Numerische Mathematik

Peter Junglas 24. 01. 2022

Inhaltsverzeichnis

Fehlerarten

- Numerische Mathematik:

  Entwicklung und theoretische Analyse von Verfahren (Algorithmen) zur zahlenmäßigen Lösung mathematischer Fragestellungen

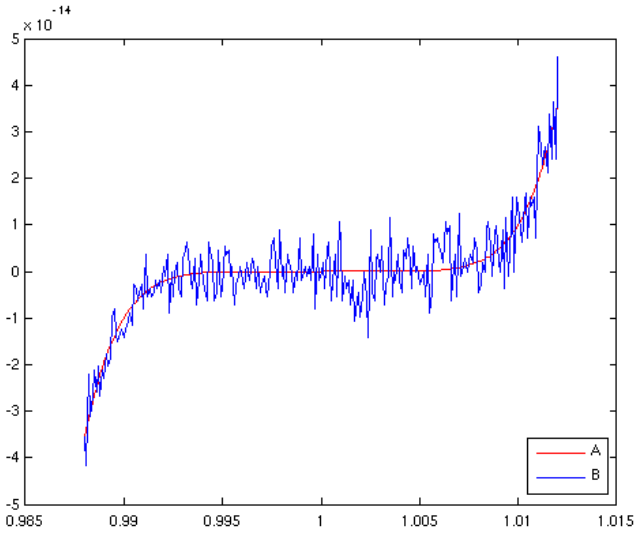
  Ansprüche an Algorithmen
  - endlich (selten) oder zumindest konvergent (Normalfall)
  - schnell = Zahl der Operationen (für eine gewünschte Genauigkeit) möglichst klein
  - stabil = ähnliche Ergebnisse bei kleinen Störungen der Eingangsdaten

- Plot eines Polynoms mit Matlab [1, S.42]:

  Beispiel-Polynom

$$\begin{aligned} y &= (x-1)^7 && \text{(A)} \\ &= x^7 - 7x^6 + 21x^5 - 35x^4 + 35x^3 - 21x^2 + 7x - 1 && \text{(B)} \end{aligned}$$

  direkte Berechnung als Potenz (A) oder ausmultipliziert (B) mit Matlab liefert



- wichtige Fehlerquellen:

  Abbruchfehler bei iterativen Verfahren

  Rundungsfehler durch ungenaue Darstellung der Zahlen (Computer-Arithmetik)

  Datenfehler durch ungenaue Eingangswerte

- Fehlermaße:

  sei x ein genauer Wert, $\tilde{x}$ ein Näherungswert

  absoluter Fehler

$$\varepsilon_{\text{abs}}(x) := |x - \tilde{x}|$$

  relativer Fehler (für x ≠ 0)

$$\varepsilon_{\text{rel}}(x) := \frac{|x - \tilde{x}|}{|x|}$$

  absoluter Fehler bei der Addition

$$\begin{aligned} \varepsilon_{\text{abs}}(x+y) &= |x + y - \tilde{x} - \tilde{y}| \\ &= |(x - \tilde{x}) + (y - \tilde{y})| \\ &\leq |x - \tilde{x}| + |y - \tilde{y}| \\ &= \varepsilon_{\text{abs}}(x) + \varepsilon_{\text{abs}}(y) \end{aligned}$$

relativer Fehler bei der Multiplikation

$$\begin{aligned}\varepsilon_{\text{rel}}(xy) &= \frac{|xy-\tilde{x}\tilde{y}|}{|xy|}\\ &= \frac{|(x-\tilde{x})y+\tilde{x}(y-\tilde{y})|}{|x||y|}\\ &\leq \frac{|x-\tilde{x}||y|}{|x||y|}+\frac{|\tilde{x}||y-\tilde{y}|}{|x||y|}\\ &= \varepsilon_{\text{rel}}(x)+\frac{|\tilde{x}|}{|x|}\varepsilon_{\text{rel}}(y)\\ &\approx \varepsilon_{\text{rel}}(x)+\varepsilon_{\text{rel}}(y)\end{aligned}$$

in vielen Anwendungen ist der relative Fehler entscheidend!

