

- Einführung in Matlab
- Lösen von Gleichungen
- Programmieren mit Matlab
- Integration von Differentialgleichungen
- Berechnung von Eigenschwingungen
- Multimedia-Anwendungen
- Datenbanken und SQL
- Aufgaben
- Anhang

Peter Junglas 12. 07. 2023

Inhaltsverzeichnis

Übersicht

- Einführung in Matlab
 - Grundfunktionen
 - Vektoren und Plots
 - Erstellen eigener Funktionen
 - Matrizen
- Lösen von Gleichungen
- Programmieren mit Matlab
 - Ein- und Ausgabe
 - Kontrollstrukturen
 - Datentypen
- Integration von Differentialgleichungen
 - Grundlegende Beispiele
 - Mehrdimensionale Schwingungen
- Berechnung von Eigenschwingungen
- Multimedia-Anwendungen
 - Bilder
 - Animationen
 - Erstellen graphischer Oberflächen
 - Klänge
- Datenbanken und SQL
 - Datenbank-Managementsysteme
 - Aufbau von Tabellen
 - Grundlagen von SQL
 - Arbeiten mit Tabellen
 - Datenbankentwurf
 - Abfragen in Datenbanken
 - Einfache Abfragen
 - Abfragen über mehrere Tabellen
 - Komplizierte Abfragen
 - Datenbank-Zugriff mit Matlab
- Aufgaben
 - Aufgabe 1
 - Aufgabe 2
 - Aufgabe 3
 - Aufgabe 4
 - Aufgabe 5
 - Aufgabe 6
 - Aufgabe 7
 - Aufgabe 8
 - Aufgabe 9
 - Aufgabe 10
 - Aufgabe 11
 - Aufgabe 12
 - Aufgabe 13
 - Aufgabe 14
 - Aufgabe 15
 - Aufgabe 16
 - Aufgabe 17
 - Aufgabe 18
 - Aufgabe 19
- Anhang
 - Literatur
 - Matlab-Beispiele
 - bild26.m
 - bild28.m
 - computeEigenvalues.m
 - createDemoTruss.m
 - createEnvelope.m

- createMatrices.m
- createSineWave.m
- demo2dfunc.m
- erzwungen.m
- erzwungenp.m
- getNumber.m
- glocke.m
- loadTruss1.m
- loadTruss.m
- matschwing2d.m
- plotModeAnimation1.m
- plotModeAnimation.m
- plotMode.m
- plotTruss1.m
- plotTruss.m
- radioaktiv.m
- radiokette.m
- saveTruss1.m
- saveTruss.m
- schwing2d.m
- schwingNd.m
- showPapers.m
- solveVibrationODE.m
- spektrum.m
- trapez1.m
- trapez2.m
- trapez.m
- zeichneFunktion.m
- Ergebnisse der SQL-Abfragen für die Beispiel-Datenbank
- Beispieldaten

- Grundfunktionen
- Vektoren und Plots
- Erstellen eigener Funktionen
- Matrizen

- Matlab:

Umfangreiches Programmpaket für alle Arten numerischer Berechnungen

Hersteller: [The MathWorks Inc.](http://www.mathworks.com)

viele Zusatzpakete für spezielle Anwendungen, auch von Drittfirmen

bei uns (PC-Pool Diepholz) installiert

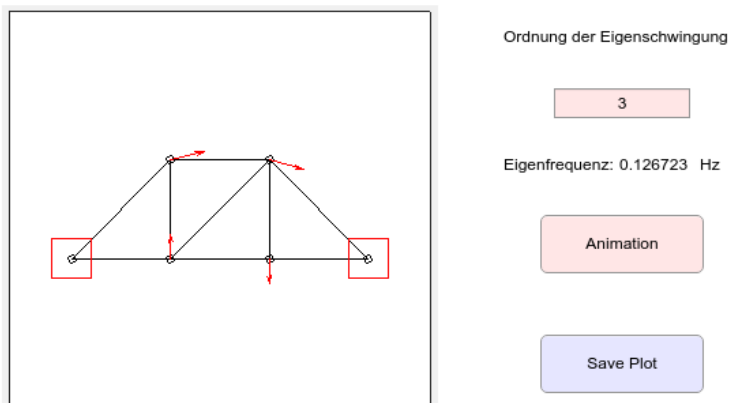
Paket	Zweck
Signal Processing Toolbox	Signalverarbeitung, Filter, ..
Image Processing Toolbox	Bildbearbeitung
Statistics and Machine Learning Toolbox	Statistische Beschreibung und Analyse von Daten
Simulink	graphisches Simulations-Tool
Simscape	Simulink-Blöcke für Physical Modeling
Stateflow	Simulink-Blöcke 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

Zielprogramm der Vorlesung

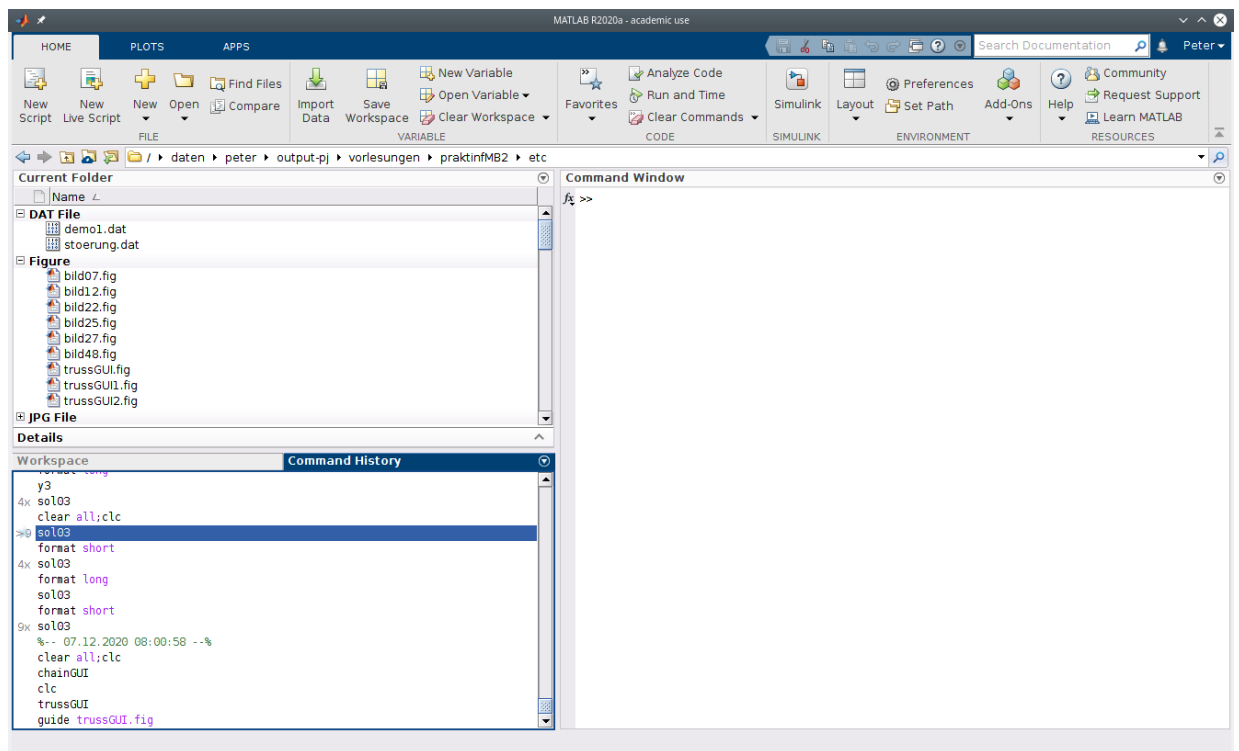
- Berechnung der Eigenschwingungen eines zweidimensionalen Fachwerks
- Darstellung und Animation der Schwingungen
- graphische Benutzeroberfläche

Eigenschwingungen eines Fachwerks



- 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 rechnerische 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?

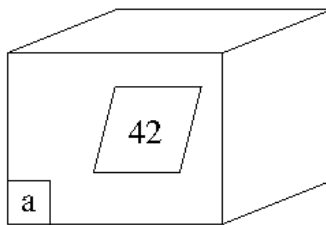
Bedeutung von $a = b$

- 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
```

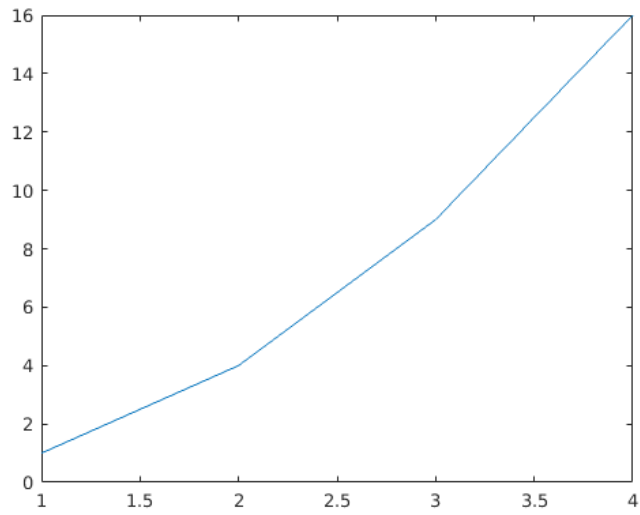
```
ans =
```

```
1.0000 1.4000 1.8000 2.2000 2.6000 3.0000
```

- Plotten von Vektoren:

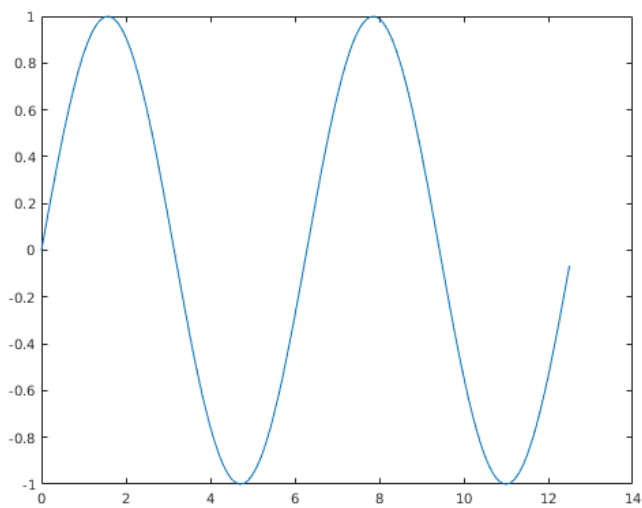
`plot` verbindet Punkte, gegeben durch Vektoren ihrer x - und y -Komponenten

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



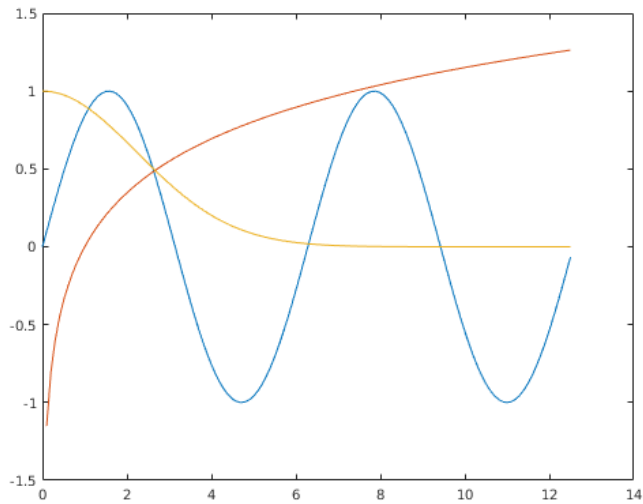
viele x-Werte → Plot einer Funktion

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



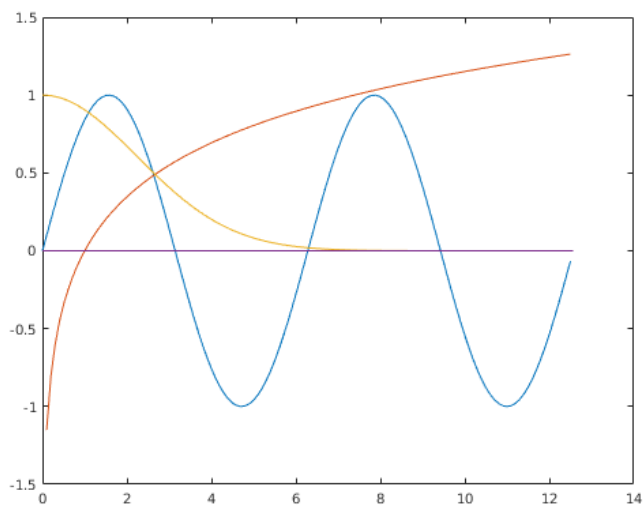
auch mehrere Vektoren möglich, jeweils als x- und y-Koordinaten

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



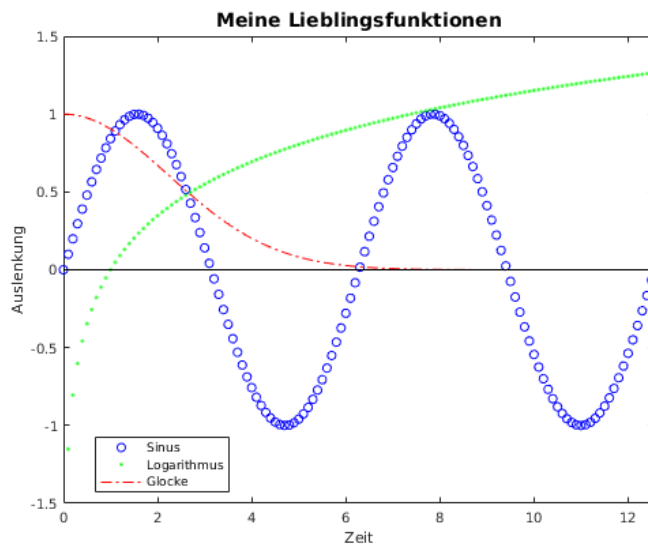
x-Achse als Datensatz mit zwei Punkten

```
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

- 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 $h = \text{plot}(X, Y)$

- 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 Ergebnisse möglich

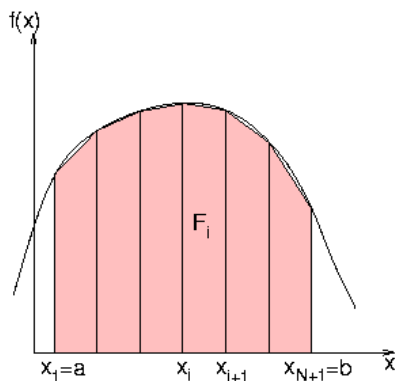
- `[theta, rho] = cart2pol(x, y)`

- 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](#)

- 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

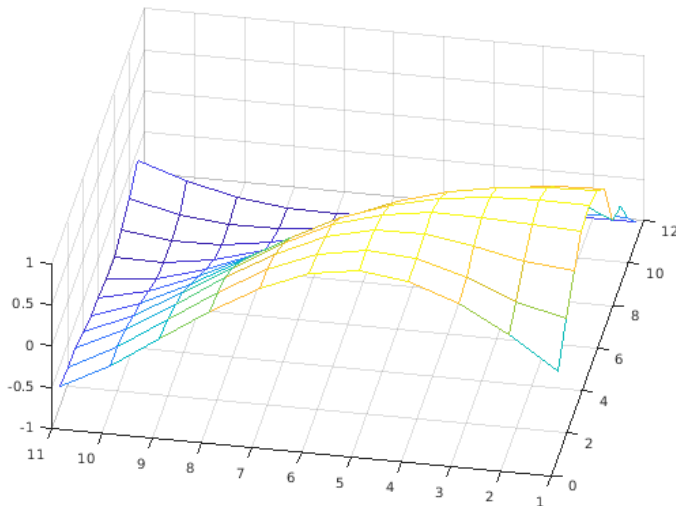
- Darstellung von Matrizen:

Demo-Matrix aus `demo1.dat` laden

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

Darstellung als 3d-Gitter

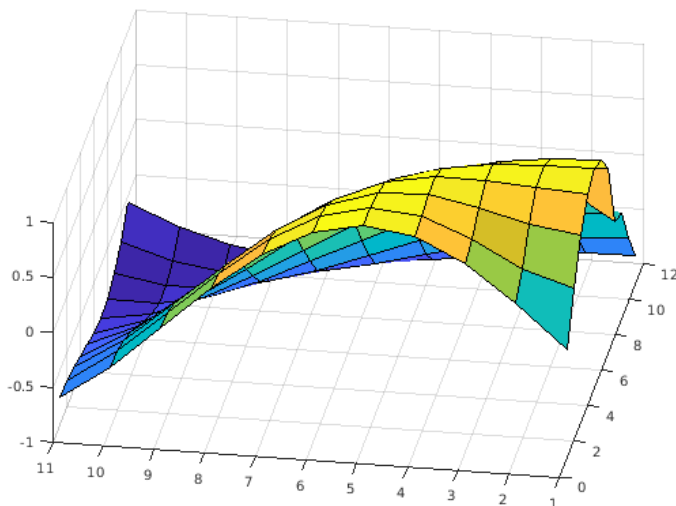
```
mesh(demo1)
```



Änderung der Ansicht durch Lupen- und Drehfunktionen

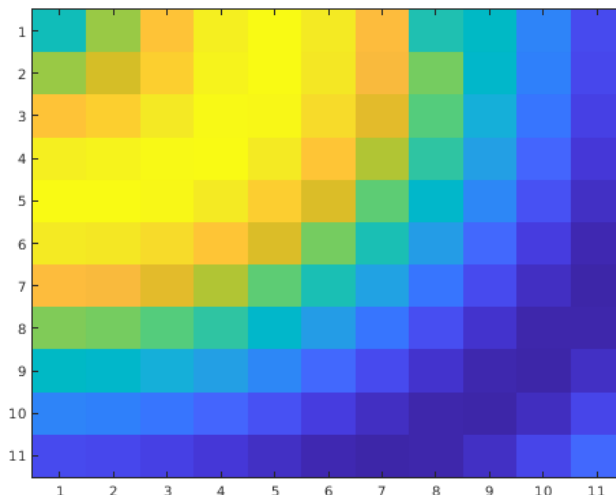
Darstellung als Oberfläche

```
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

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

daher zunächst Hilfsmatrizen X, Y bestimmen

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

- erzeugt Matrizen

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

Funktionswerte als Matrix jetzt einfach mit

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

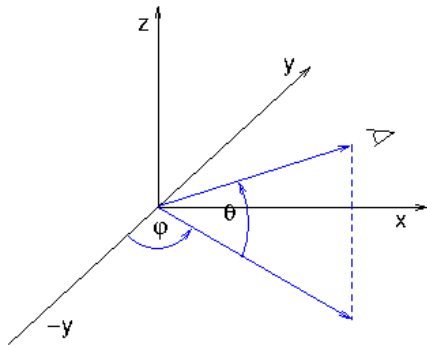
3d-Darstellung z. B. mit

```
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);
```

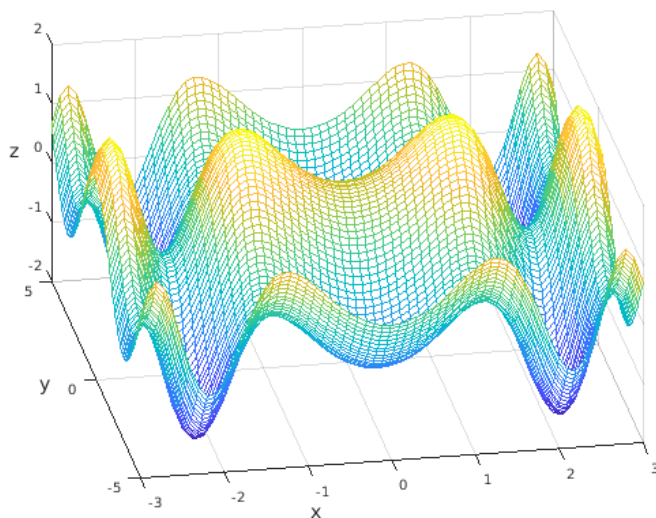
Drehen der Beschriftung der z-Achse

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

Verschiebung der Beschriftung der y-Achse

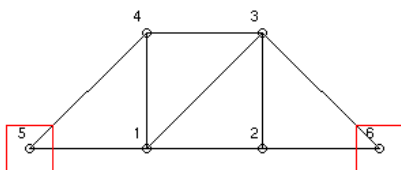
```
hy = get(gca(), "YLabel");
get(hy, "Position") % liefert aktuellen Wert -3.4289 -1.5464 -2.7851
set(hy, "Position", [-10.3 -64 20]);
```

Ergebnis



- Beschreibung eines Fachwerks mit Matrizen:

zweidimensionales Fachwerk mit einigen festen Knoten



physikalische Größen

- Masse m der Knoten (jeweils gleich)

- Federkonstante c der Balken (jeweils gleich)

geometrische Größen

- 2d-Koordinaten der Knoten
- welche Knoten sind fest gelagert?
- zwischen welchen Knoten sitzen Balken?

mögliche Beschreibungsweise

- M Knoten insgesamt, davon N bewegliche
- Knoten so numerieren, dass bewegliche zuerst kommen
- Koordinaten der Knoten als $2 \times M$ -Matrix

$$x_0 = \begin{pmatrix} x_1 & x_2 & \dots & x_M \\ y_1 & y_2 & \dots & y_M \end{pmatrix}$$

- symmetrische Verbindungsmatrix A mit

$$a_{ij} = \begin{cases} 1 & \text{falls Knoten } i \text{ und Knoten } j \text{ verbunden} \\ 0 & \text{sonst} \end{cases}$$

Beispiel in Matlab

