s.truss konnte nicht geöffnet werden\n", filename)
  % Dummy-Rückgabewerte
  truss.x0 = []; truss.A = []; truss.N = 0; truss.m = 0; truss.c = 0;
  return
end

fgetl(fid);
dims = fscanf(fid, "%d %d\n", 2);
truss.x0 = fscanf(fid, "%f", dims');
fgetl(fid);         % holt das \n von der x0-Zeile

fgetl(fid);
dims = fscanf(fid, "%d %d\n", 2);
truss.A = fscanf(fid, "%f", dims');
fgetl(fid);    % holt das \n von der A-Zeile

fgetl(fid);
truss.N = fscanf(fid, "%d\n", 1);

fgetl(fid);
truss.m = fscanf(fid, "%f\n", 1);

fgetl(fid);
truss.c = fscanf(fid, "%f\n", 1);

fclose(fid);
```

plotMode.m

```
function plotMode(axes, xe, truss)
% zeichnet den Eigenvektor xe als Verschiebungsvektoren an den Orten
% der Fachwerkknoten
% Parameter
%    axes      verwendetes Achsensystem
%    xe        Eigenvektor der Fachwerk-Schwingung
%    truss     Fachwerk

xp = truss.x0(1,:)';
yp = truss.x0(2,:)';
N = length(xe)/2;

hold(axes, "on")
quiver(axes, xp(1:N), yp(1:N), xe(1:2:end), xe(2:2:end), ...
      "r-", "AutoScaleFactor", 0.5);
hold(axes, "off")
```

plotModeAnimation.m

```
function plotModeAnimation(axes, xe, truss, nLoop)
% zeigt eine Animation der Eigenschwingung xe
% Parameter
%    axes      verwendetes Achsensystem
%    xe        Eigenvektor der Fachwerk-Schwingung
%    truss     Fachwerk
%    nLoop     Anzahl angezeigter Schwingungen

amp = 0.3;          % Amplitude der Eigenschwingung
nFrames = 16;       % Zahl der Bilder pro Schwingung

N = truss.N;
trussT = truss;

for j = 1:nFrames*nLoop
  t = 2*pi*j/nFrames;
  trussT.x0(1,1:N) = truss.x0(1,1:N) + amp*sin(t)*xe(1:2:end)';
  trussT.x0(2,1:N) = truss.x0(2,1:N) + amp*sin(t)*xe(2:2:end)';

  plotTruss(axes, trussT);
  drawnow;
end
```

plotTruss.m

```
function plotTruss(axes, truss)
% zeichnet das Fachwerk truss

xp = truss.x0(1,:);
yp = truss.x0(2,:);
M = length(xp);

% zeichne Knoten
plot(axes, xp, yp, "ko")
axis(axes, [-0.5 3.5 -1.5 2.5])
set(axes, "XTick", [], "YTick", [])

% zeichne Verbindungen
hold(axes, "on")
for I=1:M
  for J = 1:(I-1)
    if truss.A(I,J) ~= 0
       plot(axes, [xp(I) xp(J)], [yp(I) yp(J)], "k-")
    end
  end
end

% markiere fixierte Knoten
for I=truss.N+1:M
  plotSquare(axes, xp(I), yp(I))
end
hold(axes, "off")

% mache Achsensystem quadratisch
axis(axes, "equal")

%-----------------------------------------------------------------------

function plotSquare(axes, x, y)
% zeichnet ein kleines rotes Rechteck um (x,y)

d = 0.2;
xi = [x-d, x+d, x+d, x-d, x-d];
yi = [y+d, y+d, y-d, y-d, y+d];

plot(axes, xi, yi, "r-")
```

radioaktiv.m

```
function dy = radioaktiv(t, y)
% rechte Seite der DGL beim radioaktiven Zerfall
% Parameter lambda fest
lambda = 0.1;
dy = -lambda*y;
```

radiokette.m

```
function dy = radiokette(t, y)
% rechte Seite der DGL bei radioaktiver Zerfallskette

% Parameter lambda1, lambda2 fest
lambda1 = 0.1;
lambda2 = 0.03;

dy1 = -lambda1*y(1);
dy2 = lambda1*y(1)-lambda2*y(2);
dy = [dy1; dy2];      % Spaltenvektor
```

readTensileData.m

```
function [L, d, DLMat, FMat] = readTensileData(filename)
% read tensile test data from file
% parameter
%   filename     file with test data
% returns
%   L             specimen length [mm]
%   d             specimen diameter [mm]
%   DLMat         elongation [mm]
%   FMat          applied force [kN]
fid = fopen(filename,"r");

% read header
data = textscan(fid,"%f", 2, "CommentStyle", "#");
L = data{1}(1);                 % in mm
d = data{1}(2);                 % in mm

% read data sets
I = 0;
while ~feof(fid)
  data1 = textscan(fid,"%f %f", "EmptyValue", NaN); % read data til comment
  I = I + 1;
  len(I) = size(data1{1},1);    % collect data sizes
  DL{I} = data1{1};             % in mm
  F{I} = data1{2};              % in kN
  fgetl(fid);                   % read next comment
end