- Auslöschung:

  relativer Fehler bei der Addition

  $$\begin{aligned}\varepsilon_{\text{rel}}(x+y) &= \frac{|x+y-\tilde{x}-\tilde{y}|}{|x+y|}\\ &\leq \frac{|x-\tilde{x}|}{|x+y|}+\frac{|y-\tilde{y}|}{|x+y|}\\ &= \frac{|x|}{|x+y|}\varepsilon_{\text{rel}}(x)+\frac{|y|}{|x+y|}\varepsilon_{\text{rel}}(y)\end{aligned}$$

  problematisch für

  $$|x+y| \ll 1$$

  also bei Subtraktion etwa gleich großer Zahlen

- Lösung quadratischer Gleichungen:

  Beispiel

  $$x^2+62x+1=0$$

  Lösungsformel

  $$x_{1,2} = -\frac{p}{2} \pm \sqrt{\frac{p^2}{4}-q}$$

  liefert als erste Lösung

  $$x_1 = -31+\sqrt{960} \approx -0.01613$$

  bei Rechnung mit 4 signifikanten Stellen aber

  $$\tilde{x}_1 = -31+30.98 = -0.02$$

  mit einem relativen Fehler

  $$\varepsilon_{\text{rel}}(x_1) = \left|\frac{x_1-\tilde{x}_1}{x_1}\right| \approx 23.97\%$$

  also: möglichst nie etwa gleich große Zahlen subtrahieren!

- Alternativer Algorithmus:

  Lösungsformel umstellen

  $$\begin{aligned}x_1 &= \left(-\frac{p}{2}+\sqrt{\frac{p^2}{4}-q}\right)\cdot\frac{-\frac{p}{2}-\sqrt{\frac{p^2}{4}-q}}{-\frac{p}{2}-\sqrt{\frac{p^2}{4}-q}}\\ &= \frac{q}{-\frac{p}{2}-\sqrt{\frac{p^2}{4}-q}}\end{aligned}$$

Rechnung auf 4 signifikanten Stellen

$$\tilde{x}_1 = \frac{1}{-31 - \sqrt{960}} \approx \frac{1}{-61.98} \approx -0.01613$$

liefert also volle 4 signifikanten Stellen Genauigkeit

- Kernthemen dieses Kurses:

  wichtige Algorithmen in Beispielen

  Sätze zu Konvergenz- und Stabilitätseigenschaften, aber keine Beweise

  Implementierungen in Matlab

- Gleitpunktzahlen:

  Zahlen im Rechner
  - nur endlich viel Speicherplatz pro Zahl
  - → reelle Zahlen approximiert durch endlich viele Werte

  Darstellung
  - bezogen auf eine Basis (fast immer 2, manchmal 10)
  - endlich viele Ziffern (**Mantisse**) und ein Exponent
  - ermöglicht sehr kleine und sehr große Zahlen

  Beispiele
  - $3.561 * 10^{-12}$, $124.2 * 10^{3}$
  - $1.101001101 * 2^{-3}$, $0.011001011 * 2^{15}$

- Normalisierte Gleitpunktzahlen:

  genau eine Ziffer, ≠ 0, vor dem Dezimalpunkt

  Beispiele
  - $3.242 * 10^{5}$ statt $324.2 * 10^{3}$ oder 324200
  - $1.010101 * 2^{-1}$ statt 0.1010101

  Vorteil: keine Zweideutigkeit (Ausnahme: 0!)

  Spezialität im 2-er-System
  - erste Ziffer ist immer 1
  - muss nicht gespeichert werden
  - → ein extra Bit zur Erhöhung der Genauigkeit (**hidden bit**)

- Gleitpunktsysteme:

  beschrieben durch 4 Größen

β	Basis
t	Zahl der Ziffern = Mantissenlänge
e_{min}	kleinster Exponent
e_{max}	größter Exponent

  Kennwerte

realmin	kleinster positiver Wert
realmax	größter positiver Wert
ε	Maschinengenauigkeit
N_{gesamt}	Anzahl der Zahlen

  ε = betragsmäßig kleinste Zahl, die man zu 1 addieren kann, so dass sich der Wert ändert

  Beispiel A: β = 10, t = 2, emin = -2, emax = 3

realmin	$1.0 * 10^{-2}$
realmax	$9.9 * 10^{3}$
ε	1.1 - 1.0 = 0.1
N_{gesamt}	2*9*10*(5+1) + 1 = 1081