```
N = 4;
m = 1;
c = 1;
x0 = [1 2 2 1 0 3; 0 0 1 1 0 0];
A = [0 1 1 1 1 0; 1 0 1 0 0 1; 1 1 0 1 0 1; ...
     1 0 1 0 1 0; 1 0 0 1 0 0; 0 1 1 0 0 0];
```

- Aufgaben:

Aufgabe 5

Aufgabe 6

- 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

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

- 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 GHz-PC

Probe

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

```
error =
```

```
2.0761e-13
```

- Überbestimmte Systeme:

mehr Gleichungen als Unbekannte

i.a. keine Lösung

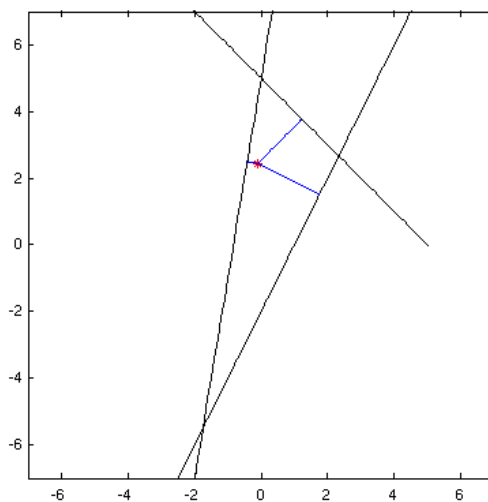
gesucht: Lösung mit geringstem Fehler

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

$$y = 6x + 5$$

$$y = -x + 5$$

$$y = 2x - 2$$

- geschrieben als lineares System

$$6x - y = -5$$

$$-x - y = -5$$

$$2x - y = 2$$

- also in Matlab

```
A = [6 -1; -1 -1 ; 2 -1];
```

```
r = [-5 -5 2]';
```

```
x0 = A \ r
```

- Ergebnis: $x_0 = [-0.0946 \ 2.4459]'$

- 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
```

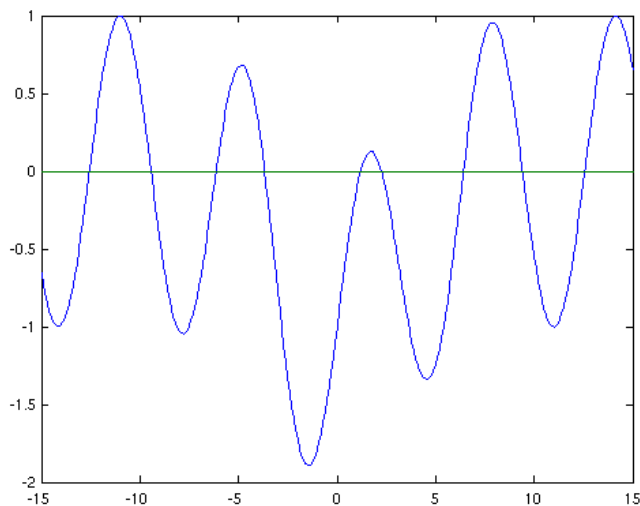
andere Startwerte liefern andere Lösungen

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 $F(x_0)$ und $F(x_1)$ verschieden
- genau eine Lösung im Intervall \rightarrow `fzero` findet sie
- mehrere Lösungen im Intervall \rightarrow `fzero` findet eine

• Aufgaben:

Aufgabe 7

- Ein- und Ausgabe
- Kontrollstrukturen
- Datentypen

- Ausgabe ins Kommandofenster mit `fprintf`:

einfacher Text

```
fprintf("Hallo Peter")
```

erzeugt keine neue Zeile, besser mit zusätzlichem Newline-Zeichen

```
fprintf("Hallo Peter\n")
```

Ausgabe von Variablen durch Typbezeichner (**Conversion character**)

```
fprintf("Zwei mal Drei sind %d\n", 2*3)
fprintf("pi = %f, und zum Quadrat: %f\n", pi, pi^2)
```

einige wichtige Bezeichner

Typzeichner	Verwendung
d	ganze Zahl
x	ganze Zahl, hexadezimal
f	reelle Zahl mit Dezimalpunkt
e	reelle Zahl in Exponential-Schreibweise
g	das kürzere von e oder f
s	Zeichenkette
%	Prozentzeichen

gewünschte Stellen mit `%n.mf` bzw. `%nd`

- m = Zahl der Nachkommastellen
- n = Zahl der Zeichen (incl. Vorzeichen und Dezimalpunkt!)

```
fprintf("%5d\n%5d\n%5d\n%5d\n", 2^5, 2^10, 2^15, 2^20)
fprintf("pi = %12.8f\n", pi)
```

Angabe von Vektoren oder Matrizen → Formatstring wird wiederholt angewandt

```
v1 = 1:10; fprintf("%f, %f\n", v1);
v2 = 1:2:10; fprintf("%f, %f\n", v2);
```

Reihenfolge der Elemente bei Matrizen spaltenweise!

```
A = [1, 2, 3; 4, 5, 6]
fprintf("%4.2f %4.2f\n", A)
fprintf("%4.2f %4.2f %4.2f\n", A')
```

- Eingabe vom Kommandofenster mit `input`:

Beispiel

```
>> t = input("Zahl zwischen 1 und 10 eingeben: ")
Zahl zwischen 1 und 10 eingeben: 7
t = 7
```

Zeichenketten in der Eingabe in `".."` einschließen

```
>> s = input("Wie heißt Du? ")
Wie heißt Du? "Peter"
s = "Peter"
```

- Arbeiten mit Dateien:

grundsätzliche Vorgehensweise

- Datei öffnen
- Datei lesen oder schreiben
- Datei schließen

Datei öffnen

- zum Lesen: `fid = fopen(filename, "r");`
- zum Schreiben: `fid = fopen(filename, "w");`
- zum Lesen und Schreiben: `fid = fopen(filename, "r+");`

Filezeiger `fid` bei Lese-/Schreiboperationen angeben

in geöffnete Datei schreiben

```
fprintf(fid, Formatstring, Variablen)
```

am Ende Datei schließen

```
fclose(fid)
```

- Lesen aus einer Datei:

nächste Zeile (bis zum Newline) aus der Datei holen

```
line = fgetl(fid);
```

- `line` enthält das Newline-Zeichen am Ende nicht!

Daten einzeln lesen

```
mass = fscanf(fid, "%f");
```

- Formatstring wie in `fprintf`

Vektor einlesen

```
dims = fscanf(fid, "%d %d\n", 2);
```

- holt zwei ganze Zahlen aus einer (kompletten) Zeile und speichert in Vektor `dims` der Länge 2

Array einlesen

```
x0 = fscanf(fid, "%f", [2,6]);
```

- holt 12 Fließkommazahlen und speichert sie in Array `x0` mit 2 Zeilen und 6 Spalten

- Speichern und Laden eines Fachwerks:

Fachwerkdaten speichern

- als ASCII-Daten in `NAME.truss`
- mit zusätzlichen Infos in der Datei
- Beispiel-Fachwerk in `bruecke.truss`

Routine `saveTruss1` zum Abspeichern

Routine `loadTruss1` zum Laden

- Aufgaben:

Aufgabe 8

- Bedingte Anweisung:

ermöglicht unterschiedliche Kommandos in Abhängigkeit von Bedingungen

Syntax in Matlab

```
if Bedingung
    mache etwas
else
    mache etwas anderes
end
```

Bedeutung

- Bedingung ist erfüllt (Wert `true` bzw. `1`) → `mache etwas` wird ausgeführt
- Bedingung ist nicht erfüllt (Wert `false` bzw. `0`) → `mache etwas anderes` wird ausgeführt

`else`-Teil kann entfallen ("mache nichts")

Bedingung enthält häufig Vergleichsoperatoren

- `<`, `<=`, `>`, `>=`, `==` (**gleich**), `~=` (**ungleich**)
- `!=` (Zuweisung) und `==` (Gleichheit) verwechseln!

komplexe Bedingung durch logische Verknüpfungen `&&` (**und**), `||` (**oder**), `~` (**nicht**)

- Abweisende Schleife:

Syntax

```
while Bedingung
    mache etwas
end
```

wird solange ausgeführt, bis Bedingung nicht mehr erfüllt ist

Bedingung schon zu Beginn falsch → Schleife wird übersprungen

- Beispiel Eingabeanforderung:

Problemstellung

- Benutzer soll eine Zahl zwischen 1 und 100 eingeben
- bei falscher Eingabe wird immer wieder gefragt

mit Funktion `getNumber`

Erläuterungen

- `sprintf` funktioniert wie `fprintf`, gibt aber einen String zurück
- `isnumeric(n)` liefert `true`, wenn `n` eine Zahl ist, sonst `false`
- in Bedingung `A || B` wird `B` nicht geprüft, wenn `A` schon wahr ist (**short circuit logic**)

- Zählschleife:

Syntax

```
for VARIABLE = WERTE
    mache etwas (verwende dabei VARIABLE)
end
```

`VARIABLE` \triangleq Laufindex

- typische Namen: `I`, `J`, `K`
- *nicht* `i`, `j` (in Matlab komplexe Einheit!)

`WERTE` \triangleq Werte, die der Reihe nach durchlaufen werden

- typisch `I = 1:10`, `K = 5:-0.5:-5`

- aber auch beliebig $I = [3, -1, 0.3]$

in Matlab Schleifen häufig durch (schnellere) Matrixoperationen ersetzt

- Beispiel Fakultät:

berechne $n! = 1 * 2 * 3 * \dots * n$

mit Schleife

```
n = 69;
nFac = 1;
for I = 1:n
    nFac = nFac*I;
end
nFac
```

mit Vektor

```
fac = cumprod(1:n);
nFac = fac(n)
```

mit vorhandener Matlab-Funktion

```
nFac = factorial(n)
```

- Plotten eines Fachwerks:

mit Funktion `plotTruss1`

Erläuterungen

- `hold("on")` zeichnet neue Plots in den vorhandenen Plot
- `hold("off")` stellt Normalzustand wieder her → neuer Plot überschreibt alten
- `gca` = aktuelles Achsensystem
- Eigenschaften `XTick/YTick` definieren Teilstriche auf der x-/y-Achse als Vektor (z. B. `[0:0.5:3]`)
- Doppelschleife für Verbindungen nur über untere Dreiecksmatrix von A
- Datei enthält interne Hilfsfunktion `plotSquare`
- Quadrat sieht auch quadratisch aus mit `axis("equal")`

Datentypen



- Allgemein:

beschreiben die Menge möglicher Werte

legen die möglichen Operationen fest

geben häufig die Speichergröße vor

- Datentypen in Matlab:

alle Daten stehen in Arrays (ggf. 1x1-Arrays)

Elemente eines Arrays haben jeweils identischen Datentyp

einfache Datentypen beschreiben einen Wert

Datentyp	Bedeutung	Beispiel
double	Fließkommazahl mit 64 Bit (\approx 15 Dezimalstellen)	3.14, 6.023e23
char	einzelnes Zeichen	'a', '?'
logical	Wahrheitswert	true, false
int32	ganze Zahl mit 32 Bit	int32(-1), int32(2)^31 - 1
uint8	positive ganze Zahl mit 8 Bit	uint8(0), uint8(1), uint8(255)

Standard-Datentyp: double

höhere Datentypen fassen mehrere Werte zusammen

Datentyp	Bedeutung
Funktionszeiger	beinhaltet Information zur Verwendung einer Funktion
Strukturen	s.u.
Cell-Arrays	s.u.
User-Klassen	vom Benutzer definierte Klassen
Java-Klassen	in Java definierte Klassen

- Zeichenketten:

Datentyp `string` = spezielle Klasse

Stringkonstanten mit "

- ein String: `name = "Peter Junglas"`
- String-Array: `band = ["Peter", "Paul", "Mary"]`

nicht zu verwechseln mit Character-Array

- `name1 = 'Peter'`
- Vektor aus 5 char's

Zusammenhängen mit + (**concatenation**)

- `name2 = "Peter" + " " + "Junglas"`

viele Operationen mit Strings, z.B. `strlength`, `strfind`, `strtrim`

Umwandeln von Strings in Zahlen mit double

- `double(["23", "3.14"]) % -> 23.0000 3.1400`
- nicht verwechseln mit Character-Array
- `double('23') % -> 50 51`

- Cell-Arrays:

Arrays, deren Elemente verschiedenen Typ haben können

Erzeugen mit

- `zelle = cell(2,3)`
- `zelle` hat leere Arrays (`[]`) als Elemente

Zugriff mit {} statt mit ()

```
zelle{1,1} = 3.0
zelle{1,2} = "juhu"
zelle{1,3} = [1,3,5]
zelle{2,1} = zelle{1,1} - 4.0
zelle{2,2} = "blubber"
zelle{2,3} = zelle{1,2} + zelle{2,2}
```

- **Strukturen:**

Container für verschiedenartige Daten (**Datenfelder**) zu einem Objekt

Beispiel Person

Datenfeld	Datentyp	Beispiel
name	string	"Simpson"
vorname	string	"Homer"
alter	double	39
kinder	string array	["Bart", "Lisa", "Maggie"]

Zugriff auf die Elemente durch VARIABLE.FELD

schrittweise Erzeugung einer Struktur

```
p1.name = "Simpson"
p1.vorname = "Homer"
p1.alter = 39
p1.kinder = ["Bart", "Lisa", "Maggie"]
```

komplette Erzeugung einer Struktur

```
p2 = struct("name", "Simpson", "vorname", "Marge", ...
           "alter", 34, "kinder", ["Bart", "Lisa", "Maggie"])
```

Erzeugung eines Arrays von Strukturen

```
pkids = struct("name", "Simpson", "vorname", {"Bart", "Lisa", "Maggie"}, ...
              "alter", {10, 8, 1}, "kinder", [])
```

- **Beispiel Fachwerk:**

Daten zur Beschreibung eines Fachwerks in einer Struktur zusammenfassen

Erzeugen eines Beispiel-Fachwerks mit `createDemoTruss`

Funktionen anpassen: `loadTruss`, `saveTruss`, `plotTruss`

verwenden einfach mit

```
truss = createDemoTruss();
plotTruss(truss);
saveTruss(truss, "demo");
```

- **Aufgaben:**

[Aufgabe 9](#)

[Aufgabe 10](#)

- Grundlegende Beispiele
- Mehrdimensionale Schwingungen

- 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"

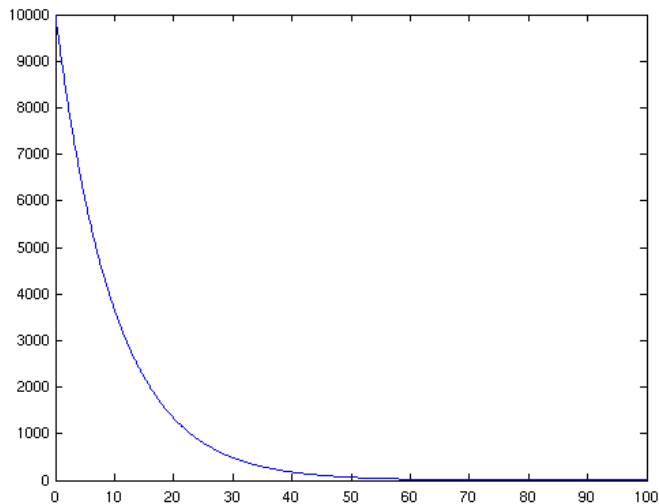
$$\begin{aligned}\frac{N(t+h) - N(t)}{h} &= -\lambda N(t) \\ \Rightarrow N(t+h) &= (1 - \lambda h)N(t)\end{aligned}$$

- 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]);
```

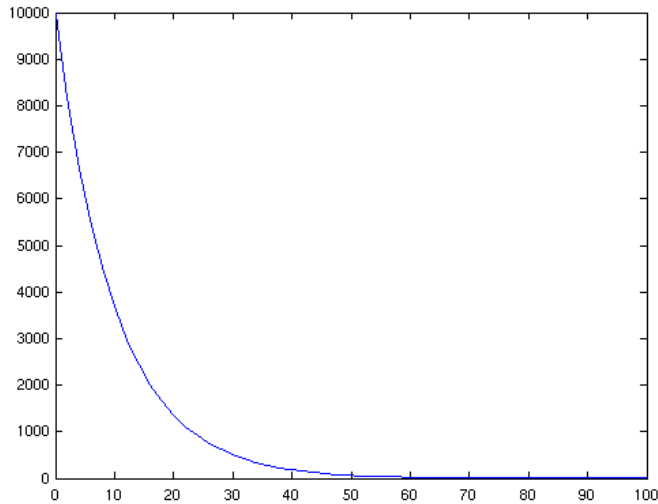
Parameter

- 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 $y' = f(t, y)$

$$y_1' = -\lambda_1 y_1$$

$$y_2' = \lambda_1 y_1 - \lambda_2 y_2$$

rechte Seiten als Matlab-Funktion [radiokette.m](#)

Anfangsbedingungen z.B.

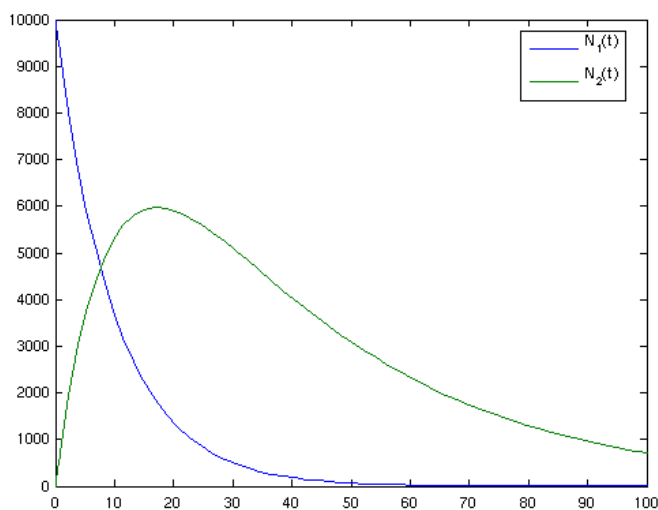
- 10000 Teilchen von Sorte 1
- keine Teilchen von Sorte 2

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

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} := \begin{pmatrix} x \\ \dot{x} \end{pmatrix}$$

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

$$= \begin{pmatrix} y_2 \\ -\frac{b}{m}y_2 - \frac{c}{m}y_1 + \frac{A}{m}\cos(\omega t) \end{pmatrix}$$

rechte Seiten als Matlab-Funktion `erzwungen.m`

Anfangsbedingungen z.B. Masse in Ruhe am Ursprung

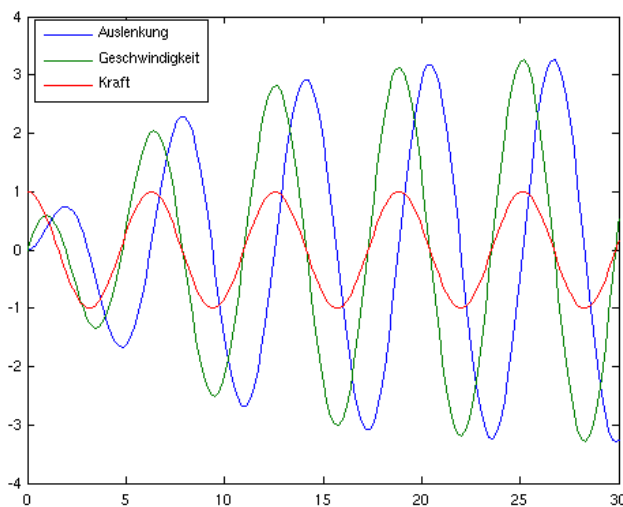
Lösen

```
[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", "best");
```



- Übergabe von Parametern:

Problem:

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

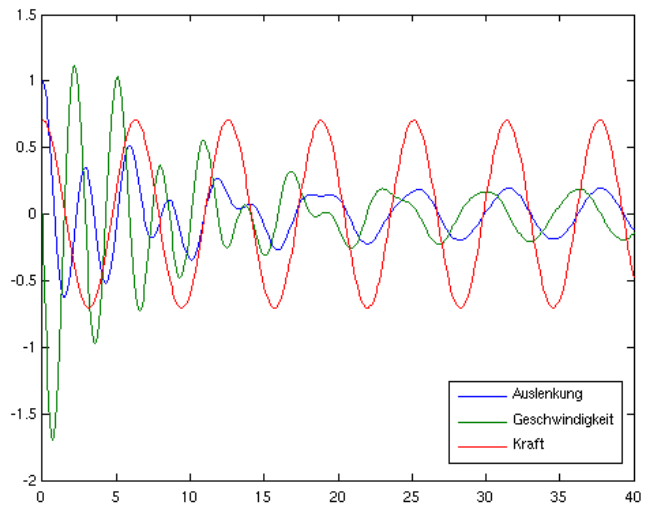
Parameter als zusätzliche Argumente der Funktion (`erzwungenp.m`)

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]);
```