% prepare matrices for all datasets, fill with NaN's
N = max(len);                   % maximal data size
nSets = length(len);            % number of data sets
DLMat = NaN(N, nSets);
FMat = NaN(N, nSets);

% insert data columns in matrices
for I=1:nSets
  DLMat(1:size(DL{I},1), I) = DL{I};
  FMat(1:size(F{I},1), I) = F{I};
end

fclose(fid);
```

tensileTest.m

```
function tensileTest()
% reads data from a tensile test
% plots stress-strain curves and computes Young modulus

[L, d, DLMat, FMat] = readTensileData("tensileTest.dat");

epsMat = DLMat/L*100;     % in %
S0 = pi*d^2/4;            % in mm^2
sigMat = 1000*FMat/S0;    % in N/mm^2 = MPa

plot(epsMat, sigMat)
xlabel("\epsilon [%]", "Fontsize", 14)
ylabel("\sigma [MPa]", "Fontsize", 14)
title("Stress-strain curves", "Fontsize", 14)
F = getframe(gcf());
imwrite(F.cdata, "bild14.png");

% get linear data
epsMax = 0.25;
idxMax = find(epsMat(:,1) <= epsMax, 1, "last");

epsLin = epsMat(1:idxMax, 1);
sigLin = sigMat(1:idxMax, :);

plot(epsLin, sigLin, "*")
xlabel("\epsilon [%]", "Fontsize", 14)
ylabel("\sigma [MPa]", "Fontsize", 14)
title("Stress-strain curves, linear part", "Fontsize", 14)
F = getframe(gcf());
imwrite(F.cdata, "bild22.png");

% get mean values and standard deviations
sigMean = mean(sigLin', "omitnan")';
sigDevs = std(sigLin', "omitnan")';

errorbar(epsLin, sigMean, sigDevs/2, ".")
xlabel("\epsilon [%]", "Fontsize", 14)
ylabel("\sigma [MPa]", "Fontsize", 14)
title("Stress-strain curves, mean values", "Fontsize", 14)
F = getframe(gcf());
imwrite(F.cdata, "bild23.png");

% linear fit without using std's
poly1 = polyfit(epsLin, sigMean, 1)
E = poly1(1)/10;           % in GPa
fprintf("Young modulus without using standard deviations: %7.2f GPa\n", E)

epsMax = epsLin(end);
epsP = 0:epsMax/100:epsMax;
sigP = polyval(poly1, epsP);
plot(epsLin, sigMean, "r*", epsP, sigP, "k-")
xlabel("\epsilon [%]", "Fontsize", 14)
ylabel("\sigma [MPa]", "Fontsize", 14)
title("Stress-strain curves, linear fit", "Fontsize", 14)
F = getframe(gcf());
imwrite(F.cdata, "bild24.png");
```

trapez1.m

```
% Integration mit der Trapezregel
%   anpassen:
%       Grenzen a, b
%       Funktion (statt sin)
%       Zahl der Intervalle N
a = 0;
b = pi/2;
N = 100;

h = (b-a)/N;
x = a:h:b;
y = sin(x);
I = h*(0.5*y(1) + sum(y(2:N)) + 0.5*y(N+1))
```

trapez2.m

```
function I = trapez2(a, b, N)
% Integration mit der Trapezregel
%   Argumente
%       Grenzen a, b
%       Zahl der Intervalle N
%   Ergebnis
%     Integral des Sinus von a bis b,
%     genaehert mit Trapezregel mit N Intervallen

h = (b-a)/N;
x = a:h:b;
y = sin(x);
I = h*(0.5*y(1) + sum(y(2:N)) + 0.5*y(N+1));
```

trapez.m

```
function I = trapez(func, a, b, N)
% Integration mit der Trapezregel
%   Argumente
%       zu integrierende Funktion func
%       Grenzen a, b
%       Zahl der Intervalle N
%   Ergebnis
%     Integral von func von a bis b,
%     genaehert mit Trapezregel mit N Intervallen

h = (b-a)/N;
x = a:h:b;
y = func(x);
I = h*(0.5*y(1) + sum(y(2:N)) + 0.5*y(N+1));
```

zeichneFunktion.m

```
% Beispiel: Plot von Funktionen
t = 0:0.1:4*pi;
y = sin(t);
y2 = 0.5*log(t);
y3 = exp(-0.1*t.*t);

plot(t, y, "bo", t, y2, "g.", t, y3, "r-.", [0, 4*pi], [0, 0], "k-")
title("Meine Lieblingsfunktionen", ...
      "FontSize", 14, "FontWeight", "bold")
xlabel("Zeit");
ylabel("Auslenkung");
xlim([0, 4*pi])
legend("Sinus", "Logarithmus", "Glocke", "Location", "best");

% Plot als Datei abspeichern
F = getframe(gcf);
imwrite(F.cdata, "bild06.png");
```

Beispieldaten

- demo1.dat
- cps.dat
- tensileTest.dat
- ex02.dat
- ex04.dat
- bruecke.truss
- turm.truss
- trussGUI1.mlapp
- trussGUI2.mlapp
- trussGUI.mlapp