  Beispiel B: β = 2, t = 4, emin = -3, emax = 3

realmin	$1.000_2 * 2^{-3} = 1/8$
realmax	$1.111_2 * 2^3 = 15$
ε	$1.001_2 - 1.000_2 = 1/8$
N_{gesamt}	$2*1*2^3*(6+1) + 1 = 113$

allgemein

realmin	$\beta^{e_{min}}$
realmax	$(1 - \beta^{-t})*\beta^{e_{max} + 1}$
ε	$\beta^{(1-t)}$
N_{gesamt}	$2*(\beta - 1)*\beta^{(t-1)}*(e_{max} - e_{min} + 1) + 1$

- Verteilung der Gleitpunktzahlen:

  positive Zahlen für Beispiel B

  - linear

  - logarithmisch

  negative Zahlen durch Spiegelung nach links → beachte große Lücke bei 0

- Runden:

  reelle Zahl x → nächstgelegene Fließkommazahl fl(x)

  Bereichsüberschreitung

  - x > realmax → fl(x) = ∞ (**Overflow**)
  - x < -realmax → fl(x) = -∞ (**Overflow**)
  - |x| < realmin → fl(x) = 0 (**Underflow**)

  verschiedene Spezialregeln, wenn x genau zwischen zwei Werten

  - zur größeren Zahl
  - zur betragsmäßig größeren Zahl
  - so, dass letzte Mantissenzahl gerade ist (**round to even**)

  round to even hat beweisbare Vorteile (kein langsames Aufsteigen durch Runden)

- IEEE 754-Standard

  Festlegung des Gleitpunktsystems und der genauen Bedeutung jedes Bits für 32-Bit-Zahlen (**float**) und 64-Bit-Zahlen (**double**)

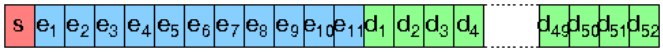
  für double (Standard heutiger Hardware)

  - $\beta = 2$, $t = 53$, $e_{max} = 1023$, $e_{min} = -1022$
  - round to even
  - hidden bit

  Kennzahlen

realmin	2.2251e-308
realmax	1.7977e+308
ε	2.2204e-16
N_{gesamt}	1.8429e+19 (vgl. 2^{64} = 1.8447e+19)

Festlegung der Bits



- 1 Bit Vorzeichen s
- 11 Bit Exponent, als positive ganze Zahl E
- 52 Bit Mantisse (ohne das hidden Bit), als positive ganze Zahl M

Interpretation des Bitmusters

$$x = (-1)^s(1 + M \cdot 2^{-52}) \cdot 2^{E-1023}$$

spezielle Bitmuster

E = 0, M = 0	0
E = 2047, M = 0	± ∞
E = 2047, M != 0	NaN

Ausgabe des Bitmusters (hexadezimal) in Matlab mit

```
format hex
```

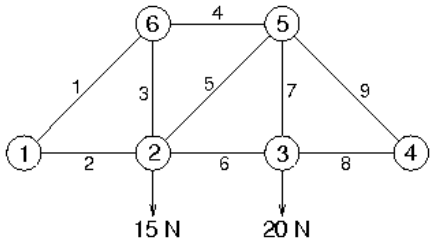
- Aufgaben:

  Aufgabe 1

  Aufgabe 2

- Berechnung eines idealen Fachwerks:

  Beispiel mit 6 Gelenken und 9 Stäben



  3 Auflagekräfte für statische Bestimmtheit
  - Gelenk 1 horizontal und vertikal fixiert
  - Gelenk 4 vertikal fixiert

  an jedem Gelenk Kräftegleichgewicht (2-dim), daher mit $\alpha = 1/\sqrt{2}$

$$
\begin{aligned}
G2 \qquad f_2 &= f_6 + \alpha f_5 \\
f_3 + \alpha f_5 &= 15\,\mathrm{N} \\
\\
G3: \qquad f_6 &= f_8 \\
f_7 &= 20\,\mathrm{N} \\
\\
G4: \qquad f_8 + \alpha f_9 &= 0 \\
\\
G5: \qquad f_4 + \alpha f_5 &= \alpha f_9 \\
\alpha f_5 + f_7 + \alpha f_9 &= 0 \\
\\
G6: \qquad \alpha f_1 &= f_4 \\
\alpha f_1 + f_3 &= 0
\end{aligned}
$$

  ergibt lineares Gleichungssystem mit 9 Gleichungen für 9 Unbekannte

- Existenz und Eindeutigkeit von Lösungen:
  - Sei A eine n x n-Matrix, b ein n-elementiger Spaltenvektor. Das lineare Gleichungssystem

    A x = b

    ist genau dann lösbar, wenn

    rang(A) = rang(A | b)

    Die Lösung ist genau dann eindeutig, wenn A nicht singulär ist (d.h. rang(A) = n oder äquivalent det(A) ≠ 0)
  - Beweis: Jedes Buch über lineare Algebra.