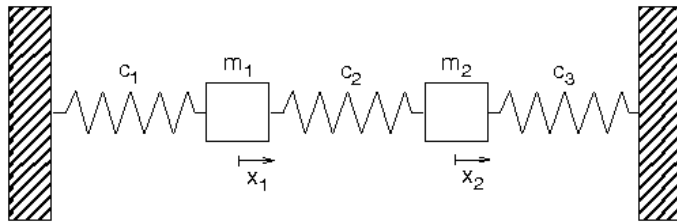


- Aufgaben:

Aufgabe 11

- Zweidimensionale Schwingerkette:

2 Massen, mit Federn gekoppelt



Bewegungsgleichung

$$m_1 \ddot{x}_1 + (c_1 + c_2)x_1 - c_2 x_2 = 0$$

$$m_2 \ddot{x}_2 - c_2 x_1 + (c_2 + c_3)x_2 = 0$$

vgl. Applet Coupled Oscillators



Bewegungsgleichung nach höchsten Ableitungen auflösen

$$\ddot{x}_1 = \frac{1}{m_1} (-(c_1 + c_2)x_1 + c_2 x_2)$$

$$\ddot{x}_2 = \frac{1}{m_2} (c_2 x_1 - (c_2 + c_3)x_2)$$

in Grundform bringen

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix} := \begin{pmatrix} x_1 \\ x_2 \\ \dot{x}_1 \\ \dot{x}_2 \end{pmatrix}$$

$$\Rightarrow \mathbf{y}' = \begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \ddot{x}_1 \\ \ddot{x}_2 \end{pmatrix} = \begin{pmatrix} y_3 \\ y_4 \\ \frac{1}{m_1} (-(c_1 + c_2)y_1 + c_2 y_2) \\ \frac{1}{m_2} (c_2 y_1 - (c_2 + c_3)y_2) \end{pmatrix}$$

rechte Seiten als Matlab-Funktion [schwing2d.m](#)

Anfangsbedingung: Massen in Ruhe, nur 1. ausgelenkt

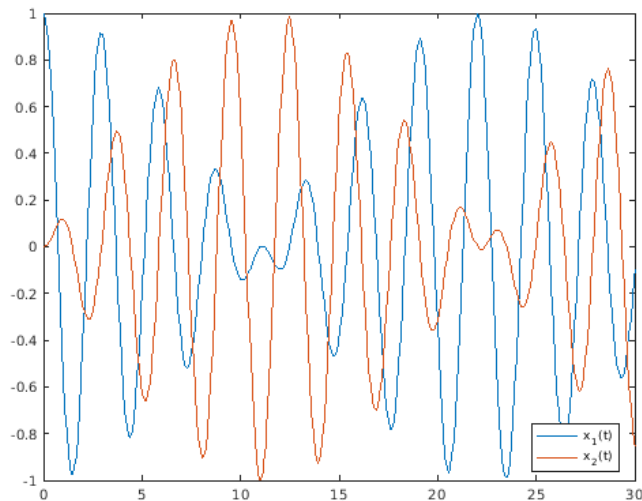
Lösen

```
[t, y] = ode45(@schwing2d, [0 30], [1 0 0 0]);
```

Plot der Bewegungen

- 1./2. Spalte = x_1/x_2
- 3./4. Spalte = v_1/v_2

```
plot(t, y(:,1), t, y(:,2))
legend("x_1(t)", "x_2(t)", "Location", "best");
```



- Matrixform der Schwingungsgleichung:

Bewegungsgleichung umschreiben

$$\begin{pmatrix} m_1 & 0 \\ 0 & m_2 \end{pmatrix} \begin{pmatrix} \ddot{x}_1 \\ \ddot{x}_2 \end{pmatrix} + \begin{pmatrix} c_1 + c_2 & -c_2 \\ -c_2 & c_2 + c_3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = 0$$

Vektoren und Matrizen einführen

$$\mathbf{x} := \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \Rightarrow \dot{\mathbf{x}} = \begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} \Rightarrow \ddot{\mathbf{x}} = \begin{pmatrix} \ddot{x}_1 \\ \ddot{x}_2 \end{pmatrix}$$

$$\mathbf{M} := \begin{pmatrix} m_1 & 0 \\ 0 & m_2 \end{pmatrix}, \quad \mathbf{C} = \begin{pmatrix} c_1 + c_2 & -c_2 \\ -c_2 & c_2 + c_3 \end{pmatrix}$$

Bewegungsgleichung lautet dann

$$\mathbf{M}\ddot{\mathbf{x}} + \mathbf{C}\mathbf{x} = 0$$

M heißt **Massenmatrix**, C **Steifigkeitsmatrix**

für die Grundform \mathbf{y} in Zweivektoren zerlegen

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$$

$$\mathbf{v} = \begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \begin{pmatrix} y_3 \\ y_4 \end{pmatrix}$$

$$\mathbf{f}(t, \mathbf{y}) = \begin{pmatrix} \mathbf{v} \\ -\mathbf{M}^{-1}\mathbf{C}\mathbf{x} \end{pmatrix}$$

rechte Seiten mit Parametern M, C, als Matlab-Funktion [matschwing2d.m](#)

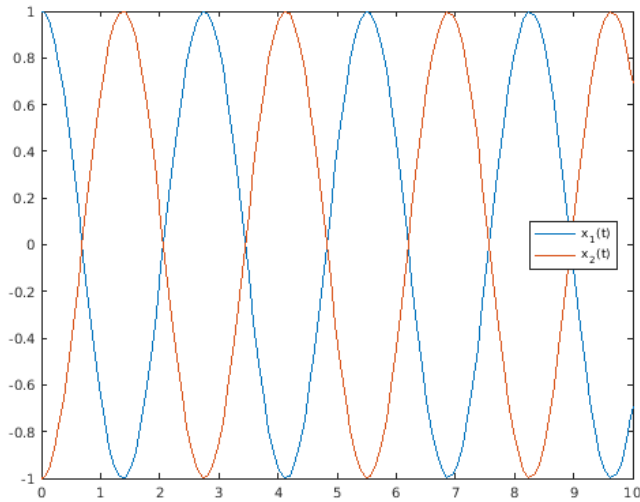
Matrizen M, C und Hilfsfunktion definieren

```
M = [m1, 0; 0, m2];
C = [c1+c2, -c2; -c2, c2+c3];
fhilf = @(t,y) matschwing2d(t, y, M, C);
```

Anfangsbedingung: Massen in Ruhe, 1. und 2. gegeneinander ausgelenkt

Lösen

```
[t, y] = ode45(fhilf, [0 30], [1 -1 0 0]);
```

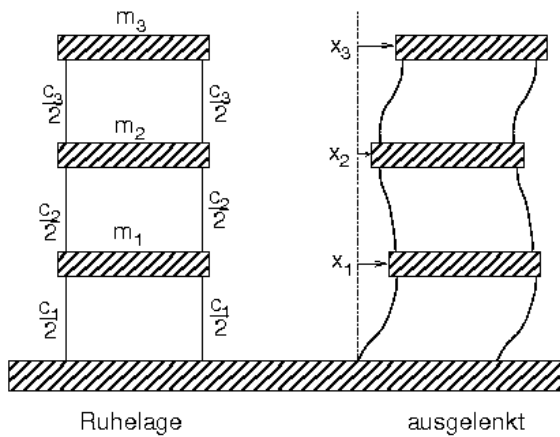


- N-dimensionale Schwingungen:

mehrere Massen und/oder Bewegung in mehr als einer Dimension

Schwingungsgleichung in Matrixform bleibt gültig

Beispiel "Schwingendes Hochhaus"

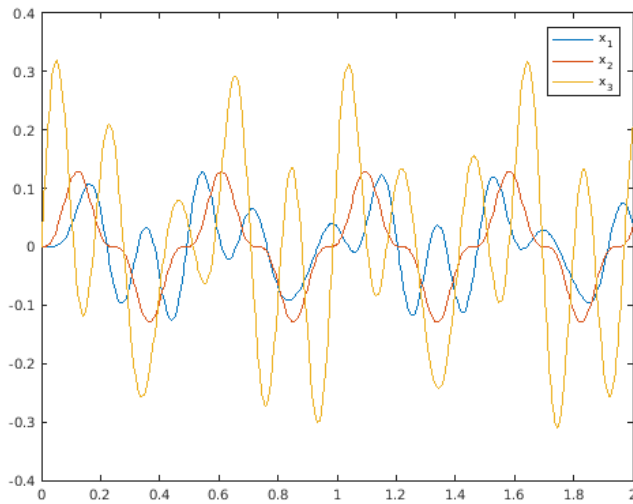


- Matrizen

$$M = \begin{pmatrix} m_1 & 0 & 0 \\ 0 & m_2 & 0 \\ 0 & 0 & m_3 \end{pmatrix}, \quad C = \begin{pmatrix} c_1 + c_2 & -c_2 & 0 \\ -c_2 & c_2 + c_3 & -c_3 \\ 0 & -c_3 & c_3 \end{pmatrix}$$

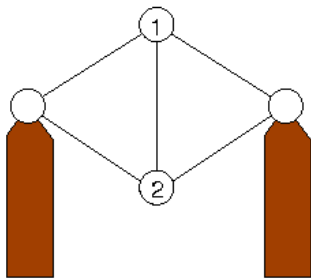
Matlab-Funktion `schwingNd.m` zerlegt Vektor y in zwei Teile

Lösen mit Anfangsgeschwindigkeit für v_3 mit Matlabskript `bild26.m`



- Bewegung in zwei Dimensionen:

Beispiel



Verschiebungen \mathbf{d}_i der Massen: 2 Vektoren mit jeweils 2 Komponenten

$$\mathbf{d}_1 = \begin{pmatrix} d_{1x} \\ d_{1y} \end{pmatrix}, \quad \mathbf{d}_2 = \begin{pmatrix} d_{2x} \\ d_{2y} \end{pmatrix}$$

Bewegungsgleichungen

$$\begin{aligned} m\ddot{d}_{1x} + c d_{1x} &= 0 \\ m\ddot{d}_{1y} + c(2d_{1y} - d_{2y}) &= 0 \\ m\ddot{d}_{2x} + c d_{2x} &= 0 \\ m\ddot{d}_{2y} + c(2d_{2y} - d_{1y}) &= 0 \end{aligned}$$

Reihenfolge der Koordinaten festlegen

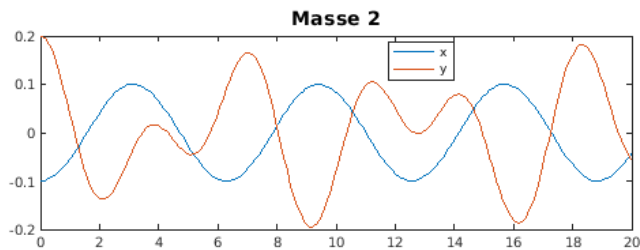
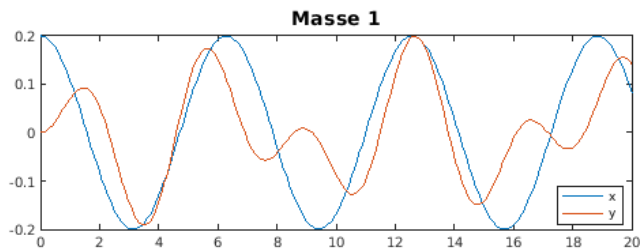
$$\mathbf{x} = \begin{pmatrix} d_{1x} \\ d_{1y} \\ d_{2x} \\ d_{2y} \end{pmatrix}$$

dann kann man die Matrizen ablesen

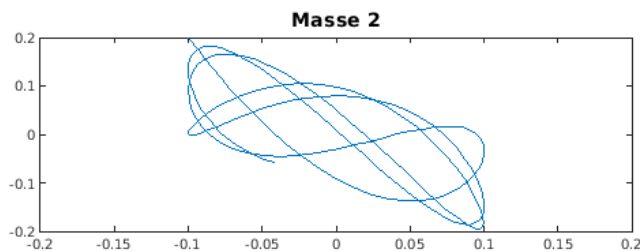
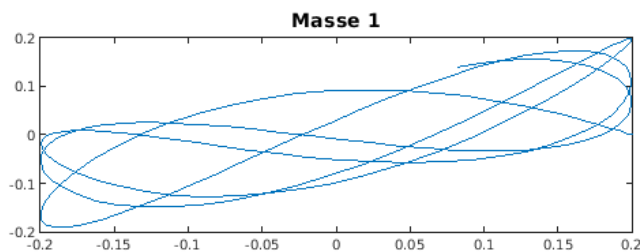
$$\mathbf{M} = m \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad \mathbf{C} = c \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & -1 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 2 \end{pmatrix}$$

Lösen und plotten mit Matlabskript [bild28.m](#)

- x- und y-Koordinaten



■ Bahnkurven



Bewegung schwer nachzuvollziehen, besser mit Animation

● Schwingungen eines Fachwerks:

Aufstellen der Bewegungsgleichungen

- mühsam (vgl. [fachwerk.pdf](#))
- Ergebnis

$$m\ddot{\vec{d}}_k + c \sum_{i=1}^M a_{ki} (\vec{e}_{ki}^0 \cdot (\vec{d}_k - \vec{d}_i)) \cdot \vec{e}_{ki}^0 = 0 \quad k = 1, \dots, N$$

für die Abweichungen von der Gleichgewichtslage

$$\vec{d}_k := \vec{x}_k - \vec{x}_k^0$$

mit den Richtungsvektoren

$$\vec{e}_{ij}^0 := \frac{\vec{x}_i^0 - \vec{x}_j^0}{|\vec{x}_i^0 - \vec{x}_j^0|}, \quad i, j = 1, \dots, M$$

Ermitteln der Steifigkeitsmatrix

- erledigt Routine `createMatrices`
- Reihenfolge der Koordinaten $x_1, y_1, x_2, y_2, \dots$

bequemes Lösen der DGL mit `solveVibrationODE`

- Anfangsauslenkungen d_0 als $N \cdot 2d$ -Spaltenvektoren in einer Matrix

- Anfangsgeschwindigkeiten v_0 ebenso
- Ergebnisse d, v als $(N \times 2 \times N)$ -Tensor

Umsortieren mit `reshape(A, N, M)`

- ordnet erst alle Spalten von A untereinander zu langem Vektor
- teilt dann neu auf in $N \times M$ -Matrix

Funktionen mit optionalen Parametern

- hintere Argumente können weggelassen werden
- Variable `nargin` enthält Zahl der übergebenen Parameter
- Funktion prüft `nargin` und definiert ggf. Standardwerte

- Aufgaben:

Aufgabe 12

Aufgabe 13

Berechnung von Eigenschwingungen

- Zweidimensionale Schwingerkette:

experimentiere mit verschiedenen Anfangsauslenkungen



i.a. komplexes Verhalten, ähnlich Schwebung

spezielle Anfangsbedingung

$$\mathbf{x}(0) = a \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

- liefert einfache Schwingung mit Schwingungsdauer

$$T_1 = 3.14 \text{ s}$$

spezielle Anfangsbedingung

$$\mathbf{x}(0) = a \begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

- liefert einfache Schwingung mit Schwingungsdauer

$$T_1 = 2.76 \text{ s}$$

diese speziellen Schwingungen heißen **Eigenschwingungen**

ihre Frequenzen sind die **Eigenfrequenzen** f_i (in aufsteigender Reihenfolge)

$$f_1 = 0.3183 \text{ Hz}, f_2 = 0.3629 \text{ Hz}$$

die zugehörigen Anfangsauslenkungen (meistens auf Länge 1 normiert) heißen **Eigenvektoren** \mathbf{x}_i

$$\mathbf{x}_1 = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad \mathbf{x}_2 = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

- Dreidimensionale Schwingerkette:

experimentiere mit verschiedenen Anfangsauslenkungen



i.a. komplexes Verhalten

Eigenvektoren (bei Standardwerten für m_i und c_i)

$$\mathbf{x}_1 = \begin{pmatrix} 0.5000 \\ 0.7071 \\ 0.5000 \end{pmatrix}, \quad \mathbf{x}_2 = \begin{pmatrix} 0.7071 \\ 0.0000 \\ -0.7071 \end{pmatrix}, \quad \mathbf{x}_3 = \begin{pmatrix} 0.5000 \\ -0.7071 \\ 0.5000 \end{pmatrix}$$

Eigenfrequenzen

$$f_1 = 0.1218 \text{ Hz}, f_2 = 0.2251 \text{ Hz}, f_3 = 0.2941 \text{ Hz}$$

- Berechnung von Eigenschwingungen:

per Hand recht mühsam, Stoff der Vorlesung "Schwingungslehre"

in Matlab einfach mit

$$[\text{Phi}, \text{om2}] = \text{eig}(C, M)$$

Matrix Phi enthält die Eigenvektoren der Reihe nach als Spalten (unnormiert)

Diagonalmatrix om2 enthält die Quadrate der Kreisfrequenzen

$$\omega_i = 2 \pi f_i$$

3d-Beispiel mit doppelter Masse in der Mitte

```
M = diag([1,2,1]);
C = [2, -1, 0; -1, 2, -1; 0, -1, 2];
[Phi, om2] = eig(C,M)
```

liefert

```
Phi =
    0.3717 -0.7071 -0.6015
    0.6015  0.0000  0.3717
    0.3717  0.7071 -0.6015
```

```
om2 =
    0.3820  0  0
    0  2.0000  0
    0  0  2.6180
```

Ermittlung der Frequenzen

```
freq = sqrt(diag(om2))/(2*pi)
```

Normierung der Eigenvektoren

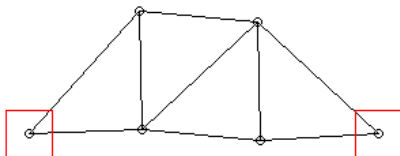
```
norms = diag(Phi'*Phi);
Phi = Phi*diag(1./sqrt(norms))
```

zusammengefasst in Funktion `computeEigenvalues`

- Eigenschwingungen des Beispiel-Fachwerks:

8 Koordinaten (4 Knoten in 2 Dimensionen) → 8 Eigenschwingungen

zweite Eigenschwingung



Berechnung der zweiten Eigenschwingung

```
truss = createDemoTruss();
[M, C] = createMatrices(truss);
[Phi, freq] = computeEigenvalues(M, C);
x2 = Phi(:,2)
```

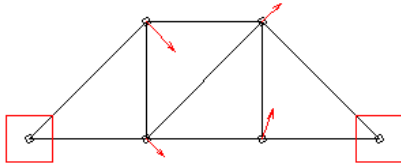
Ergebnis

```
x2 = [-0.2859; 0.2859; -0.1767; -0.4626; -0.2859; -0.2859; -0.4626; 0.4626]
```

- interpretiert als 4 Zweier-Vektoren

$$\mathbf{x}_1 = \begin{pmatrix} 0.2859 \\ -0.2859 \end{pmatrix}, \quad \mathbf{x}_2 = \begin{pmatrix} 0.1767 \\ 0.4626 \end{pmatrix}, \quad \mathbf{x}_3 = \begin{pmatrix} 0.2859 \\ 0.2859 \end{pmatrix}, \quad \mathbf{x}_4 = \begin{pmatrix} 0.4626 \\ -0.4626 \end{pmatrix}$$

- im Bild



Plotten des Eigenvektors mit `plotMode`

Matlab-Funktion `quiver(x,y,u,v)` plottet Vektoren (u,v) an den Stellen (x,y)

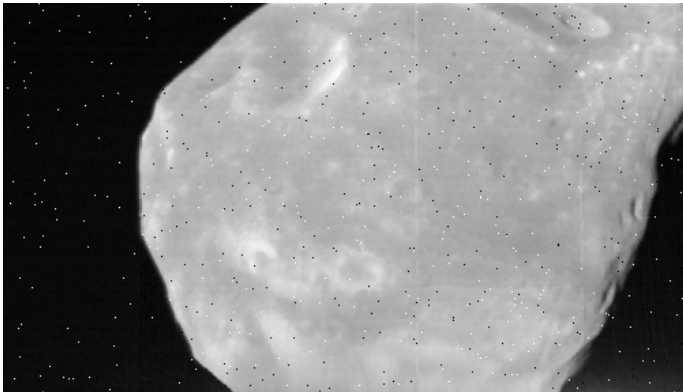
- Aufgaben:

Aufgabe 14

- Bilder
- Animationen
- Erstellen graphischer Oberflächen
- Klänge

- Laden und Darstellen von Bildern:

Beispielbild (im GIF-Format)



Einlesen in Matlab

```
B = imread("../images/bild32.gif");
```

Datentyp von B (mit `whos B`)

- 287x504-Matrix, Elemente vom Typ `uint8`
- Beispielwerte `B(150:155, 250:255)`

```
187 191 189 190 192 197
188 190 192 193 195 195
183 189 190 192 193 194
181 187 188 190 191 192
181 186 187 189 190 191
183 185 186 187 189 190
```

- Zahl \triangleq Nummer in Farbtabelle (hier mit 256 Stufen von schwarz bis weiß)

Darstellung in Matlab

```
image(B)
```

Achsen weg, Skala an Bild-Seitenverhältnis anpassen

```
axis("off")
axis("image")
```

verwendet Standard-Farbtabelle, angezeigt mit

```
colorbar
```

- Farbtabelle:

Zuordnung von Nummern und Farben

- Matrix mit $N \times 3$ Werten
- jeweils Rot-, Grün-, Blauanteil für jede Nummer N
- Werte zwischen 0 und 1

diverse Farbtabellen vordefiniert (mit standardmäßig 256 Einträgen), z.B.