- Gaußscher Algorithmus:
  - Beispielsystem

$$
\begin{pmatrix} \boxed{2} & -3 & 1 \\ 1 & -4 & 0 \\ -3 & 1 & 1 \end{pmatrix}
\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}
=
\begin{pmatrix} 4 \\ 3 \\ -11 \end{pmatrix}
$$

  - 1. Schritt: x_1 eliminieren in 2. und 3. Gleichung. Koeffizient von x_1 ist das 1. Pivot-Element. 1. Gleichung mit 1/2 multiplizieren und von 2. Gleichung subtrahieren, dann 1. Gleichung mit (-3/2) multiplizieren und von der 3. Gleichung subtrahieren. Ergebnis:

$$\begin{pmatrix} 2 & -3 & 1 \\ 0 & \boxed{-\frac{5}{2}} & -\frac{1}{2} \\ 0 & -\frac{7}{2} & \frac{5}{2} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 4 \\ 1 \\ -5 \end{pmatrix}$$

2. Schritt: x_2 eliminieren in 3. Gleichung.Koeffizient von x_2 ist das 2. Pivot-Element. 2. Gleichung mit 7/5 multiplizieren und von 3. Gleichung subtrahieren. Ergebnis:

$$\begin{pmatrix} 2 & -3 & 1 \\ 0 & -\frac{5}{2} & -\frac{1}{2} \\ 0 & 0 & \frac{16}{5} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 4 \\ 1 \\ -\frac{32}{5} \end{pmatrix}$$

- Aufgrund der Dreiecksgestalt der Matrix kann das System nun von unten nach oben gelöst werden (**Rückwärtssubstitution**)

$$\begin{aligned} \frac{16}{5}x_3 &= -\frac{32}{5} \\ \Rightarrow \quad x_3 &= -2 \end{aligned}$$

in 2. Gleichung einsetzen

$$\begin{aligned} -\frac{5}{2}x_2 - \frac{1}{2}\cdot(-2) &= 1 \\ \Rightarrow \quad x_2 &= 0 \end{aligned}$$

in 1. Gleichung einsetzen

$$\begin{aligned} 2x_1 - 3\cdot 0 + 1\cdot(-2) &= 4 \\ \Rightarrow \quad x_1 &= 3 \end{aligned}$$

- Einsetzen der Lösung in das ursprüngliche System verifiziert das Ergebnis.

- LU-Zerlegung:
  - obere Dreiecksmatrix vor der Rückwärtssubstitution

$$U = \begin{pmatrix} 2 & -3 & 1 \\ 0 & -\frac{5}{2} & -\frac{1}{2} \\ 0 & 0 & \frac{16}{5} \end{pmatrix}$$

untere Dreiecksmatrix aus den Vorfaktoren beim Eliminieren, mit 1 in der Diagonalen

$$L = \begin{pmatrix} 1 & 0 & 0 \\ \frac{1}{2} & 1 & 0 \\ -\frac{3}{2} & \frac{7}{5} & 1 \end{pmatrix}$$

dann gilt (nachrechnen!)

L U = A

das klappt immer (bis auf Vertauschungen, s. u.)! Beweis z.B. [3]

  - Mit dieser Zerlegung schnelle Lösung für andere rechte Seiten

A x = c

Löse

L y = c

($y_1 = c_1$, in Gleichung für y_2 einsetzen ergibt y_2, ... (**Vorwärtssubstitution**))

Löse durch Rückwärtssubstitution

U x = y

Dann löst x die Ausgangsgleichung

A x = L U x = L y = c

- Falls A symmetrisch und positiv (d.h. (x, Ax) >= 0), kann auch die LU-Zerlegung symmetrisch gemacht werden

  $A = L L^T$ (**Cholesky-Zerlegung**)

- Zahl der Schritte

  berücksichtigt werden +, - * / als jeweils ein Schritt

  Eliminieren der 1. Variablen
  - N Multiplikationen und N Additionen pro Gleichung
  - zusammen also 2 N (N-1)

  analog für die folgenden Variablen, insgesamt also

$$\begin{aligned} 2N(N-1) &+ 2(N-1)(N-2) + \cdots + 2 \cdot 2 \cdot 1 \\ &= 2\sum_{k=1}^{N} k(k-1) \\ &= \frac{2}{3}N(N+1)^2 \\ &= \frac{2}{3}N^3 + O(N^2) \end{aligned}$$

  Rückwärtssubstitution

$$1 + 3 + 5 + \cdots + (2N-1) = N^2$$

  Vorwärtssubstitution analog, aber N Multiplikationen weniger wegen Faktor 1 in der Diagonalen, somit

  N^2 - N

- Was kann schiefgehen:

  Pivot-Element kann 0 sein oder zumindest sehr klein

  Beispiel [1, S.58ff]

$$\begin{pmatrix} \boxed{10} & -7 & 0 \\ -3 & 2.099 & 6 \\ 5 & -1 & 5 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 7 \\ 3.901 \\ 6 \end{pmatrix}$$

  exakte Lösung ist (nachrechnen!)

  $x = (0, -1, 1)^T$

  Rechnung auf 5 signifikante Stellen
  - x_1 eliminieren

$$\begin{pmatrix} 10 & -7 & 0 \\ 0 & \boxed{-0.001} & 6 \\ 0 & 2.5 & 5 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 7 \\ 6.001 \\ 2.5 \end{pmatrix}$$

  - x_2 eliminieren (immer auf 5 signifikante Stellen runden!)

$$\begin{pmatrix} 10 & -7 & 0 \\ 0 & -0.001 & 6 \\ 0 & 0 & 15005 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 7 \\ 6.001 \\ 15004 \end{pmatrix}$$

  Rückwärtssubstitution

$$
\begin{aligned}
x_3 &= 0.99993 \\
-0.001x_2 + 6 \cdot 0.99993 &= 6.001 \\
\Rightarrow \quad x_2 &= -1.4000 \\
10x_1 - 7 \cdot (-1.4) &= 7 \\
\Rightarrow \quad x_1 &= -0.28
\end{aligned}
$$

Ursache: kleiner Pivot-Wert 0.001

- Pivotisierung:
  - Lösung: vertausche Gleichungen oder Variablen oder beides, so dass Pivot-Element betragsmäßig möglichst groß

    übliches Vorgehen: Gleichungen vertauschen, so dass betragsmäßig größtes Element der Spalte nach oben kommt (**Spalten-** oder **partielle Pivotisierung**)
  - im Beispiel

    $$
    \begin{pmatrix} 10 & -7 & 0 \\ 0 & \boxed{2.5} & 5 \\ 0 & -0.001 & 6 \end{pmatrix}
    \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} =
    \begin{pmatrix} 7 \\ 2.5 \\ 6.001 \end{pmatrix}
    $$

    - x_2 eliminieren

      $$
      \begin{pmatrix} 10 & -7 & 0 \\ 0 & 2.5 & 5 \\ 0 & 0 & 6.002 \end{pmatrix}
      \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} =
      \begin{pmatrix} 7 \\ 2.5 \\ 6.002 \end{pmatrix}
      $$

    - Rückwärtssubstitution

      $$
      \begin{aligned}
      x_3 &= 1 \\
      x_2 &= -1 \\
      x_1 &= 0
      \end{aligned}
      $$
  - bei LU-Zerlegung muss die Vertauschung durch eine Permutationsmatrix P berücksichtigt werden

    $L\,U = P\,A$

    P bewirkt eine Vertauschung der Zeilen von A

    bei mehrfachen Vertauschungen P_1, P_2, ..., P_n ergibt sich P als Produkt

    $P = P_n \cdots P_2 \cdot P_1$
  - im Beispiel Vertauschung von Zeile 2 und 3

    $$
    \begin{aligned}
    P &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \\
    \Rightarrow \quad PA &= \begin{pmatrix} 10 & -7 & 0 \\ 5 & -1 & 5 \\ -3 & 2.099 & 6 \end{pmatrix}
    \end{aligned}
    $$

    man liest aus der Rechnung ab

    $$
    \begin{aligned}
    U &= \begin{pmatrix} 10 & -7 & 0 \\ 0 & 2.5 & 5 \\ 0 & 0 & 6.002 \end{pmatrix} \\
    L &= \begin{pmatrix} 1 & 0 & 0 \\ 0.5 & 1 & 0 \\ -0.3 & -4 \cdot 10^{-4} & 1 \end{pmatrix}
    \end{aligned}
    $$

    wobei auch die Werte in L aus früheren Schritten entsprechend vertauscht werden müssen

    damit gilt (nachrechnen!)