- `colormap("gray")`
- `colormap("hsv")`
- `colormap("hot")`

eigene Grautabelle mit 256 Stufen erzeugen

```
scale = (0:255)'/255;
graymap = [scale, scale, scale];
colormap(graymap)
```

Negativ-Darstellung

```
colormap(1 - graymap)
```

Tabelle mit schnellen Wechseln → macht leichte Helligkeitsvariationen sichtbar

```
scale2 = rem((0:255)'/16, 1);  
scale3 = rem((0:255)'/32, 1);  
map3 = [scale, scale2, scale3];  
colormap(map3)
```

Colormap des GIF-Bildes verwenden

```
[B, map] = imread("../images/bild32.gif");  
image(B)  
colormap(map)
```

• Bild-Bearbeitung mit Filtern:

Bild enthält fehlerhafte Pixel (ganz weiß, ganz schwarz), z.B.

```
A = B(211:215, 352:356)
```

Idee: (Grau-)Werte durch Mittelwerte der umliegenden Punkte ersetzen

$$a_{22} = \frac{1}{9}(a_{11} + a_{12} + a_{13} + a_{21} + a_{22} + a_{23} + a_{31} + a_{32} + a_{33})$$

Problem

- am Bildrand nicht definiert
- Bild um einen Rahmen aus 0-Werten erweitern

```
A0 = zeros(size(A)+2)  
A0(2:end-1, 2:end-1) = A
```

Implementierung umständlich mit einfachen Array-Operationen

```
A1 = (1/9)* ( ...  
    A0(1:end-2, 1:end-2) + A0(2:end-1, 1:end-2) + A0(3:end, 1:end-2) ...  
    + A0(1:end-2, 2:end-1) + A0(2:end-1, 2:end-1) + A0(3:end, 2:end-1) ...  
    + A0(1:end-2, 3:end) + A0(2:end-1, 3:end) + A0(3:end, 3:end) )
```

einfacher mit Matlabs zweidimensionaler Filter-Funktion

```
filterA = ones(3,3)/9  
A2 = filter2(filterA, A)
```

auf Bild anwenden

```
B1 = filter2(filterA, double(B));  
image(B1)
```

Ergebnis nicht optimal

- Fehler sind schwächer, aber ausgedehnt
- Bild insgesamt unschärfer

Filter zur Betonung von Änderungen

```
filterB = (1/8)*[-1,-1,-1; -1,8,-1; -1,-1,-1]  
B2 = filter2(filterB, double(B));  
image(B2)  
colormap(map3)
```

• Median-Filterung:

Nachteil des (arithmetischen) Mittelwerts

- Ausreißer verändern die umgebenden Werte stark

Alternative

```
medianFilter = @(x) median(x,3);
```

- Median = mittlerer Wert einer Zahlenfolge
- Größe eines Extremwerts spielt keine Rolle
- aufwändiger zu berechnen (sortieren!)

Beispiel

```
a = [1 2 3 4 5 6 100]
mean(a)
median(a)
```

- Medianfilter in der "Image Processing Toolbox" ersetzt Bildpunkt durch Median der 3x3-Umgebung

```
B3 = medfilt2(B);
image(B3)
colormap(map)
```

Ergebnis

- alle "defekten Pixel" sind komplett beseitigt
- Bild insgesamt etwas unschärfer

- Selektive Median-Filterung:

Medianfilter nur bei defekten Pixeln verwenden → Bild bleibt scharf

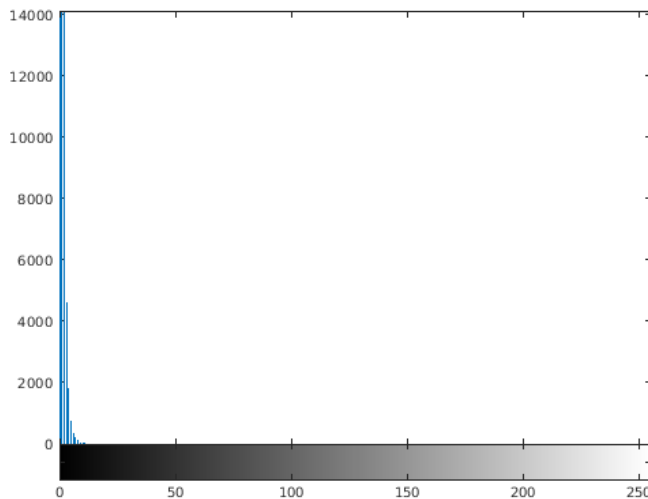
Unterschied zwischen Original und gefiltertem Bild bestimmen

```
diffs = abs(double(B) - double(B3));
```

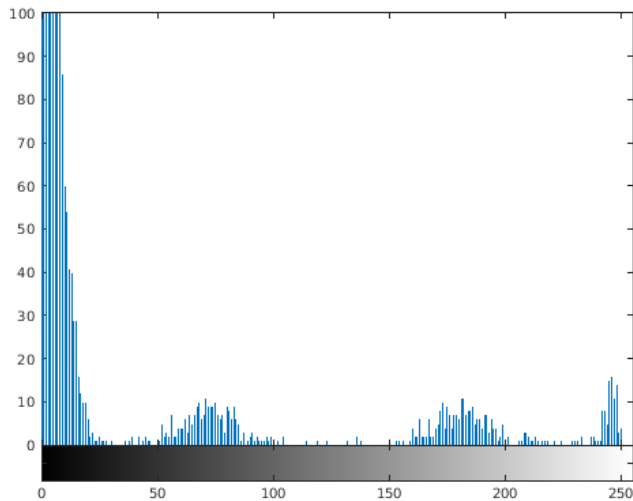
- Vorsicht: *nicht* `abs(B - B3)` wegen Datentyp `uint8`!

Schwelle für "defekte Pixel" durch Histogramm (Verteilung der Werte) bestimmen

```
imhist(uint8(diffs))
```



- Skala der y-Achse ändern mit `ylim([0,100])`



- Schwellwert etwa bei 30 (im Zweifel etwas kleiner)

Index-Werte für Punkte jenseits der Schwelle berechnen

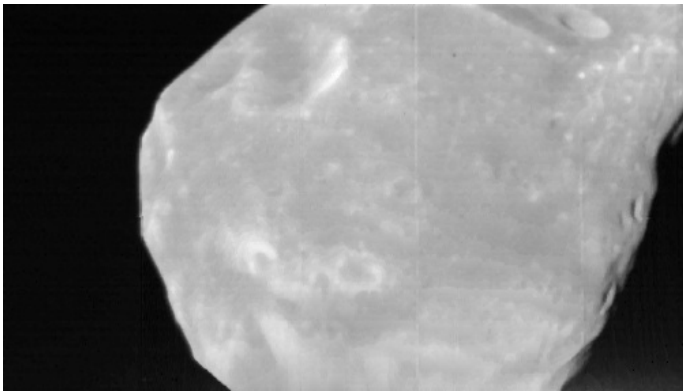
```
badspots = diffs > 30;
indices = find(badspots);
```

- `badspots` ist 1 bei den defekten Pixeln, 0 sonst
- `indices` enthält Nummern (Array spaltenweise durchnummeriert) der defekten Pixel

Bild nur an den defekten Stellen durch Medianwert ersetzen

```
B4 = B;
B4(indices) = B3(indices);
```

Ergebnis



abspeichern mit

```
imwrite(B4, map, "corrected.gif");
```

- Echtfarben-Bilder:

Beispielbild (im JPG-Format)



- kontrastarm und blaustichig!

Einlesen in Matlab und darstellen

```
B = imread("../images/bild35.jpg");  
image(B);  
axis("off")  
axis("image")
```

- verwendet die Echtfarben statt einer Farbtabelle

Datentyp von B

- 427x640x3-Array, Elemente vom Typ `uint8`
- je eine Matrix für Rot-, Grün- und Blau-Anteile

Farbnegativ anzeigen

```
image(255-B)
```

Farbanteile separieren

```
Br = double(B); Br(:, :, [2, 3]) = 0;  
Bg = double(B); Bg(:, :, [1, 3]) = 0;  
Bb = double(B); Bb(:, :, [1, 2]) = 0;
```

- Ergebnis

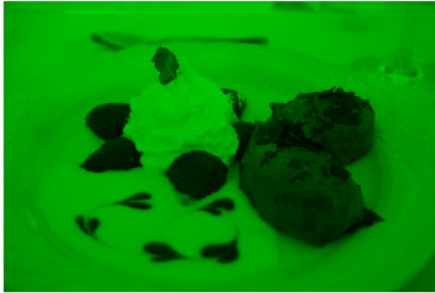
Original



rot



grün



blau



Kontrast des Bildes für jeden Farbkanal einzeln erhöhen

- kleinsten und größten Wert bestimmen

```
lowPix = min(min(min(Br)))  
highPix = max(max(max(Br)))
```

- Werte linear strecken auf [0 255]

```
Brs = 255*(Br - lowPix)/(highPix - lowPix);
```

- analog für Grün- und Blauanteil

Teile zusammenführen und abspeichern

```
Bs = uint8(Brs + Bgs + Bbs);  
imwrite(Bs, "bild37.jpg");
```

Ergebnis



- Animationen in Matlab:

- einfache Methode

- Einzelschritte berechnen und plotten
 - daraus Momentaufnahmen (**Frames**) erzeugen mit

- ```
frame1 = getframe;
```

- `getframe` erzeugt Pixel-Screenshots im Standard-Farbbild-Format (als `uint8`-Array  $w \times h \times 3$ )
    - Frames in einem Array `F` sammeln

- abspielen mit

- ```
movie(F);
```

- achtmal hintereinander mit 32 Frames/s

- ```
movie(F, 8, 32);
```

- andere Vorgehensweisen

- einzelne Objekte löschen und neu zeichnen
    - Werte der Daten innerhalb des Plots ändern

- Animation einer Welle:

- einfache Wellenfunktion  $\Phi(x,t) = \cos(t - x)$

- Bild räumlich über zwei Wellenlängen (also  $x$  von 0 bis  $4\pi$ )

- Animation zeitlich über eine Schwingung (also  $t$  von 0 bis  $2\pi$ )

- Matlab-Code

- ```
t = 0:2*pi/16:2*pi;
x = 0:0.1:4*pi;
for I=1:17
    phi = cos(t(I) - x);
    plot(x, phi)
    F(I) = getframe;
end
movie(F,2)
```

- Animation einer Fachwerk-Eigenschwingung:

- Berechnung der Animation erledigt Funktion `plotModeAnimation1`

- Vorgehen

- berechnet in festen Zeitschritten Positionen der Knoten
 - kopiert dazu `truss` und verschiebt Knoten entlang Eigenvektor
 - plottet verschobenen Truss `trussT`
 - sammelt Plots mit `getframe`

- Anwendung

- ```
truss = createDemoTruss();
[M, C] = createMatrices(truss);
[Phi, freq] = computeEigenvalues(M, C);
x2 = Phi(:,2);
plotTruss(gca, truss)
F = plotModeAnimation1(gca, x2, truss);
movie(gca, F, 4)
```

- Anpassung an Matlab-App

- weder `getframe` noch `movie` in App erlaubt

- daher viermal direkt plotten statt mit `movie`
- fertiges Bild mit `drawnow` sofort anzeigen (sonst fehlen ggf. Zwischenbilder)
- Ergebnis `plotModeAnimation`
- Aufruf jetzt `plotModeAnimation(gca, x2, truss, 4);`

- Aufgaben:

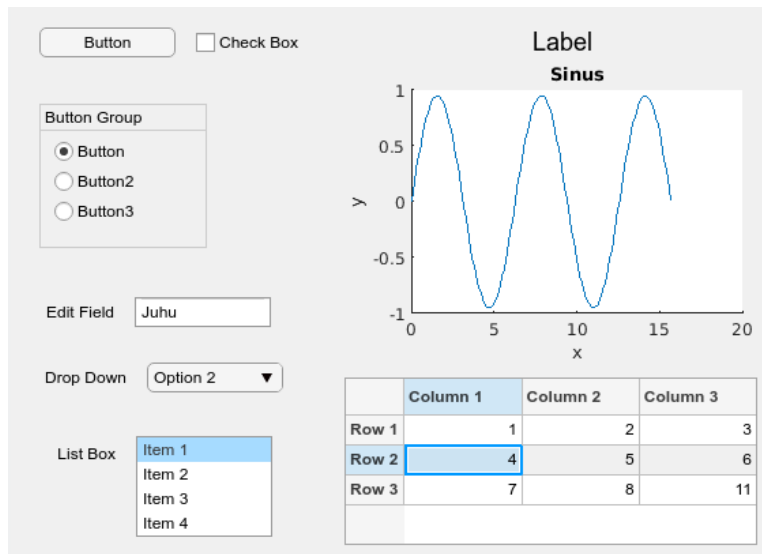
### Aufgabe 15

# 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:

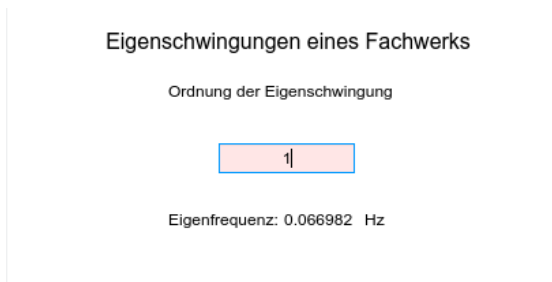
```
truss = loadTruss("bruecke");
[M, C] = createMatrices(truss);
[Phi, freq] = computeEigenvalues(M, C);
plotTruss(axes, truss);
plotMode(axes, Phi(:,1), truss);
F = plotModeAnimation(axes, xe, truss);
```

- Beispielprogramm `trussGUI1`:

### Aufgabenstellung

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

### gewünschte GUI



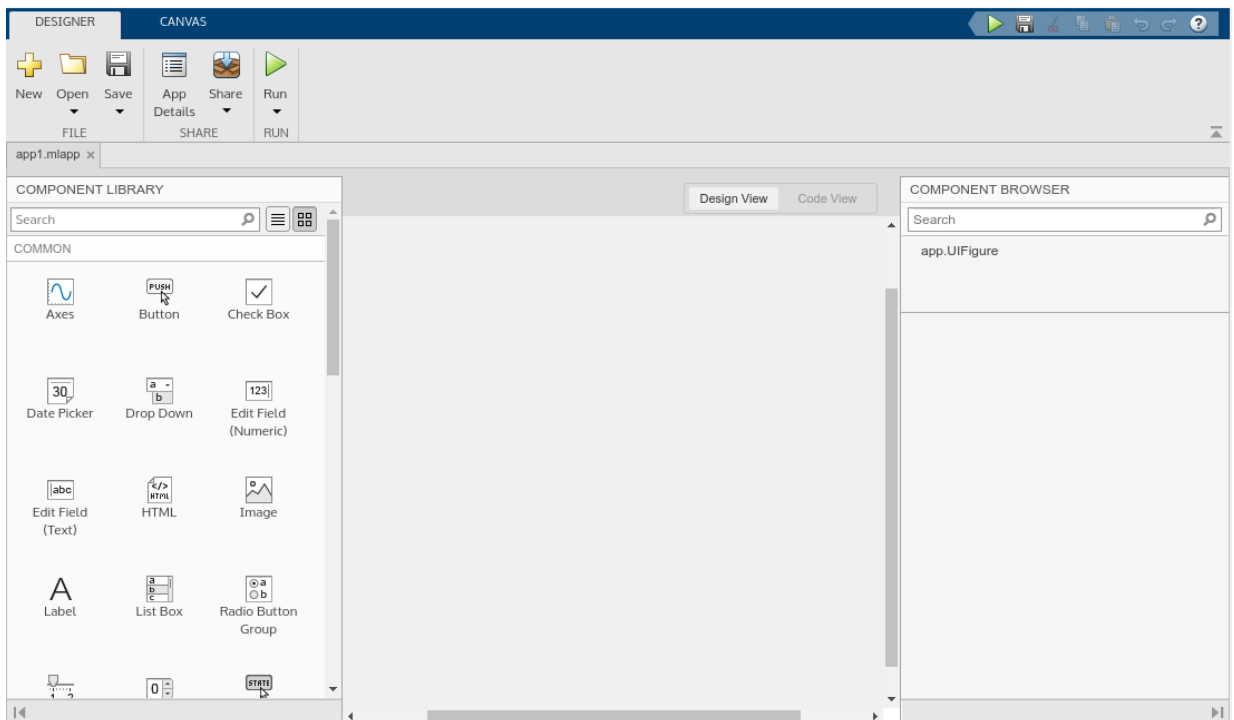
### Erstellen der GUI

- hier mit interaktivem 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`

|                    |                                                  |
|--------------------|--------------------------------------------------|
| <code>app</code>   | Objekt <code>app</code> mit allen UI-Komponenten |
| <code>event</code> | 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 `trussGUI1/Callbacks/Go to 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

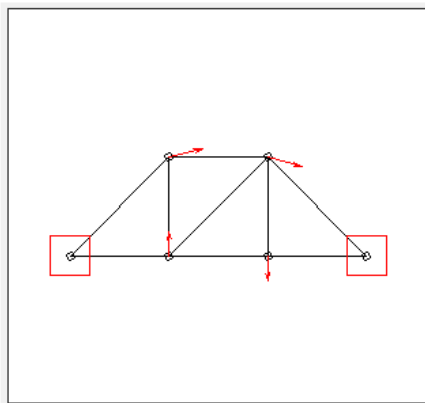
- ersetze `truss` durch `app.Truss`
- ersetze `freq` durch `app.Freq`

komplettes Programm: [trussGUI1.mlapp](#)

- GUI mit Plotbereich:

Ziel: Erweitern von `trussGUI` um Anzeige der Eigenvektoren

Eigenschwingungen eines Fachwerks



Ordnung der Eigenschwingung

3

Eigenfrequenz: 0.126723 Hz

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

## interaktive Veränderung des Plots verhindern

- Maus im Plot → Toolbar erscheint



- außerdem `Truss`-Objekt mit Maus verschiebbar
- in den Eigenschaften von `app.UIAxes` unter `INTERACTIVITY` die Checkbox zu `Toolbar.Visible` ausschalten
- unter `CALLBACK EXECUTION CONTROL` Wert von `PickableParts` auf `none` setzen

## 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

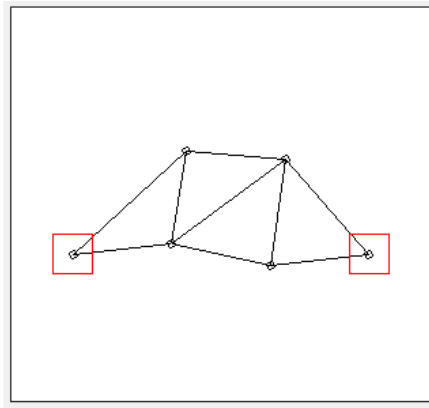
- `getframe`, `saveas` funktionieren nicht in App Designer
- Alternative ab Version 2020a: `exportgraphics`

Inhalt von `SavePlotButtonPushed`

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

fertige Version in `trussGUI.mlapp`

### Eigenschwingungen eines Fachwerks



Ordnung der Eigenschwingung

3

Eigenfrequenz: 0.126723 Hz

Animation

Save Plot

- Aufgaben:

Aufgabe 16

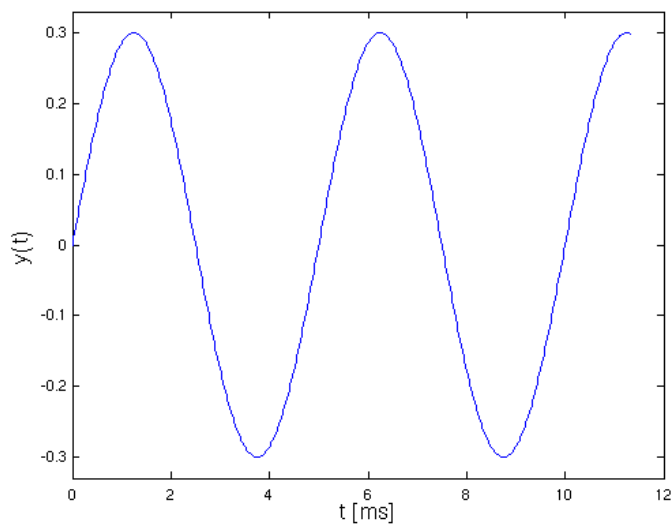
- Erzeugen eines Klangs:

Berechnen eines Sinustons vorgegebener Frequenz, Amplitude und Dauer mit `createSineWave` hier immer mit fester Samplefrequenz 44.1 kHz

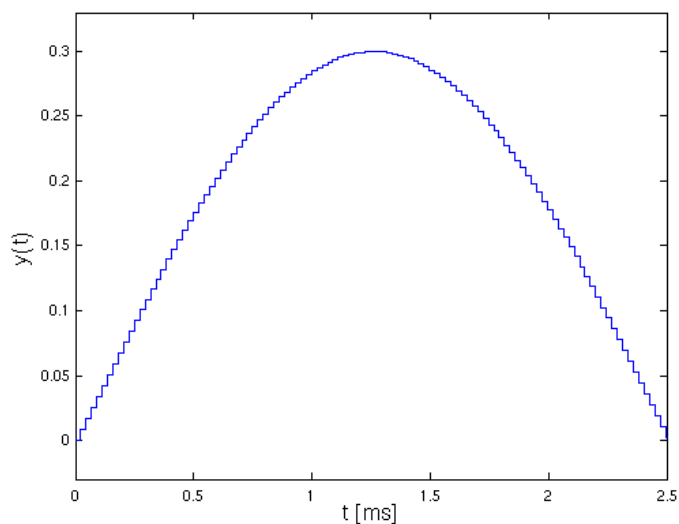
Beispiel

```
f = 200;
A = 0.3;
T = 2;
[y1, t] = createSineWave(f, A, T);
```

Plot der Schwingung



- Vergrößerung zeigt die Diskretisierung (mit Plotfunktion `stairs`)



abspielen mit

```
sound(y1, 44100)
```

Wertebereich für Sounds in Matlab: [-1, 1] (für `double`-Werte)

Klang verändern durch Hinzufügen von Obertönen

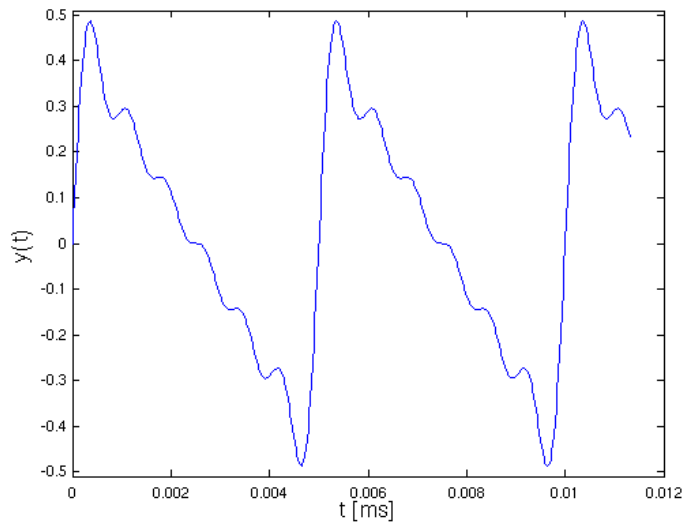
```
y2 = createSineWave(2*f, A/2, T);
y3 = createSineWave(3*f, A/3, T);
y4 = createSineWave(4*f, A/4, T);
```

```

y5 = createSineWave(5*f, A/5, T);
y6 = createSineWave(6*f, A/6, T);
y = y1 + y2 + y3 + y4 + y5 + y6;

```

Schwingungsform nahezu sägezahnförmig



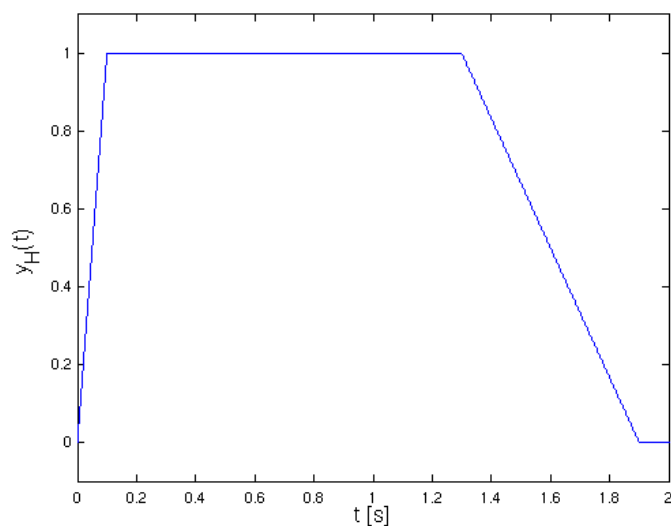
Klang "schärfer" (obertonreicher) als beim Sinus  
abspeichern als Wave-Datei

```
audiowrite("ton.mp3", y, 44100)
```

- Verändern des zeitlichen Lautstärke-Verlaufs:

typische Hüllkurve

- steigt von 0 auf 1 an (**Attack-Phase**)
- bleibt auf festem Wert (**Sustain-Phase**)
- klingt wieder auf 0 ab (**Release-Phase**)



erzeugen mit Funktion `createEnvelope`

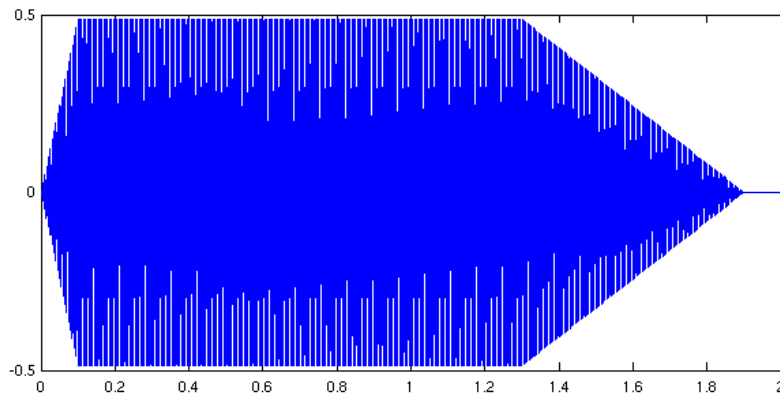
Ton mit Hüllkurve versehen und abspeichern

```

yEnv = createEnvelope(0.1, 1.2, 0.6, T);
yH = yEnv.*y;
audiowrite("ton1.mp3", yH, 44100)

```

Ergebnis `ton1.mp3`



- Analyse eines Tons:

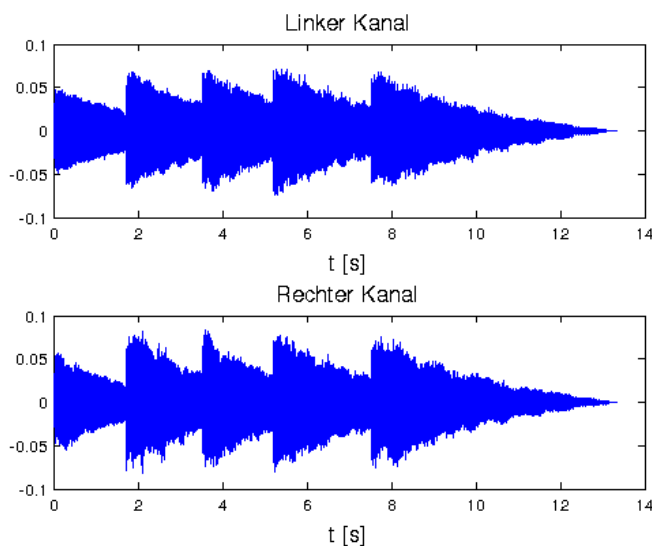
Laden von `ton2.mp3`

```
[y, Fs] = audioread("ton2.mp3");
```

- $F_s = 44100$  (Samplerate)

Eigenschaften von `y`

- Array von  $588672 \times 2$  double-Werten
- 2 Kanäle (Stereo)
- Länge  $588672/44100 = 13.3486$  s



in Mono verwandeln

```
yMono = (y(:,1) + y(:,2))/2;
```

1. Ton herausgreifen

- Anzahl der Sample aus dem Bild abgeschätzt und für das Folgende auf Zweierpotenz abgerundet

```
N1 = 73400;
N = 2^16
y1 = yMono(1:N);
```

Spektralanalyse

- bestimmt Aufbau eines Signals aus Grund- und Obertönen
- grundlegendes mathematisches Verfahren: Fourieranalyse
- in Matlab als Funktion `fft(y)`
- deutlich schneller, wenn `length(y)` Zweierpotenz

Hilfsfunktion `spektrum`

- berechnet das Spektrum  $Y$  der Werte `y`

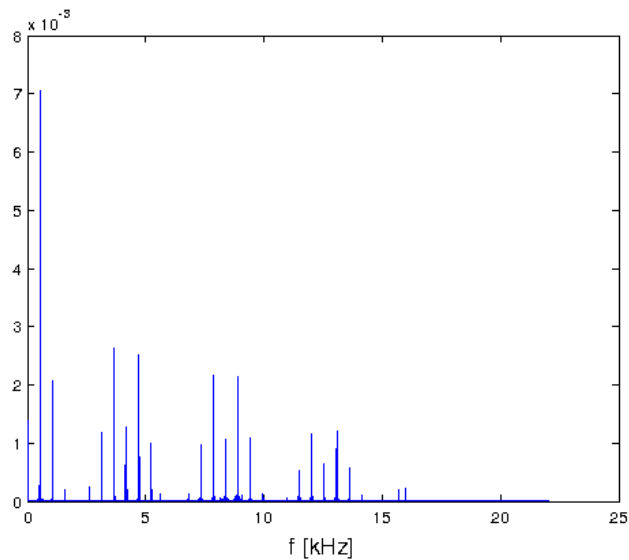
- gibt außerdem die passenden Frequenzwerte zurück
- Samplerate als Parameter

#### Eigenschaften des Spektrums

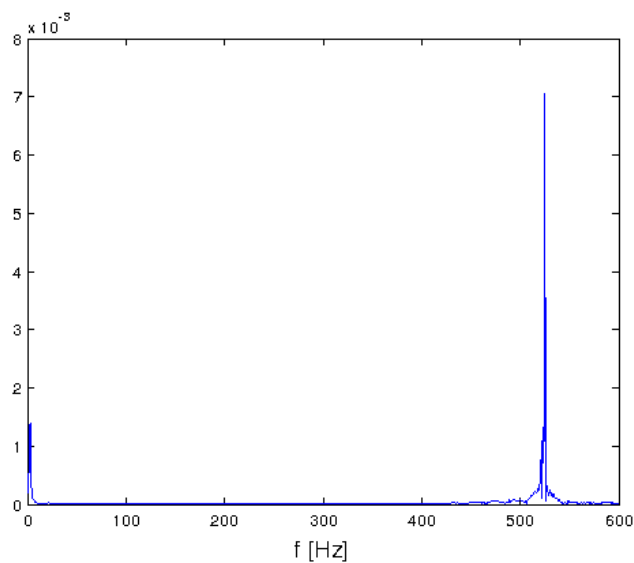
- größte erlaubte Frequenz:  $f_{\max} = F_S/2$
- Frequenzauflösung (Frequenz-Schrittweite von Y)  $\Delta f = 1/T$

#### im Beispiel

```
[Y, f] = spektrum(y1, 44100);
plot(f, Y)
```

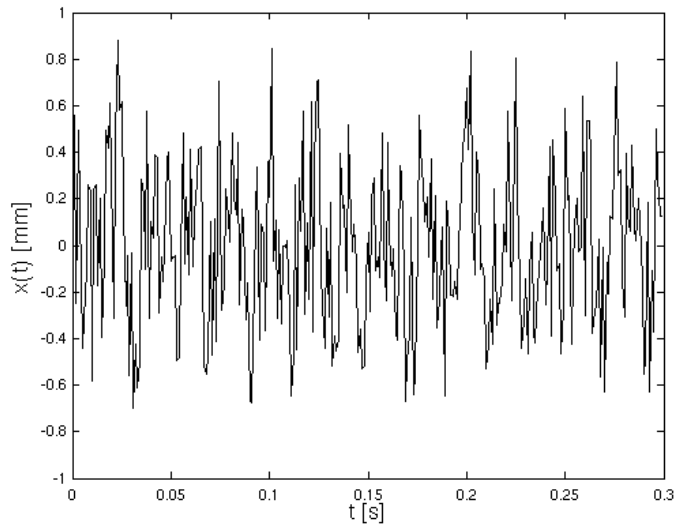


#### ■ Ausschnitt



#### Bedeutung

- Grundton bei 525 Hz (Ton:  $c^2$ )
  - viele Obertöne bei ganzzahligen Vielfachen
  - niederfrequente Schwingung bei etwa 3 Hz (Vibrato)
- Analyse einer Schwingung:
    - Störschwingungen einer Maschine werden in festen Zeitabständen aufgezeichnet



- und abgespeichert (`stoerung.dat`)

in Matlab laden

```
xx = load("stoerung.dat");
t = xx(:,1);
y = xx(:,2);
```

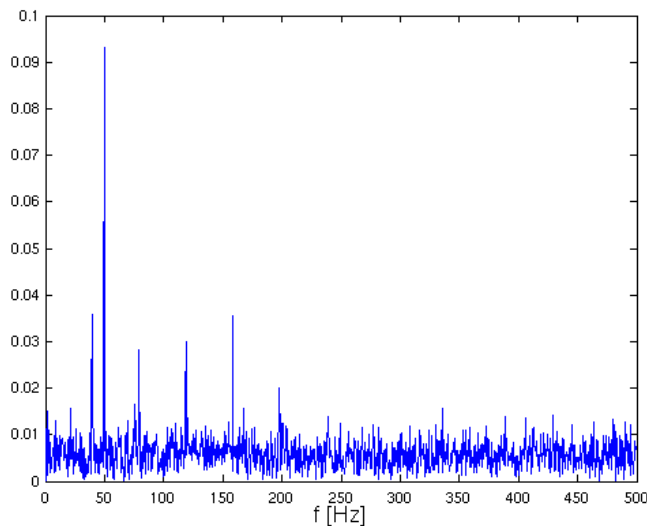
- und Samplefrequenz bestimmen

```
dt = t(2) - t(1); % sollten alle gleich sein
Fs = 1/dt
```

FFT-Analyse mit

```
[Y, f] = spektrum(y, Fs);
```

Ergebnis:



- große Spitze bei 50 Hz
- Spitzen in festen Frequenzabständen (Grundfrequenz 39.6 Hz)
- Untergrund bei allen Frequenzen

Interpretation

- Rauschen (Messfehler + allgemeine Störungen) als Untergrund
- Trafoschwingungen bei 50 Hz
- Störung mit Grundfrequenz 39.6 Hz und starken Oberfrequenzen (d. h. schnellen Änderungen, etwa Stöße)

- Datenbank-Managementsysteme
- Aufbau von Tabellen
- Grundlagen von SQL
- Arbeiten mit Tabellen
- Datenbankentwurf
- Abfragen in Datenbanken
- Datenbank-Zugriff mit Matlab

- Datenbank:

Sammlung von Daten an zentraler Stelle  
verschiedene Organisationsformen  
relationale Datenbank  $\triangleq$  Menge von Tabellen

- Datenbank-Managementsystem (**DBMS**):

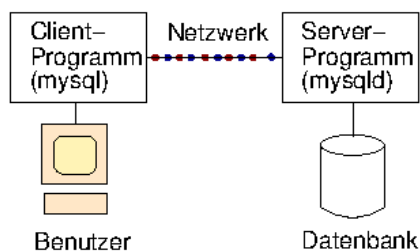
Programm zur Verwaltung von (großen) Datenbanken

viele Systeme auf dem Markt, die beliebtesten (nach [DB-Engines Ranking](#) vom Juli 2023):

- Oracle Database Server
- MySQL (Oracle/Open Source, im Kurs verwendet)
- Microsoft SQL Server
- PostgreSQL (Open Source)
- Mongo (NoSQL/Document, Open Source-artig)
- Redis (NoSQL/Key-Value, Open Source)
- IBM Db2

erlaubt einheitlichen, netzwerkweiten Zugriff auf Daten

- nach außen immer gleiches Vorgehen für alle Daten
- internes Speicherschema versteckt
- Client-Server-Architektur



gewährleistet Konsistenz der Daten, z.B. durch

- Typen mit erlaubten Bereichen (Wert für Datumsfeld muss gültig sein)
- Kontrolle beim Einfügen von Daten (Kundennummer muss zu einem Kunden gehören)
- Kontrolle beim Löschen von Daten (Kunde löschen  $\rightarrow$  sein Konto wird auch gelöscht)

garantiert Sicherheit und Vertraulichkeit

- eigene Benutzerverwaltung
- Anmelden etwa mit

```
mysql -h meinrechner -u meinuser -p <RET>
meinpasswort
```

- ausgefeiltes System von Zugriffsrechten
- z.B. Rechte zum Erstellen, Ändern, Ansehen oder Löschen von Daten

- SQL ("**Structured Query Language**):

Sprache zur Definition und Manipulation von Daten

beschreibt Aufgabe, nicht den Lösungsweg (nicht-prozedural)

Syntax an normales Englisch angelehnt

standardisiert durch ISO

- grundlegende Version: SQL92 (oder SQL2)
- SQL:1999 (SQL3) u.a. mit objektorientierten Erweiterungen
- SQL:2003 u.a. mit XML-Unterstützung
- aktuell: SQL:2019 u.a. mit mehrdimensionalen Arrays

# Aufbau von Tabellen



- Ein erstes Beispiel:

| author   | title                                                                                             | publication        | volume | number | year |
|----------|---------------------------------------------------------------------------------------------------|--------------------|--------|--------|------|
| Chow     | Parallel DEVS: A parallel, hierarchical, modular modeling formalism and its distributed simulator | Trans Soc Comp Sim | 13     | 2      | 1996 |
| Junglas  | NSA-DEVS: Combining Mealy behaviour and Causality                                                 | SNE                | 31     | 2      | 2021 |
| Zimmer   | A Modelica Library for MultiBond Graphs and its Application in 3D-Mechanics                       | Master Thesis      | NULL   | NULL   | 2006 |
| Freymann | Aufgabenorientierte Multi-Robotersteuerungen auf Basis des SBC-Frameworks und DEVS                | PhD Thesis         | NULL   | NULL   | 2020 |

- Datensatz:

Zeile in einer Tabelle, z.B.

| author  | title                                             | publication | volume | number | year |
|---------|---------------------------------------------------|-------------|--------|--------|------|
| Junglas | NSA-DEVS: Combining Mealy behaviour and Causality | SNE         | 31     | 2      | 2021 |

Beschreibung eines Objekts (**Entität**) durch Werte für die interessierenden Eigenschaften (**Attribute**)

Attribute haben Wertebereiche (**Domänen**), z.B.

| Attribut                   | Bereich            |
|----------------------------|--------------------|
| author, title, publication | Zeichenkette       |
| volume, number             | ganze Zahl         |
| year                       | gültige Jahreszahl |

spezieller Wert **NULL** = Hilfwert bei unbekanntem oder undefinierten Wert

- Tabelle:

Menge von Datensätzen

Reihenfolge der Datensätze beliebig

keine doppelten Datensätze (in der Regel)

- Schlüssel:

jeder Datensatz beschreibt ein festgelegtes Objekt (einen bestimmten Artikel, eine bestimmte Person, Produkt, etc)

ein oder mehrere Attribute (**Schlüssel**) legen Objekt fest

Beispiele:

| Objektart      | Schlüssel                  |
|----------------|----------------------------|
| Artikel        | DOI                        |
| Buch           | ISBN                       |
| Auto           | Kennzeichen                |
| Vorlesungsraum | Gebäudenummer + Raumnummer |

Anforderungen an Schlüssel

- Eindeutigkeit (verschiedene Objekte haben verschiedene Schlüssel)
- Definiertheit (jedes Objekt hat einen Schlüsselwert != NULL)
- Konstanz (jedes Objekt hat immer denselben Schlüsselwert)

natürliche Schlüssel häufig ungeeignet

z.B. Vorname und Name

- nicht eindeutig (Hans Müller)

- nicht definiert (direkt nach der Geburt)
- nicht konstant (Namensänderungen, etwa bei Hochzeit)

Aushilfe: künstliche Schlüssel wie ISBN, Kundennummer, Matrikelnummer etc.

hier immer in der Form `tabelleId` (z. B. `authorId`, `paperId`)

- Allgemeine Syntax:

Schlüsselwörter unabhängig von Klein-/Großschreibung

bei Namen für Tabellen, Attribute etc. Klein-/Großschreibung relevant!