    $L\,U = P\,A$

L U = P A

- Rundungsfehler:
  - Unterschied zwischen exakter Lösung x und berechneter Lösung x_* durch Runden

    Residuum

    $$r := b - Ax_*$$

    Fehler

    $$e := x - x_*$$

  - Beispiel

    $$\begin{pmatrix} 1.0781 & 0.780 \\ 0.662 & 0.479 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0.2981 \\ 0.1830 \end{pmatrix}$$

    exakte Lösung

    $$x = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

    Lösung bei Rechnung mit 4 signifikanten Stellen (in jedem Schritt runden!)

    $$x_* = \begin{pmatrix} 0.2765 \\ 0.0000 \end{pmatrix}$$

    Residuum

    $|r| = 4.333 \cdot 10^{-5}$

    aber Fehler

    $|e| = 1.234$

  - Das Residuum ist beim Gaußalgorithmus mit Spalten-Pivotisierung immer klein (Details in [1] und [8]).

- Norm und Kondition einer Matrix

  Fehler bei linearem Gleichungssystem wächst mit "Größe" von A, etwa bei Multiplikation des Systems mit einem großen Faktor

  "Größe von A" wird präzisiert als Norm von A

  $$\|A\| := \max_{x \neq 0} \frac{|Ax|}{|x|} =: M(A)$$

  maximal möglicher Streckungsfaktor M(A) eines Vektors

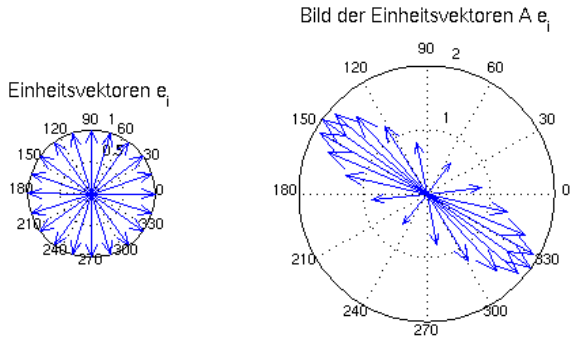
  minimaler Streckungsfaktor analog

  $$\begin{aligned} m(A) &:= \min_{x \neq 0} \frac{|Ax|}{|x|} \\ &= \min_{A^{-1}x \neq 0} \frac{|x|}{|A^{-1}x|} = \frac{1}{\|A^{-1}\|} \end{aligned}$$

  für nicht-singuläres A, sonst 0

  Veranschaulichung

  2x2-Matrix bildet Einheitsvektoren auf (verdrehte) Ellipse ab



M(A) bzw. m(A) = größte bzw. kleinste Halbachse

mehrdimensional analog (Ellipsoid)

Konditionszahl einer nicht-singulären Matrix

$$\mathrm{cond}(A) := \|A\| \cdot \|A^{-1}\| = \frac{M(A)}{m(A)}$$

Eigenschaften der Konditionszahl

$$\begin{aligned} \mathrm{cond}(A) &\geq 1 \\ \mathrm{cond}(\lambda A) &= \mathrm{cond}(A) \\ D = \mathrm{diag}(d_1 \dots d_n) &\Rightarrow \mathrm{cond}(D) = \frac{\max |d_i|}{\min |d_i|} \end{aligned}$$

Ist cond(A) groß, so ist A "fast singulär".

Fehler im Ergebnis wächst mit cond(A) - unabhängig vom Algorithmus!

allgemein gilt

$$\frac{|x - x_*|}{|x|} \leq \mathrm{cond}(A) \cdot \frac{|r|}{|b|}$$

Beweis im Anhang

Faustregel

Jeder Faktor 10 in cond(A) kostet eine signifikante Stelle im Ergebnis.

obige 2d-Beispielmatrix hatte

cond(A) = $4.9 \cdot 10^4$

- Matlab-Routinen:

  direktes Auflösen eines linearen Gleichungssystems (über LU-Zerlegung)

  ```
  x = A \ b
  ```

  explizite LU-Zerlegung

  ```
  [L, U, P] = lu(A)
  ```

  Cholesky-Zerlegung für positives A

  ```
  L = chol(A, "lower")
  ```

  Vorwärts-/Rückwärtssubstitution automatisch bei `x = A \ b` und A Dreiecksmatrix

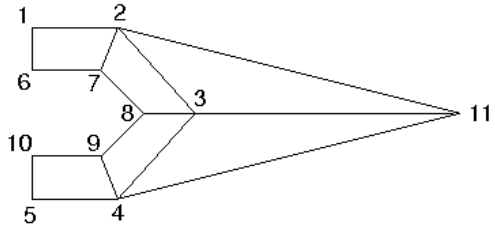
  Konditionszahl

  - `cond(A)` genau, aber aufwändig
  - `condest(A)` schneller Schätzwert

- Dünnbesetzte Matrix (**sparse matrix**)