Befehle über mehrere Zeilen möglich

Ende eines Befehls mit ;

i.f. Großbuchstaben für Schlüsselwörter

- Datenbank in SQL:

erzeugen mit SQL-Kommando

```
CREATE DATABASE datenbank;
```

danach zum Benutzen "in Datenbank wechseln" mit

```
USE datenbank;
```

erspart bei Tabellen Angabe des vollen Namens `datenbank.tabelle`

Liste aller vorhandenen Tabellen in der ausgewählten Datenbank:

```
SHOW TABLES;
```

Liste aller vorhandenen Datenbanken mit

```
SHOW DATABASES;
```

- Datentypen für Attribute:

viele verschiedene Typen definiert

einige Beispiele

| Name           | Bedeutung                                            |
|----------------|------------------------------------------------------|
| CHAR (n)       | Zeichenkette mit genau n Zeichen                     |
| VARCHAR (n)    | Zeichenkette variabler Länge mit höchstens n Zeichen |
| INTEGER        | ganze Zahl mit Standardbereich                       |
| DECIMAL (p, n) | Zahl mit p Stellen und n Nachkommastellen            |
| DATE           | Datum (Format 'YYYY-MM-DD')                          |
| TIME           | Uhrzeit (Format '14:12:00')                          |

- Erzeugen von Tabellen:

Syntax

```
CREATE TABLE tabelle (feld1 typ1, feld2 typ2, ...);
```

zusätzliche Parameter bei CREATE

|                |                                                                                     |
|----------------|-------------------------------------------------------------------------------------|
| NOT NULL       | darf nicht den Wert NULL haben                                                      |
| UNIQUE         | Werte für alle Datensätze verschieden                                               |
| PRIMARY KEY    | zentraler Zugriffsschlüssel<br>erfordert NOT NULL<br>impliziert UNIQUE              |
| AUTO_INCREMENT | automatisches Hochzählen des Index<br>für PRIMARY KEY<br>nur bei Integer-Datentypen |

Beispiel

```
CREATE TABLE Author (
 name varchar(80),
 firstName varchar(80),
 institution varchar(80),
 authorId integer NOT NULL AUTO_INCREMENT,
 PRIMARY KEY(authorId)
);
```

- Löschen von Tabellen:

Syntax

```
DROP TABLE tabelle;
```

- Information über Tabellen:

Syntax

```
SHOW COLUMNS FROM tabelle;
```

andere Form

```
DESCRIBE tabelle;
```

- Einfügen von Daten:

Syntax

```
INSERT INTO tabelle (feld1, feld2 ..) VALUES
 (wert1, wert2 ...);
```

Beispiel

```
INSERT INTO Author (name, firstName, institution) VALUES
 ('Zimmer', 'Dirk', 'Deutsches Zentrum für Luft- und Raumfahrt (DLR)');
```

fehlende Felder bekommen den Wert NULL oder ggf. einen vordefinierten Defaultwert

auch mehrere Datensätze auf einmal möglich, durch Komma getrennt

- Letzte durch AUTO\_INCREMENT erzeugte Id abfragen

Syntax

```
SELECT LAST_INSERT_ID();
```

nicht standardisiert, hier MySQL-Variante

- Aufgaben

- Aufgabe 17

- Beispiel Literatur-Datenbank:

| Author                              | Title                                                                                                                           | Pages | Year | Publication            | Keyword                                               |
|-------------------------------------|---------------------------------------------------------------------------------------------------------------------------------|-------|------|------------------------|-------------------------------------------------------|
| Chow                                | Parallel DEVS: A parallel, hierarchical, modular modeling formalism and its distributed simulator                               | 14    | 1996 | Trans Soc Comp Sim     | simulation, discrete events                           |
| Zimmer, Cellier                     | The Modelica Multi-bond Graph Library                                                                                           | 10    | 2006 | 5th Intl Modelica Conf | simulation, bond graphs                               |
| Junglas, Pawletta                   | Non-standard Queuing Policies: Definition of ARGESIM Benchmark C22                                                              | 5     | 2019 | SNE                    | simulation, discrete events, benchmark                |
| Jammer, Pawletta, Kunert, Pawletta  | Beschleunigung eines Reinforcement-Learning-Algorithmus durch Parallelverarbeitung für Robotikanwendungen                       | 4     | 2019 | 8. Workshop STS/GMMS   | parallel computing, robotics, artificial intelligence |
| Jammer, Junglas, Pawletta, Pawletta | Implementing Standard Examples with NSA-DEVS                                                                                    | 8     | 2022 | SNE                    | simulation, discrete events                           |
| Zimmer, Meißner, Weber              | The DLR ThermoFluid Stream Library                                                                                              | 21    | 2022 | Electronics            | simulation, thermofluid                               |
| Jammer, Junglas, Pawletta           | Solving ARGESIM Benchmark CP2 'Parallel and Distributed Simulation' with Open MPI and Matlab PCT – Lattice Boltzmann Simulation | 8     | 2023 | SNE                    | benchmark, parallel computing                         |

- Probleme mit der Tabelle:

Felder mit mehreren Einträgen (Author, Keyword)

- → kann nicht nach individuellem Autor oder Schlagwort durchsucht werden

hohe Redundanz

- gleiche Begriffe an mehreren Stellen (Keyword, Author, Publication)
- → vergrößerter Speicherbedarf
- → Gefahr von Inkonsistenzen (z.B. durch Schreibfehler)

Datenbanken brauchen klaren Entwurf!

- Entity-Relationship-Modell (**ERM**):

Verfahren zum systematischen Entwurf von Datenbanken

## Entity

- eindeutig identifizierbares Objekt
- beschrieben durch eine Menge von Attributen
- Typ des Objekts = Liste der Attributnamen und -typen
- Entitäten ergeben Tabellen

Vergleich mit objektorientierter Programmierung

- Tabelle  $\triangleq$  Klasse
- Attribut  $\triangleq$  Datenfeld
- Datensatz  $\triangleq$  Objekt

## Relationship

- Beziehung zwischen Entitäten
- kann selbst Eigenschaften (Attribute) haben
- kann zwei oder mehr verschiedene Arten von Entitäten verknüpfen

- kann verschiedene Vielfachheiten haben, z.B.

|     |                                               |
|-----|-----------------------------------------------|
| 1:1 | ein Objekt A mit einem Objekt B               |
| 1:N | ein Objekt A mit mehreren Objekten B          |
| N:M | ein A mit mehreren B und ein B mit mehreren A |

- ein Objekt möglicherweise mit keinem anderen verknüpft
- Notation z.B.

1:0..\* = ein Objekt A mit gar keinem, einem oder mehreren B verknüpft

graphische Darstellung z.B. als Klassendiagramm (**UML-Diagramm**)

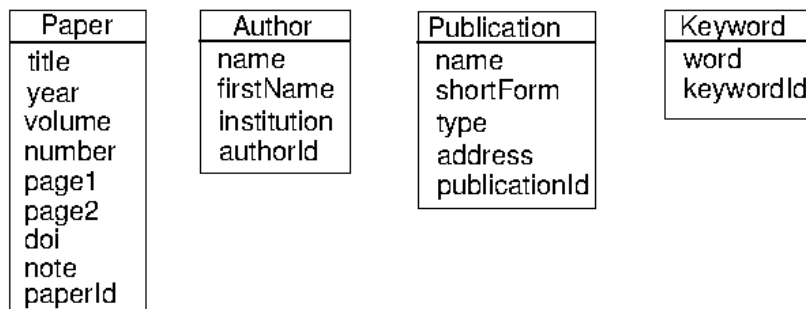


- Anwendung auf Beispiel:

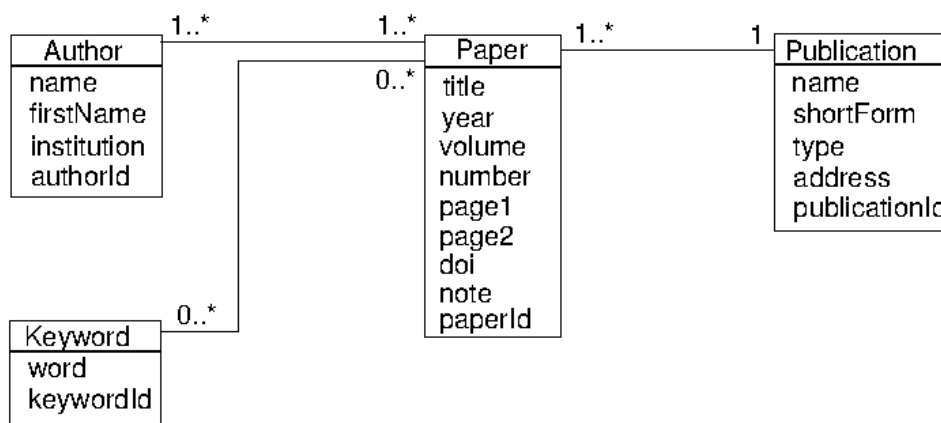
Entitäten sind

- Paper
- Author
- Publication
- Keyword

Attribute



Beziehungen zwischen den Entitäten



z.B. Beziehung zwischen Paper und Publication

- ein Paper hat genau eine Publication (1)
- ein Publication hat ein oder mehrere Paper (1..\*)

einige Designentscheidungen

- nur Autoren aufgeführt, für die Paper in der Datenbank existieren
- zu einem Keyword kann evtl. noch kein Buch vorhanden sein
- Publication type (z.B. Journal, Proceedings, Thesis, Manual, Technical Report) nicht ausgliedert

- Umsetzung der Relationen:

unterschiedlich nach Grundtypen 1:1, 1:N, N:M

bei 1:N

- Schlüssel der 1-Seite als Attribut zur Tabelle der N-Seite hinzufügen (**Fremdschlüssel**)
- im Beispiel: Tabelle `Paper` bekommt Attribut `publicationId`

bei 1:1

- gelegentlich für Spezialisierungen (z.B. allgemeine Daten in `Person`, Spezialdaten in `Employee` oder in `Customer`)
- zwei Tabellen oder beide Tabellen zu einer mit allen Attributen zusammenfassen
- häufig durch Hinzufügen der Attribute einer Tabelle zur anderen (z.B. Telefon-Nr. zu `Author`)

bei N:M

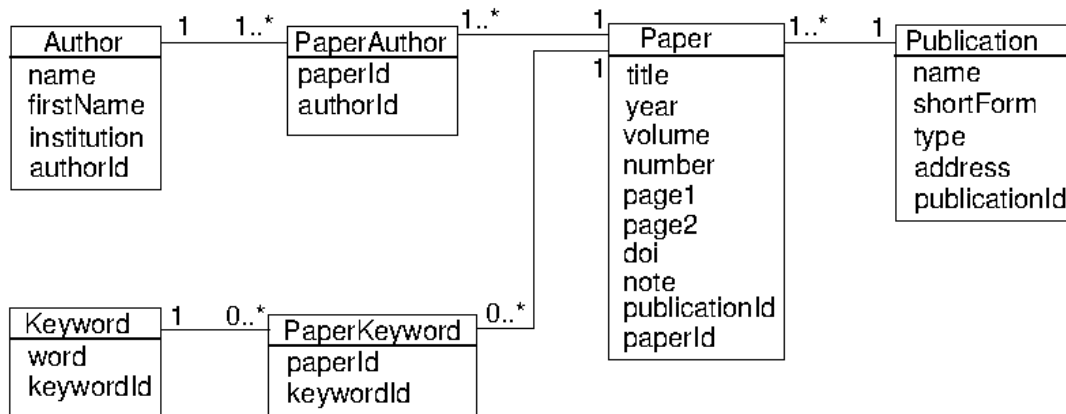
- neue Tabelle für die Relation einführen (**Verbindungsentität**)
- enthält Fremdschlüssel beider Partner
- Primärschlüssel ist häufig Paar der Fremdschlüssel
- enthält manchmal weitere Attribute

im Beispiel:

- neue Tabelle `PaperAuthor`
- Attribute `paperId` und `authorId`
- bilden zusammen Primärschlüssel
- anlegen mit `PRIMARY KEY(paperId, authorId)`
- einen Eintrag für jede `Author/Paper`-Kombination
- analog für `PaperKeyword`

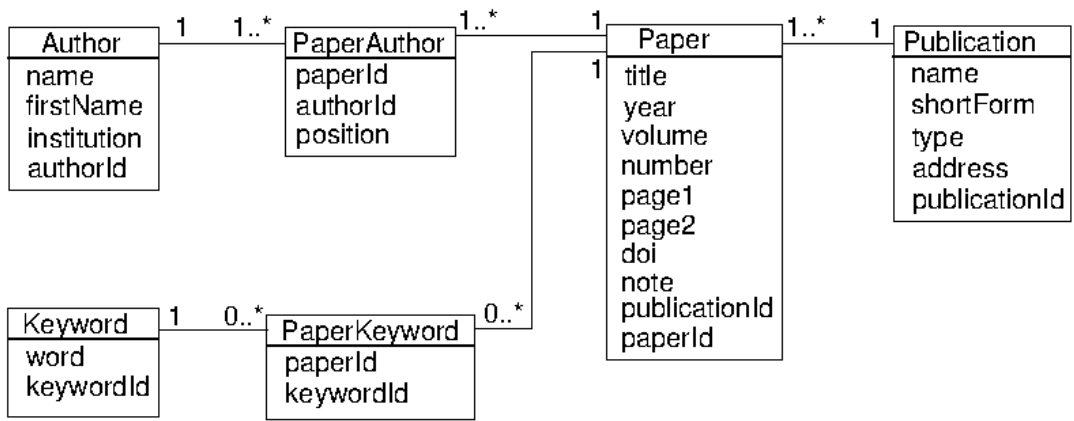
- Fertige Bücher-Datenbank:

als Klassendiagramm



Problem

- Reihenfolge der Autoren in einem `Paper` wichtig
- Info als Attribut `position` abspeichern, aber wo?
- passt nicht in `Author` (ein Autor hat mehrere `Paper` mit verschiedenen Positionen)
- passt nicht in `Paper` (ein `Paper` hat mehrere Autoren mit verschiedenen Positionen)
- gehört in `PaperAuthor`



Tabellen mit Beispieldaten

- Aufgaben

Aufgabe 17

- Einfache Abfragen
- Abfragen über mehrere Tabellen
- Komplizierte Abfragen

# Einfache Abfragen



- **SELECT-Kommando:**

grundlegendes Kommando für Abfragen (**Queries**)

sehr vielfältig

große Zahl an optionalen Klauseln

Grundabfrage zur Anzeige aller Daten einer Tabelle

```
SELECT * FROM tabelle;
```

- **Auswahl von Spalten (Projektion):**

nicht alle Angaben in einer Tabelle für die aktuelle Abfrage relevant

Auswahl über Liste von gewünschten Attributen

```
SELECT attribut1, attribut2, .. FROM tabelle;
```

Ausgabe kann sortiert werden mit der Zusatzklausel

```
ORDER BY attribut1, attribut2, ..
```

Default: aufsteigende Reihenfolge, absteigend mit

```
ORDER BY attribut DESC
```

Beispiel

```
SELECT title, page1, page2, year FROM Paper
ORDER BY year DESC, title;
```

**Resultat**

- **Auswahl von Zeilen (Selektion):**

mit der `WHERE`-Klausel

viele gängige Operationen

|            |                                                                                       |
|------------|---------------------------------------------------------------------------------------|
| Vergleiche | <, >, <=, =, !=                                                                       |
| Logik      | AND OR NOT                                                                            |
| Arithmetik | +, -, *, /, %                                                                         |
| LIKE       | String-Vergleich mit Wildcards<br>% (beliebig viele Zeichen)<br>_ (genau ein Zeichen) |

Beispiel

```
SELECT title, page1, page2, year FROM Paper
WHERE year >= 2020;
```

**Resultat**

Beispiel

```
SELECT title, page1, page2, year FROM Paper
WHERE title LIKE "%NSA-DEVS%";
```

**Resultat**

Abfragen auf `NULL`

- mit `attribut=NULL` immer leer
- stattdessen mit `attribut IS NULL`

- Funktionen im `SELECT`:

bei Auswahl-Attributen oder in WHERE-Bedingung

große Anzahl vordefiniert, z.B.

|                       |                                 |
|-----------------------|---------------------------------|
| numerische Funktionen | SIN, LOG                        |
| String-Funktionen     | SUBSTRING, CONCAT, LENGTH, TRIM |
| Datums-Funktionen     | NOW, WEEK, DATE_ADD             |

Beispiel: Berechnung der Anzahl der Seiten

```
SELECT title, page2 - page1 + 1, year FROM Paper
WHERE title LIKE "%NSA-DEVS%"
ORDER BY year;
```

### Resultat

- Weitere Anwendungen von WHERE:

#### Ändern von Daten

```
UPDATE tabelle SET attribut1=wert1, ...
WHERE bedingung;
```

#### Löschen von Daten

```
DELETE FROM tabelle WHERE bedingung;
```

# Abfragen über mehrere Tabellen



- **SELECT mit mehreren Tabellen (Join):**

- mehrere Tabellen angeben

```
SELECT attribut1, attribut2, ...
FROM table1 JOIN table2 ...
```

ohne weitere Einschränkung alle Kombinationen aus allen Tabellen

Beispiel

```
SELECT title, name FROM Paper JOIN Publication;
```

liefert 130 Einträge (13 Paper x 10 Publications)

- Verknüpfung der Tabellen durch Übereinstimmung der Schlüssel

```
SELECT title, name FROM Paper INNER JOIN Publication
ON Paper.publicationId = Publication.publicationId;
```

bei gleichen Attributbezeichnungen Verknüpfung automatisch (**Natural Join**)

```
SELECT title, name FROM Paper NATURAL JOIN Publication;
```

Resultat

- bessere Attributsbezeichnungen bei Joins mit AS, z.B.

```
SELECT title, name AS journal
FROM Paper NATURAL JOIN Publication
ORDER BY journal;
```

Resultat

- **Gruppierung:**

weitere Klausel für SELECT

```
GROUP BY attribut1, attribut2, ..
```

fasst Datensätze mit gleichen Werten für Attribute zusammen

übrige (nicht-gleiche) Attribute brauchen Funktionen zum Kombinieren

wichtigste Funktionen

|          |                                    |
|----------|------------------------------------|
| COUNT    | Anzahl der zusammengefassten Werte |
| SUM      | Summe der jeweiligen Attribute     |
| AVG      | Durchschnitt                       |
| MAX, MIN | Maximum, Minimum                   |

- **Beispiele:**

Wieviele Paper gibt es für jedes Jahr, wie groß ist der gesamte Seitenzahl pro Jahr?

```
SELECT COUNT(*) as count, SUM(page2-page1+1) as totalPages, year FROM Paper
GROUP BY year ORDER BY year;
```

Resultat

Wieviele Paper gibt es von jedem Journal?

```
SELECT name as journal, COUNT(*) as count
FROM Paper NATURAL JOIN Publication
GROUP BY journal
ORDER BY count DESC;
```



# Komplizierte Abfragen

- Einige beispielhafte Fragestellungen:
  - Bestimme die Gesamt-Seitenzahl aller Paper der einzelnen Publications.
  - Bestimme zu jedem Schlagwort die Zahl der passenden Paper.
  - Bestimme für jeden Autor alle Paper, an denen er mitgeschrieben hat.
  - Ermittle die Namen aller Autoren, die mit Thorsten Pawletta zusammen ein Paper verfasst haben.

- Vorgehensweise bei komplizierten Abfragen:

alle benötigten Tabellen (für Attribute und benötigte Relationen) zusammensuchen

- graphisch: Klassenkästchen mit benötigten Attributen nebeneinander stellen

benötigte Verknüpfungen über die Fremdschlüssel formulieren

- graphisch: entsprechende Schlüssel verbinden und mit Relation (=, !=, Wert) beschriften
- bei = ggf. NATURAL JOIN benutzen

bei Gruppierung: zu gruppierende Attribute bestimmen, Zusammenfassungsfunktionen bei den anderen

- graphisch: zu gruppierende Attribute einkreisen, an alle anderen Funktion schreiben

Feinarbeit: Umbenennung von Attributen, Sortierung

- Anwendung bei Beispielfrage 1:

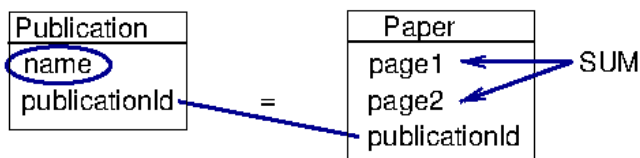
benötigt werden

| Tabelle     | Attribute                   |
|-------------|-----------------------------|
| Publication | name, publicationId         |
| Paper       | page1, page2, publicationId |

verknüpft wird über Gleichheit der publicationId-Schlüssel (NATURAL JOIN)

gruppiert wird über name, über page2-page1+1 wird summiert

graphisch also



Feinschliff

- Spalten umbenennen in verlag und gesamtpreis
- absteigend sortieren nach gesamtpreis

SQL-Kommando

```
SELECT name AS publication , SUM(page2-page1+1) AS totalPages
FROM Paper NATURAL JOIN Publication
GROUP BY publication
ORDER BY totalPages DESC;
```

Resultat

- Aufgaben

Aufgabe 18

- Database-Toolbox:

Zugriff zu SQL- und einigen NoSQL-Datenbanken  
grundsätzlich Zugriff möglich über ODBC- oder JDBC-Treiber  
für MySQL direkt mit "MySQL Native Interface"

- Verbindung zur Datenbank aufnehmen:

geeignete Options-Struktur erzeugen

```
opts = databaseConnectionOptions("native", "MySQL");
```

Daten in Options-Struktur speichern

```
opts = setoptions(opts, "DataSourceName", mySource, ...
 "DatabaseName", myDB, "Server", myServer, "PortNumber", myPort);
```

- Parameter

| Variable | Bedeutung              | Beispielwert      |
|----------|------------------------|-------------------|
| mySource | selbstgewählter Name   | "MySQLDataSource" |
| myDB     | ausgewählte Datenbank  | "literature"      |
| myServer | Adresse/IP des Servers | "127.0.0.1"       |
| myPort   | Port des Servers       | 3306              |

Testen der Verbindung (wichtig!)

```
[dbOk, message] = testConnection(opts, myUser, myPass);
if ~dbOk
 fprintf("Connection to database '%s' failed!\n", myDB)
 fprintf("message: %s\n", message)
end
```

- mit Benutzernamen `myUser` und Passwort `myPass`
- im Kurs: `myUser = "phwt", myPass = "strenggeheim"`

bei Erfolg Daten speichern mit

```
saveAsDataSource(opts);
```

Verbindung öffnen

```
conn = mysql(mySource, myUser, myPass);
```

- nach Benutzen der Datenbank wieder schließen

```
close(conn);
```

- Datenaustausch zwischen Matlab und Datenbank:

SQL-Kommando als String erzeugen, z.B.

```
sqlquery = "SELECT word FROM Keyword";
```

Kommando absetzen

```
data = fetch(conn, sqlquery);
```

`data` ist eine Tabelle mit den Ergebnis-Daten

auf Spalten über Namen zugreifen

```
>> data.word
 "simulation"
 "thermofluid"
 "bond graphs"
```

```
"parallel computing"
"robotics"
"artificial intelligence"
"discrete events"
"benchmark"
```

bei SQL-Kommando ohne Rückgabewert (z. B. INSERT)

```
execute(conn, sqlquery);
```

- Beispiel-Anwendung `showPapers` :

Aufruf

```
showPapers(keyword)
```

- liefert übersichtliche Liste der Paper zum Schlüsselwort

Beispiel

```
>> showPapers("bond graphs")
D.Zimmer:
A Modelica Library for MultiBond Graphs and its Application in 3D-Mechanics

D.Zimmer, F.Cellier:
The Modelica Multi-bond Graph Library
```

Hilfsfunktion `getAuthorString(conn, id)`

- gibt die formatierte Liste der Autoren zurück
- verkürzt Vornamen auf Anfangsbuchstaben mit `extract`
- hängt alle Namen hintereinander, einschließlich Komma
- entfernt überflüssiges Komma am Ende mit `extractBefore`


- Aufgaben

Aufgabe 19

# 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.

In einigen zusammengehörenden Aufgaben sollen Teilfunktionalitäten zur Simulation einer Schwingerkette implementiert werden. Sie sind durch das Icon  gekennzeichnet.

- 🔦 Aufgabe 1
- 🔦 Aufgabe 2
- 🔦 Aufgabe 3
- 🔦 Aufgabe 4
- 🔦 Aufgabe 5
- 🔦 Aufgabe 6
- 🔦 Aufgabe 7
- 🔦 Aufgabe 8 
- 🔦 Aufgabe 9 
- 🔦 Aufgabe 10 
- 🔦 Aufgabe 11
- 🔦 Aufgabe 12 
- 🔦 Aufgabe 13 
- 🔦 Aufgabe 14 
- 🔦 Aufgabe 15 
- 🔦 Aufgabe 16 
- 🔦 Aufgabe 17
- 🔦 Aufgabe 18
- 🔦 Aufgabe 19

# Aufgabe 1



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

$$y_1 = \sin(\omega_1 t)$$

$$y_2 = \sin(\omega_2 t)$$

und ihre Summe  $s = y_1 + y_2$  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  $T_1 = 26.5 \text{ °C}$  ist gesucht.
  - a. Die Datei `ex02.dat` enthält Ergebnisse von Messungen von  $c_p(T)$ . Auf einen Temperaturwert (in  $\text{°C}$ ) folgt dabei immer der dazugehörige  $c_p$ -Wert (in  $\text{kJ}/(\text{kg K})$ ). 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

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

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

$$V [\text{cm}^3] \quad p_{\text{unten}}[\text{bar}] \quad p_{\text{oben}} [\text{bar}]$$

wobei  $p_{\text{unten}}$  bzw.  $p_{\text{oben}}$  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  $n = 3000 \text{ U/min}$  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

$$x_1 = (2.7, 2.5)$$

$$x_2 = (36.1, 5.5)$$

$$x_3 = (28.4, 31.9)$$

$$x_4 = (4.0, 24.6)$$

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):

$$\alpha_1 = -64.5^\circ$$

$$\alpha_2 = 62.0^\circ$$

$$\alpha_3 = 147.6^\circ$$

$$\alpha_4 = -122.7^\circ$$

- Erstellen Sie eine Funktion `senderOrt(peilOrte, winkel)`, die aus den Positionen der Peilstationen (als  $4 \times 2$ -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

$$x = A \setminus b$$

## Aufgabe 7



- Finden Sie alle Lösungen der Gleichung

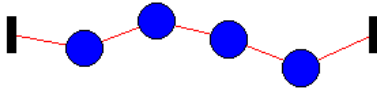
$$\tan(x) = x \ln(0.1 x)$$

im Intervall  $[0, 10]$ .

## Aufgabe 8



- Eine (transversale) Schwingerkette bestehe aus  $N$  Massen  $m_i$ , die durch  $N+1$  Federn  $c_i$  mit ihren direkten Nachbarn bzw. der Einspannung am Rand verbunden sind. Der Anfangszustand wird durch  $N$  Auslenkungen  $y_i$  charakterisiert, die Anfangsgeschwindigkeiten seien alle 0.



- Schreiben Sie Funktionen
  - `[m, c, y] = loadChain(filename)`
  - `saveChain(m, c, y, filename)`

zum Laden bzw. Speichern einer Schwingerkette zusammen mit ihrem Anfangszustand. Die Parameter bedeuten

|                       |                                 |
|-----------------------|---------------------------------|
| <code>m</code>        | Massen (N-Vektor)               |
| <code>c</code>        | Federsteifigkeiten (N+1-Vektor) |
| <code>y</code>        | Anfangsauslenkungen (N-Vektor)  |
| <code>filename</code> | Name der Datei (ohne Endung)    |

- Die Dateien haben die Endung `.chain`. Das Format kann dem Beispiel [demo.chain](#) entnommen werden. Testen Sie Ihre Routinen auch am Beispiel [long.chain](#).

## Aufgabe 9



- Führen Sie eine Datenstruktur ein, die die Informationen einer Schwingerkette zusammenfasst, und schreiben Sie die Dateifunktionen entsprechend um:
  - `chain = loadChain(filename)`
  - `saveChain(chain, filename)`
- Erzeugen Sie außerdem eine einfache Hilfsfunktion
  - `chain = createDemoChain(),`die die in `demo.chain` gespeicherte Schwingerkette direkt erzeugt.

## Aufgabe 10



- Schreiben Sie eine Funktion

```
plotChain(y),
```

die eine um  $y$  ausgelenkte Schwingerkette graphisch Bild darstellt. Orientieren Sie sich an der Abbildung in [Aufgabe 16](#).

- Erweitern Sie die Funktion für spätere Zwecke um ein Argument `axes`,

- `plotChain(axes, y)`,

das einen Zeiger (**handle**) auf ein schon vorhandenes Achsensystem enthält. Wozu könnte das sinnvoll sein?

# Aufgabe 11



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

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

mit

- $F_R(x) = -b_1 \dot{x}$  (laminare Reibung)
- $F_R(x) = -b_2 \dot{x}^2 \text{sign}(\dot{x})$  (turbulente Reibung)
- Plotten Sie beide Kurven in einem Diagramm, zusammen mit der Bahn bei reibungslosem freien Fall. Plotten Sie ebenso die drei Geschwindigkeitskurven.
- Werte:

$$m = 75 \text{ kg}$$

$$g = 9.81 \text{ m/s}^2$$

$$b_1 = 5 \text{ kg/s}$$

$$b_2 = 0.1 \text{ kg/m}$$

$$x_0 = 5000 \text{ m}$$

$$v_0 = 0$$

## Aufgabe 12



- Schreiben Sie eine Funktion

- `[M, C] = createMatrices(chain),`

die zu einer Schwingerkette die Massenmatrix  $M$  und die Steifigkeitsmatrix  $C$  berechnet.

- Zur einfachen Lösung der Differentialgleichung der Schwingerkette erzeugen Sie eine Funktion

- `[t, x] = solveVibrationODE(M, C, x0, tEnd, tStep),`

mit folgenden Parametern

|       |                      |
|-------|----------------------|
| M     | Massenmatrix         |
| C     | Steifigkeitsmatrix   |
| x0    | Anfangsauslenkungen  |
| tEnd  | Endzeit der Lösung   |
| tStep | Ausgabe-Schrittweite |

(Anfangszeit sei  $t_0 = 0$ , Anfangsgeschwindigkeiten  $v_i(0) = 0$ )

und Ergebniswerten

|   |                                                           |
|---|-----------------------------------------------------------|
| t | Vektor mit Zeitwerten                                     |
| x | Array mit Werten der Auslenkung zur Zeit t für jede Masse |

- Verwenden Sie zum Testen die Demo-Schwingerkette und plotten Sie die Ergebnisse.

## Aufgabe 13



- Schreiben Sie eine Funktion

- `plotODESolutions(axes, t, x, index),`

die die Lösungen  $(t, x)$  der Differentialgleichung der Schwingerkette in einem vorgegebenen Achsensystem `axes` plottet. Der Vektor `index` enthält die Nummern der zu plottenden Kurven.

- Wählen Sie zur Darstellung einen schwarzen Hintergrund und für die Bewegung der Massen abgestufte Grüntöne (von vollem Grün bis in Weiß übergehend).

## Aufgabe 14



- Schreiben Sie eine Funktion

```
f = computeFrequency(chain, nr),
```

die die Eigenschwingungen der Schwingerkette `chain` berechnet und die Frequenz der `nr`-ten Eigenschwingung zurückgibt.

## Aufgabe 15



- Schreiben Sie eine Funktion

- `plotChainAnimation(axes, x)`,

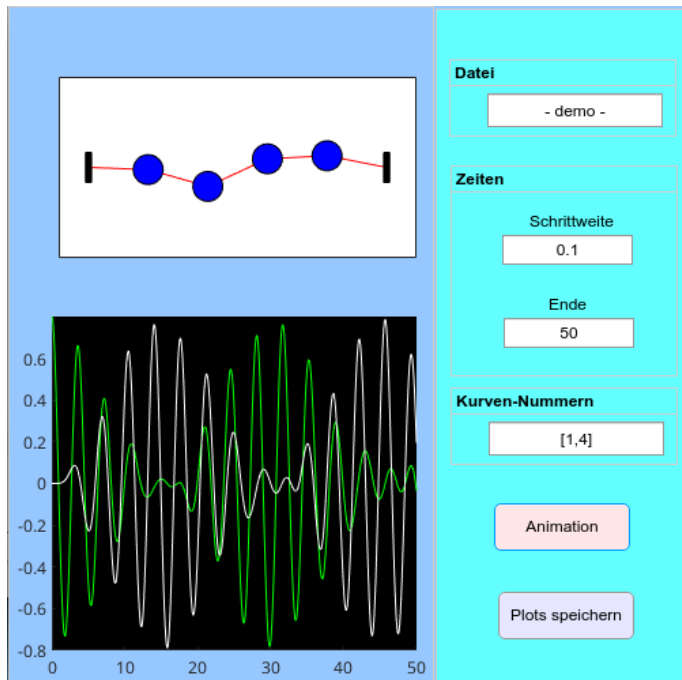
die die zu festen Zeitschritten ermittelte Lösung  $x$  der Differentialgleichung der Schwingerkette in einem vorgegebenen Achsensystem `axes` als Animation anzeigt. Verwenden Sie dazu die vorhandene Funktion `plotChain`.

- Verwenden Sie zum Testen die Schwingerkette `long.chain` und vergleichen Sie das Ergebnis mit dem des Applets.

## Aufgabe 16



- Programmieren Sie eine graphische Oberfläche `chainGUI`, die die Schwingungen einer Schwingerkette gleichzeitig als Animation und als Kurvenplot darstellt, etwa wie in folgendem Screenshot:



Folgende Größen müssen eingegeben werden können:

- Datei zur Definition der Schwingerkette
  - Schrittweite und Endzeit der Simulation
  - Auswahl der anzuzeigenden Kurven
  - Start der Animation
- Fügen Sie einen Button zum Download beider Fenster als Dateien `animation.png` und `schwingung.png` hinzu. Laden Sie die Schwingerkette `long.chain` in Ihr Programm, lassen Sie es laufen und erzeugen Sie Dateien `animation.png` und `schwingung.png` mit den jeweiligen Plots.

## Aufgabe 17



- In einem Labor werden Zugversuche gemacht, deren Ergebnisse in einer Datenbank gespeichert werden:
  - Eine Messung enthält jeweils eine Auslenkung in mm sowie eine Kraft in kN.
  - Eine Messreihe umfasst viele Messungen, Länge und Durchmesser der Probe (jeweils in mm), ein Datum und den Namen des Bearbeiters.
- Entwerfen Sie ein entsprechendes Datenbankschema, implementieren Sie es und füllen Sie es mit Datensätzen. Geben Sie allen Testreihen ein Datum, das mindestens ein Jahr her ist.

## Aufgabe 18



- Entwickeln Sie SQL-Kommandos zur Beantwortung der folgenden Fragestellungen und testen Sie sie an der Beispiel-Datenbank:
  - a. Bestimmen Sie zu jedem Schlagwort die Zahl der passenden Paper.
  - b. Bestimmen Sie für jeden Autor die Titel aller Paper, an denen er mitgeschrieben hat.
  - c. Erstellen Sie eine Liste aller Autoren und der Publikationen, in denen sie Paper veröffentlicht haben, sortiert nach Namen der Autoren. Achten Sie darauf, dass keine Einträge mehrmals vorkommen.
  - d. Ermitteln Sie die Namen aller Autoren, die Paper zum Schlagwort "thermofluid" geschrieben haben, mit bzw. ohne Angabe der Titel, jeweils sortiert nach Autornamen.
  - e. Ermitteln Sie die Namen aller Autoren, die mit Thorsten Pawletta zusammen ein Paper verfasst haben.

# Aufgabe 19



- Eine Testmaschine speichert die Ergebnisse einer Messreihe als ASCII-Datei `tensileTestNNN.dat` (NNN = fortlaufende dreiziffrige Nummer) in folgendem Format:

```
L0 in mm
d in mm
Datum (in ISO-Standardform)
Bearbeiter (Vor- und Nachname)
Anzahl der Messungen
Messung 1
Messung 2
..
Messung N
```

Dabei hat eine Messung die Form

```
Auslenkung_in_mm Kraft_in_kN
```

Eine einfache Datei könnte z. B. so aussehen:

```
60.00
 4.00
2022-06-04
Antonia Adler
 2
 0.070 2.889
 0.080 3.207
```

- Schreiben Sie eine Matlabfunktion `importTensileData(nr)`, die eine solche Datei mit gegebener Nummer einliest und die Daten in der Datenbank aus [Aufgabe 17](#) speichert.
- Gehen Sie davon aus, dass nur die folgenden Bearbeiter vorkommen:

```
Antonia Adler
Bernd Biber
Christa Chamäleon
Dieter Dorsch
```

und legen Sie sie der Einfachheit halber schon vorweg manuell in der Tabelle `Operator` an.

- Testen Sie Ihre Funktion mit Hilfe der drei Datensätze `tensileTest001.dat`, `tensileTest002.dat`, `tensileTest003.dat`.

- Literatur
- Matlab-Beispiele
- Ergebnisse der SQL-Abfragen für die Beispiel-Datenbank
- Beispieldaten

1. U. Stein: Programmieren mit MATLAB  
Carl Hanser Verlag, 6. Aufl. 2017, ISBN: 978-3446448643
2. Angermann, Beuschel, Rau, Wohlfarth: Matlab - Simulink- Stateflow  
Oldenbourg, 10. Aufl 2020, ISBN: 978-3110641073
3. S. Attaway: MATLAB: A Practical Introduction to Programming and Problem Solving  
Butterworth-Heinemann, 5. Aufl. 2018, ISBN: 978-0128154793
4. W. D. Pietruszka, M. Glöckler: MATLAB und Simulink in der Ingenieurpraxis,  
Springer Vieweg, 5. Aufl. 2021, ISBN: 978-3658297398
5. B. R. Hunt, R. L. Lipsman et. al.: A Guide to MATLAB  
Cambridge University Press, 3. Aufl. 2014, ISBN: 978-1107662223
6. M. Kofler: Datenbanksysteme — Das umfassende Lehrbuch  
Rheinwerk Computing, 2022, ISBN: 978-3836284226

- bild26.m
- bild28.m
- computeEigenvalues.m
- createDemoTruss.m
- createEnvelope.m
- createMatrices.m
- createSineWave.m
- demo2dfunc.m
- erzwungen.m
- erzwungenp.m
- getNumber.m
- glocke.m
- loadTruss1.m
- loadTruss.m
- matschwing2d.m
- plotModeAnimation1.m
- plotModeAnimation.m
- plotMode.m
- plotTruss1.m
- plotTruss.m
- radioaktiv.m
- radiokette.m
- saveTruss1.m
- saveTruss.m
- schwing2d.m
- schwingNd.m
- showPapers.m
- solveVibrationODE.m
- spektrum.m
- trapez1.m
- trapez2.m
- trapez.m
- zeichneFunktion.m

```
function bild26()
% Matrixwerte
m1 = 6.0e3; m2 = 6.0e3; m3 = 1.0e3;
c1 = 3.0e6; c2 = 3.0e6; c3 = 1.0e6;
M = diag([m1 m2 m3]);
C = [c1 + c2, -c2, 0; -c2, c2 + c3, -c3; 0, -c3, c3];

% Anfangsbedingungen
x0 = zeros(3,1);
v0 = [0; 0; 10];
y0 = [x0; v0];

% DGL lösen
tEnd = 2;
myfunc = @(t, y) schwingNd(t, y, M, C);
[t, y] = ode45(myfunc, [0 tEnd], y0);

% Ergebnis plotten
plot(t, y(:,1), t, y(:,2), t, y(:,3));
legend("x_1", "x_2", "x_3", "Location", "best");

F = getframe(gcf);
imwrite(F.cdata, "bild26.png");
```

```
function bild28()
% Erzeugung von Bild 28 und 29

% Matrixwerte
M = eye(4);
C = [1,0,0,0; 0,2,0,-1; 0,0,1,0; 0,-1,0,2];

% Anfangsbedingungen
x0 = [0.2; 0; -0.1; 0.2];
v0 = zeros(4, 1);

% DGL lösen
tEnd = 20;
myfunc = @(t, y) schwingNd(t, y, M, C);
[t, y] = ode45(myfunc, [0, tEnd], [x0; v0]);

% Ergebnisse plotten
subplot(2,1,1)
plot(t, y(:,1), t, y(:,2));
title("Masse 1", "FontSize", 14)
legend("x", "y", "Location", "best");
subplot(2,1,2)
plot(t, y(:,3), t, y(:,4));
title("Masse 2", "FontSize", 14)
legend("x", "y", "Location", "best");

F = getframe(gcf);
imwrite(F.cdata, "bild28.png");

% Bahnkurven plotten
subplot(2,1,1)
plot(y(:,1), y(:,2));
title("Masse 1", "FontSize", 14)
subplot(2,1,2)
plot(y(:,3), y(:,4));
title("Masse 2", "FontSize", 14)
axis([-0.2, 0.2, -0.2, 0.2])

F = getframe(gcf);
imwrite(F.cdata, "bild29.png");
```

## 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);
```

# createDemoTruss.m



```
function truss = createDemoTruss()
% erzeugt eine Truss-Struktur für eine einfaches Brückenmodell
% Felder:
% x0 Koordinaten der Punkte im Gleichgewicht
% A Matrix der Verbindungen, symmetrisch, auf der Diagonalen 0
% N Anzahl der dynamischen (nicht fixierten) Knoten
% m Masse eines Knotens
% c Steifigkeit einer Verbindung

truss.N = 4;
truss.m = 1;
truss.c = 1;
truss.x0 = [1 2 2 1 0 3; 0 0 1 1 0 0];
truss.A = [0 1 1 1 1 0; 1 0 1 0 0 1; 1 1 0 1 0 1; ...
 1 0 1 0 1 0; 1 0 0 1 0 0; 0 1 1 0 0 0];
```

# createEnvelope.m



```
function [y, t] = createEnvelope(TA, TS, TR, T)
% erzeugt eine Hüllkurve mit einer Anstiegszeit (Attack-Phase), einer
% Dauerzeit (Sustain-Phase) und einer Abklingzeit (Release-Phase)
% Parameter:
% TA Dauer der Attack-Phase in s
% TS Dauer der Sustain-Phase in s
% TR Dauer der Release-Phase in s
% T Gesamtdauer in s
% Ergebnisse:
% y Hüllkurve
% t Zeitwerte

dt = 1/44100; % feste Samplezeit 44.1 kHz
t = 0:dt:T;