  Matrix, deren meisten Elemente 0 sind

Beispiel FEM



- Systemmatrix

$$\begin{pmatrix} X & X & 0 & 0 & 0 & X & 0 & 0 & 0 & 0 & 0 \\ X & X & X & 0 & 0 & 0 & X & 0 & 0 & 0 & X \\ 0 & X & X & X & 0 & 0 & 0 & X & 0 & 0 & X \\ 0 & 0 & X & X & X & 0 & 0 & 0 & X & 0 & X \\ 0 & 0 & 0 & X & X & 0 & 0 & 0 & 0 & X & 0 \\ X & 0 & 0 & 0 & 0 & X & X & 0 & 0 & 0 & 0 \\ 0 & X & 0 & 0 & 0 & X & X & X & 0 & 0 & 0 \\ 0 & 0 & X & 0 & 0 & 0 & X & X & X & 0 & 0 \\ 0 & 0 & 0 & X & 0 & 0 & 0 & X & X & X & 0 \\ 0 & 0 & 0 & 0 & X & 0 & 0 & 0 & X & X & 0 \\ 0 & X & X & X & 0 & 0 & 0 & 0 & 0 & 0 & X \end{pmatrix}$$

Probleme mit Speicherplatz und Standardalgorithmen

spezielle iterative Verfahren

wichtig u.a. für Solver von partiellen Differentialgleichung (FEM, Strömungen etc.)

ausgiebige Spezialliteratur, vgl. [11]

- Aufgaben:

  Aufgabe 3

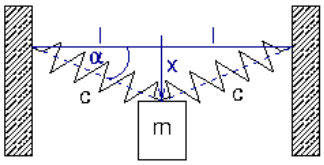
  Aufgabe 4

Lösung nichtlinearer Gleichungen

Grundlegende Methoden bei einer Unbekannten

- Gleichgewichtslage eines Feder-Masse-Systems:

  Masse m sei zwischen zwei Federn eingespannt, die durch das Gewicht symmetrisch nach unten auslenkt werden



  bei waagerechter Position seien beide Federn entspannt (keine Vorspannung)

  Kräftebilanz zur Bestimmung der Gleichgewichtsposition x liefert nicht-lineare Gleichung

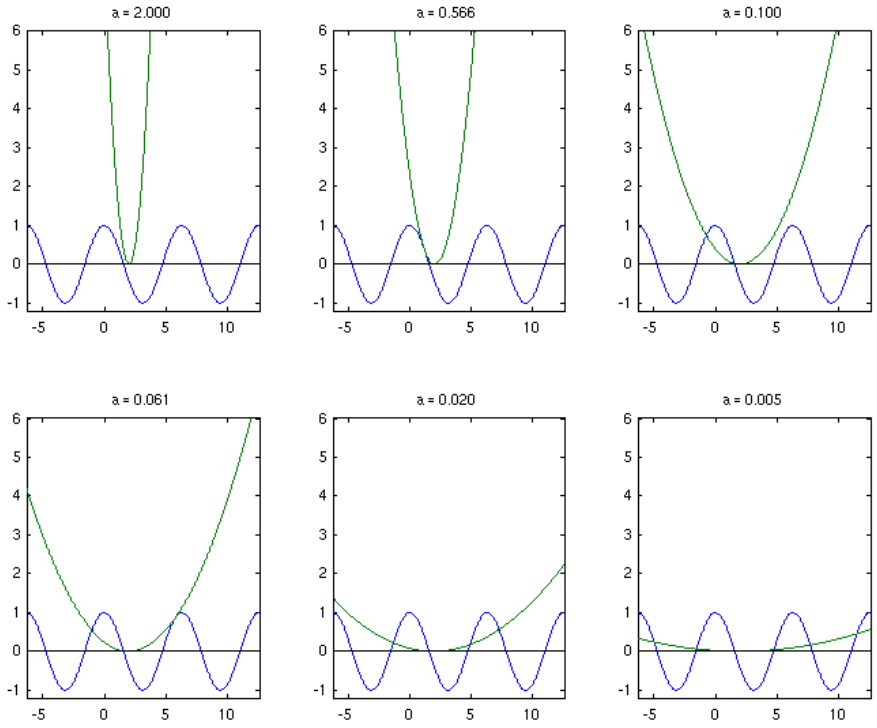
$$2\frac{c}{m}x\left(1-\frac{l}{\sqrt{x^2+l^2}}\right)=g$$

- Existenz und Eindeutigkeit von Lösungen:

  keine allgemeinen Aussagen möglich

  vgl. Beispiel

  $\cos(x) = a\,(x-2)^2$



  leicht in eine Grundform überführbar, z.B.