% Berechnen der Index-Werte an den Grenzen
indexA = floor(TA/dt); % Ende der Attack-Phase
indexS = floor((TA+TS)/dt); % Ende der Sustain-Phase
indexR = floor((TA+TS+TR)/dt); % Ende der Release-Phase

% zunächst überall auf 0
y = zeros(size(t));

% Attack-Phase
index = 1:indexA;
tA = t(index);
y(index) = tA/TA;

% Sustain-Phase
index = (indexA + 1):(indexS);
y(index) = 1.0;

% Release-Phase
index = (indexS + 1):indexR;
tR = t(index);
y(index) = (TA + TS + TR - tR)/TR;
```

# 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);
```

## createSineWave.m



```
function [y, t] = createSineWave(f, A, T)
% erzeugt eine Sinusschwingung
% Parameter:
% f Frequenz in Hz
% A Amplitude
% T Tondauer in s
% Ergebnisse:
% y Schwingung
% t Zeitwerte

fS = 44100; % feste Sample-Frequenz in Hz
dt = 1/fS; % Zeitintervall zwischen zwei Samplewerten
t = 0:dt:T; % Zeitwerte

omega = 2*pi*f;
y = A*sin(omega*t);
```

## 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];
```

## getNumber.m



```
function n = getNumber(min, max)
% erfragt vom Benutzer eine Zahl zwischen min und max

prompt = sprintf("Geben Sie eine Zahl zwischen %d und %d ein: ", min, max);
n = input(prompt);

while ~isnumeric(n) || (n > max) || (n < min)
 fprintf("Falsche Eingabe!\n");
 n = input(prompt);
end
```

## glocke.m



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

# loadTruss1.m



```
function [x0, A, N, m, c] = loadTruss1(filename)
% lädt Daten eines Fachwerks
% Parameter:
% filename Name der Datei (ohne Endung .truss)
% Ergebniswerte:
% x0 Gleichgewichtskordinaten 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");

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

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

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

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

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

fclose(fid);
```

# loadTruss.m



```
function truss = loadTruss(filename)
% lädt eine Fachwerkstruktur
% Parameter:
% filename Name der Datei (ohne Endung .truss)
% Felder von truss:
% x0 Gleichgewichtskordinaten 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);
```

## matschwing2d.m



```
function dy = matschwing2d(t, y, M, C)
% rechte Seite der DGL bei zweidimensionaler Schwingerkette

x = y(1:2);
v = y(3:4);
dx = v;
dv = -inv(M)*C*x;
dy = [dx; dv];
```

# plotModeAnimation1.m



```
function F = plotModeAnimation1(axes, xe, truss)
% erstellt eine Animation der Eigenschwingung xe
% Parameter
% axes verwendetes Achsensystem
% xe Eigenvektor der Fachwerk-Schwingung
% truss Fachwerk

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

N = truss.N;
trussT = truss;

for j = 1:nFrames
 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);
 F(j) = getframe(axes);
end
```

# 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
```

# 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")
```

# plotTruss1.m



```
function plotTruss1(x0, A, N)
% zeichnet ein Fachwerk
% Parameter:
% x0 Gleichgewichtskordinaten aller Punkte
% A Verbindungsmatrix (symmetrisch, auf der Diagonalen 0)
% N Zahl der beweglichen Massen

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

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

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

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

% mache Achsensystem quadratisch
axis("equal")

%-----

function plotSquare(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(xi, yi, "r-")
```

```
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-")
```

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

```
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
```

# saveTruss1.m



```
function saveTruss1(x0, A, N, m, c, filename)
% speichert Daten eines Fachwerks
% Parameter:
% x0 Gleichgewichtskoordinaten aller Punkte
% A Verbindungsmatrix (symmetrisch, auf der Diagonalen 0)
% N Zahl der beweglichen Massen
% m Größe einer Masse
% c Steifigkeit eines Balkens
% filename Name der Datei (ohne Endung .truss)

fid = fopen(filename + ".truss", "w");

fprintf(fid, "%s x0\n");
fprintf(fid, "%d %d\n", size(x0));
fprintf(fid, "%7.4f", x0);
fprintf(fid, "\n%% A\n");
fprintf(fid, "%d %d\n", size(A));
fprintf(fid, "%1d", A);
fprintf(fid, "\n%% N\n");
fprintf(fid, "%d", N);
fprintf(fid, "\n%% m\n");
fprintf(fid, "%f", m);
fprintf(fid, "\n%% c\n");
fprintf(fid, "%f\n", c);

fclose(fid);
```

## saveTruss.m



```
function saveTruss(truss, filename)
% speichert eine Fachwerkstruktur
% Parameter:
% truss Fachwerk-Struktur
% filename Name der Datei (ohne Endung .truss)

fid = fopen(filename + ".truss", "w");

fprintf(fid, "%x\n", size(truss.x0));
fprintf(fid, "%7.4f", truss.x0);
fprintf(fid, "\n%% A\n");
fprintf(fid, "%d\n", size(truss.A));
fprintf(fid, "%ld", truss.A);
fprintf(fid, "\n%% N\n");
fprintf(fid, "%d", truss.N);
fprintf(fid, "\n%% m\n");
fprintf(fid, "%f", truss.m);
fprintf(fid, "\n%% c\n");
fprintf(fid, "%f\n", truss.c);

fclose(fid);
```

# schwing2d.m



```
function dy = schwing2d(t, y)
% rechte Seite der DGL bei zweidimensionaler Schwingerkette

% Parameter
m1 = 1.0;
m2 = 1.0;
c1 = 4.0;
c2 = 0.6;
c3 = 4.0;

dy1 = y(3);
dy2 = y(4);
dy3 = (1/m1)*(-(c1+c2)*y(1) + c2*y(2));
dy4 = (1/m2)*(c2*y(1) - (c2+c3)*y(2));

dy = [dy1; dy2; dy3; dy4];
```

## schwingNd.m



```
function dy = schwingNd(t, y, M, C)
% rechte Seite der DGL bei n-dimensionaler Schwingerkette
N = length(y)/2;
x = y(1:N);
v = y((N+1):(2*N));
dx = v;
dv = -inv(M)*C*x;
dy = [dx; dv];
```

# showPapers.m



```
function showPapers(keyword)
% output a list of papers related to the keyword
% e.g. simulation, thermofluid, benchmark, discrete events

myServer = "127.0.0.1";
myUser = "peter";
myPass = "*****"; % replace password!
myDB = "literature";
mySource = "MySQLDataSource";

% connect to mysql database
opts = databaseConnectionOptions("native", "MySQL");
opts = setoptions(opts, "DataSourceName", mySource, ...
 "DatabaseName", myDB, "Server", myServer, "PortNumber", 3306);
[dbOk, message] = testConnection(opts, myUser, myPass);
if ~dbOk
 fprintf("Connection to database '%s' failed!\n", myDB)
 fprintf("message: %s\n", message)
end
saveAsDataSource(opts);
conn = mysql(mySource, myUser, myPass);

% get list of papers
sqlquery = "SELECT paperId FROM " + ...
 "Paper NATURAL JOIN PaperKeyword NATURAL JOIN Keyword " + ...
 "WHERE word = '" + keyword + "'";
data = fetch(conn, sqlquery);
paperIds = data.paperId';

% create formatted output for each paper
for id = paperIds
 sAuth = getAuthorString(conn, id);
 title = getTitle(conn, id);
 fprintf("%s:\n%s\n\n", sAuth, title)
end

close(conn);

%-----
function sAuth = getAuthorString(conn, id)
% returns a formatted author list of paper with given id
sqlquery = "SELECT firstName, name FROM " + ...
 "Author NATURAL JOIN PaperAuthor " + ...
 "WHERE paperId = " + string(id);
data = fetch(conn, sqlquery);
sAuth = sprintf("%s.%s, ", [extract(data.firstName,1), data.name]');
sAuth = extractBefore(sAuth, strlength(sAuth)-1);
%-----
function title = getTitle(conn, id)
% returns the title of paper with given id
sqlquery = "SELECT title FROM Paper WHERE paperId = " + string(id);
data = fetch(conn, sqlquery);
title = data.title;
```

# solveVibrationODE.m



```
function [t, d, v] = solveVibrationODE(M, C, d0, v0, tEnd, dt)
% löst die (Nx2)- dimensionale Schwingungs-DGL für das Zeitintervall
% [0 tEnd]
% Parameter
% M Massenmatrix (2N x 2N)
% C Steifigkeitsmatrix (2N x 2N)
% d0 Anfangsauslenkungen (2 x N)
% v0 Anfangsgeschwindigkeiten (2 x N)
% tEnd Endzeit
% dt optional: Ausgabe-Schrittweite
% Ergebniswerte:
% t Zeitwerte (NT-Vektor, vom Solver bestimmt)
% d Auslenkungen (NT x 2 x N)
% v Geschwindigkeiten (NT x 2 x N)

% mache aus d0 und v0 einen langen Vektor
N = size(d0,2);
y0 = reshape([d0, v0], 4*N, 1);

% löse DGL
odeFunc = @(t,y) vibrationODE(t, y, M, C);
if (nargin < 6)
 [t, y] = ode45(odeFunc, [0 tEnd], y0);
else
 [t, y] = ode45(odeFunc, 0:dt:tEnd, y0);
end

% pflücke den Ergebnisvektor auseinander
NT = length(t);
d = y(:, 1:(2*N));
v = y(:, (2*N+1):end);
% und ordne ihn um
d = reshape(d, NT, 2, N);
v = reshape(v, NT, 2, N);

%-----

function dy = vibrationODE(t, y, M, C)
% Zustandsform der Schwingungsgleichung
% $M \dot{x} + C x = 0$
% Dabei ist
% $y = [x; v]$

N = length(y)/2;
x = y(1:N);
v = y(N+1:end);

dy = [v; -inv(M)*C*x];
```

## spektrum.m



```
function [Y, f] = spektrum(y, Fs)
% berechnet das Spektrum der Zeitreihe y (bei gegebener Samplerate fS)
% gibt die Spektralfunktion Y und die zugehörigen Frequenzwerte f zurück

N = length(y);
T = N/Fs; % Zeitdauer von y
X = fft(y);
Y = abs(X(1:floor(N/2)))/N;
f = (0:(N/2)-1)/T;
```

# 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));
```

```
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");
```

# Ergebnisse der SQL-Abfragen für die Beispiel-Datenbank



- Beispiel 1

Abfrage

```
SELECT title, page1, page2, year FROM Paper
ORDER BY year DESC, title;
```

Ergebnis

| title                                                                                                                           | page1 | page2 | year |
|---------------------------------------------------------------------------------------------------------------------------------|-------|-------|------|
| Implementing Thermodynamic Cyclic Processes Using the DLR ThermoFluid Stream Library                                            | 3     | 10    | 2023 |
| Solving ARGESIM Benchmark CP2 'Parallel and Distributed Simulation' with Open MPI and Matlab PCT – Lattice Boltzmann Simulation | 93    | 100   | 2023 |
| Implementing Standard Examples with NSA-DEVS                                                                                    | 195   | 202   | 2022 |
| Regularization of the Logarithmic Mean for Robust Numerical Simulation                                                          | 529   | 533   | 2022 |
| The DLR ThermoFluid Stream Library                                                                                              | 3790  | 3810  | 2022 |
| NSA-DEVS: Combining Mealy behaviour and Causality                                                                               | 73    | 80    | 2021 |
| Aufgabenorientierte Multi-Robotersteuerungen auf Basis des SBC-Frameworks und DEVS                                              | 1     | 181   | 2020 |
| Beschleunigung eines Reinforcement-Learning-Algorithmus durch Parallelverarbeitung für Robotikanwendungen                       | 49    | 52    | 2019 |
| Non-standard Queuing Policies: Definition of ARGESIM Benchmark C22                                                              | 111   | 115   | 2019 |
| Implementing the Argesim C21 benchmark with Modelica components                                                                 | 197   | 202   | 2018 |
| A Modelica Library for MultiBond Graphs and its Application in 3D-Mechanics                                                     | 1     | 156   | 2006 |
| The Modelica Multi-bond Graph Library                                                                                           | 559   | 568   | 2006 |
| Parallel DEVS: A parallel, hierarchical, modular modeling formalism and its distributed simulator                               | 55    | 68    | 1996 |

- Beispiel 2

Abfrage

```
SELECT title, page1, page2, year FROM Paper
WHERE year >= 2020;
```

Ergebnis

| title                                                                                                                           | page1 | page2 | year |
|---------------------------------------------------------------------------------------------------------------------------------|-------|-------|------|
| Solving ARGESIM Benchmark CP2 'Parallel and Distributed Simulation' with Open MPI and Matlab PCT – Lattice Boltzmann Simulation | 93    | 100   | 2023 |
| Implementing Standard Examples with NSA-DEVS                                                                                    | 195   | 202   | 2022 |
| The DLR ThermoFluid Stream Library                                                                                              | 3790  | 3810  | 2022 |
| Regularization of the Logarithmic Mean for Robust Numerical Simulation                                                          | 529   | 533   | 2022 |
| Implementing Thermodynamic Cyclic Processes Using the DLR ThermoFluid Stream Library                                            | 3     | 10    | 2023 |
| Aufgabenorientierte Multi-Robotersteuerungen auf Basis des SBC-Frameworks und DEVS                                              | 1     | 181   | 2020 |
| NSA-DEVS: Combining Mealy behaviour and Causality                                                                               | 73    | 80    | 2021 |

- Beispiel 3

Abfrage

```
SELECT title, page1, page2, year FROM Paper
WHERE title LIKE "%NSA-DEVS%";
```

Ergebnis

| title                                             | page1 | page2 | year |
|---------------------------------------------------|-------|-------|------|
| Implementing Standard Examples with NSA-DEVS      | 195   | 202   | 2022 |
| NSA-DEVS: Combining Mealy behaviour and Causality | 73    | 80    | 2021 |

- Beispiel 4

Abfrage

```
SELECT title, page2 - page1 + 1, year FROM Paper
WHERE title LIKE "%NSA-DEVS%"
ORDER BY year;
```

Ergebnis

| title                                             | page2 - page1 + 1 | year |
|---------------------------------------------------|-------------------|------|
| NSA-DEVS: Combining Mealy behaviour and Causality | 8                 | 2021 |
| Implementing Standard Examples with NSA-DEVS      | 8                 | 2022 |

- Beispiel 5

Abfrage

```
SELECT title, name FROM Paper NATURAL JOIN Publication;
```

Ergebnis

| title                                                                                                                           | name                                                |
|---------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------|
| Solving ARGESIM Benchmark CP2 'Parallel and Distributed Simulation' with Open MPI and Matlab PCT – Lattice Boltzmann Simulation | SNE Simulation Notes Europe                         |
| Implementing Standard Examples with NSA-DEVS                                                                                    | SNE Simulation Notes Europe                         |
| Implementing the Argesim C21 benchmark with Modelica components                                                                 | 7. Workshop der ASIM/GI-Fachgruppen STS/GMMS        |
| Non-standard Queuing Policies: Definition of ARGESIM Benchmark C22                                                              | SNE Simulation Notes Europe                         |
| Beschleunigung eines Reinforcement-Learning-Algorithmus durch Parallelverarbeitung für Robotikanwendungen                       | 8. Workshop der ASIM/GI-Fachgruppen STS/GMMS        |
| A Modelica Library for MultiBond Graphs and its Application in 3D-Mechanics                                                     | Master Thesis                                       |
| The DLR ThermoFluid Stream Library                                                                                              | Electronics                                         |
| Regularization of the Logarithmic Mean for Robust Numerical Simulation                                                          | IFAC-PapersOnLine                                   |
| Implementing Thermodynamic Cyclic Processes Using the DLR Thermofluid Stream Library                                            | 10. Workshop der ASIM/GI-Fachgruppen STS/GMMS/EDU   |
| The Modelica Multi-bond Graph Library                                                                                           | 5th International Modelica Conference               |
| Aufgabenorientierte Multi-Robotersteuerungen auf Basis des SBC-Frameworks und DEVS                                              | PhD Thesis                                          |
| Parallel DEVS: A parallel, hierarchical, modular modeling formalism and its distributed simulator                               | Transactions of the Society for Computer Simulation |
| NSA-DEVS: Combining Mealy behaviour and Causality                                                                               | SNE Simulation Notes Europe                         |

- Beispiel 6

Abfrage

```
SELECT title, name AS journal
FROM Paper NATURAL JOIN Publication
ORDER BY journal;
```

Ergebnis

| title                                                                                                                           | journal                                             |
|---------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------|
| Implementing Thermodynamic Cyclic Processes Using the DLR ThermoFluid Stream Library                                            | 10. Workshop der ASIM/GI-Fachgruppen STS/GMMS/EDU   |
| The Modelica Multi-bond Graph Library                                                                                           | 5th International Modelica Conference               |
| Implementing the Argesim C21 benchmark with Modelica components                                                                 | 7. Workshop der ASIM/GI-Fachgruppen STS/GMMS        |
| Beschleunigung eines Reinforcement-Learning-Algorithmus durch Parallelverarbeitung für Robotikanwendungen                       | 8. Workshop der ASIM/GI-Fachgruppen STS/GMMS        |
| The DLR ThermoFluid Stream Library                                                                                              | Electronics                                         |
| Regularization of the Logarithmic Mean for Robust Numerical Simulation                                                          | IFAC-PapersOnLine                                   |
| A Modelica Library for MultiBond Graphs and its Application in 3D-Mechanics                                                     | Master Thesis                                       |
| Aufgabenorientierte Multi-Robotersteuerungen auf Basis des SBC-Frameworks und DEVS                                              | PhD Thesis                                          |
| Solving ARGESIM Benchmark CP2 'Parallel and Distributed Simulation' with Open MPI and Matlab PCT – Lattice Boltzmann Simulation | SNE Simulation Notes Europe                         |
| Implementing Standard Examples with NSA-DEVS                                                                                    | SNE Simulation Notes Europe                         |
| Non-standard Queuing Policies: Definition of ARGESIM Benchmark C22                                                              | SNE Simulation Notes Europe                         |
| NSA-DEVS: Combining Mealy behaviour and Causality                                                                               | SNE Simulation Notes Europe                         |
| Parallel DEVS: A parallel, hierarchical, modular modeling formalism and its distributed simulator                               | Transactions of the Society for Computer Simulation |

• Beispiel 7

Abfrage

```
SELECT COUNT(*) as count, SUM(page2-page1+1) as totalPages, year FROM Paper
GROUP BY year ORDER BY year;
```

Ergebnis

| count | totalPages | year |
|-------|------------|------|
| 1     | 14         | 1996 |
| 2     | 166        | 2006 |
| 1     | 6          | 2018 |
| 2     | 9          | 2019 |
| 1     | 181        | 2020 |
| 1     | 8          | 2021 |
| 3     | 34         | 2022 |
| 2     | 16         | 2023 |

• Beispiel 8

Abfrage

```
SELECT name as journal, COUNT(*) as count
FROM Paper NATURAL JOIN Publication
GROUP BY journal
ORDER BY count DESC;
```

Ergebnis

| <b>journal</b>                                      | <b>count</b> |
|-----------------------------------------------------|--------------|
| SNE Simulation Notes Europe                         | 4            |
| 7. Workshop der ASIM/GI-Fachgruppen STS/GMMS        | 1            |
| 8. Workshop der ASIM/GI-Fachgruppen STS/GMMS        | 1            |
| Master Thesis                                       | 1            |
| Electronics                                         | 1            |
| IFAC-PapersOnLine                                   | 1            |
| 10. Workshop der ASIM/GI-Fachgruppen STS/GMMS/EDU   | 1            |
| 5th International Modelica Conference               | 1            |
| PhD Thesis                                          | 1            |
| Transactions of the Society for Computer Simulation | 1            |

- Beispiel 9

#### Abfrage

```
SELECT name AS publication , SUM(page2-page1+1) AS totalPages
FROM Paper NATURAL JOIN Publication
GROUP BY publication
ORDER BY totalPages DESC;
```

#### Ergebnis

| <b>publication</b>                                  | <b>totalPages</b> |
|-----------------------------------------------------|-------------------|
| PhD Thesis                                          | 181               |
| Master Thesis                                       | 156               |
| SNE Simulation Notes Europe                         | 29                |
| Electronics                                         | 21                |
| Transactions of the Society for Computer Simulation | 14                |
| 5th International Modelica Conference               | 10                |
| 10. Workshop der ASIM/GI-Fachgruppen STS/GMMS/EDU   | 8                 |
| 7. Workshop der ASIM/GI-Fachgruppen STS/GMMS        | 6                 |
| IFAC-PapersOnLine                                   | 5                 |
| 8. Workshop der ASIM/GI-Fachgruppen STS/GMMS        | 4                 |

# Beispieldaten



- demo1.dat
- bruecke.truss
- demo.chain
- long.chain
- stoerung.dat
- ton1.mp3
- ton2.mp3
- trussGUI1.mlapp
- trussGUI2.mlapp
- trussGUI.mlapp
- literature.sql
- ex02.dat
- ex04.dat
- tensileTest001.dat
- tensileTest002.dat
- tensileTest003.dat