  - f(x) = 0 (Nullstellenform)
  - f(x) = x (Fixpunktform)

  auch mehrdimensional (x, f Vektoren), aber noch **viel** schwerer

  häufige Ausgangssituation

  - stetige Funktion f
  - Startintervall $[x_1\ x_2]$ mit $f(x_1)\ f(x_2) < 0$ (d.h. verschiedene Vorzeichen)

  Mittelwertsatz → es gibt (mindestens!) eine Lösung

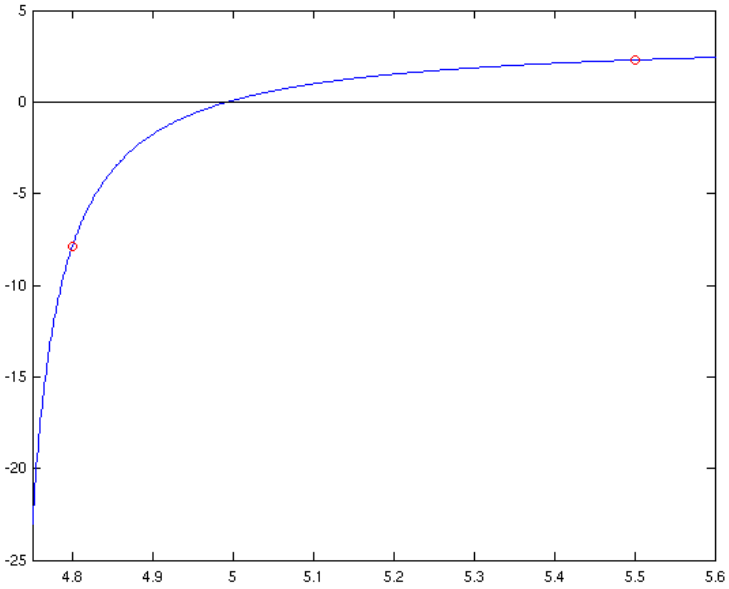
  schwieriger: Nullstellen ohne Vorzeichenwechsel (wie bei $x^2 = 0$)

- Standardbeispiele im Folgenden:

  Beispiel 1 ("normal")

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

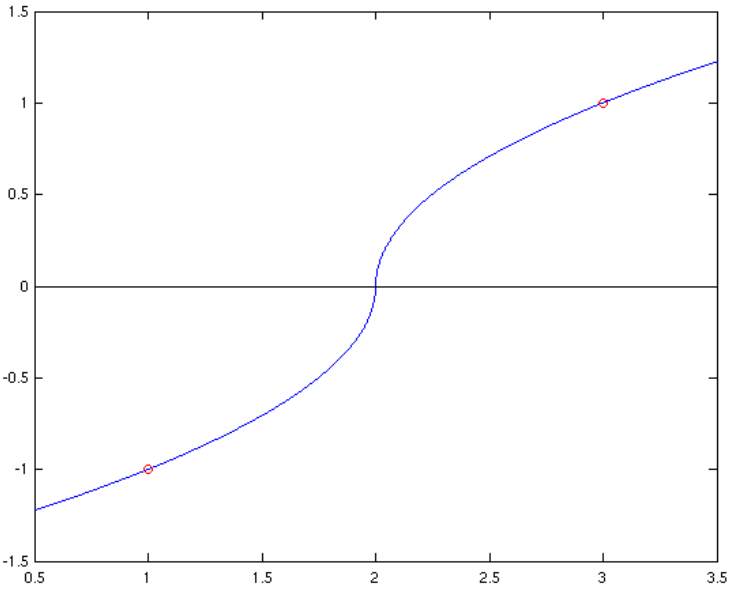
  - Startintervall [4.8, 5.5]
  - f(4.8) = -7.861819
  - f(5.5) = 2.292519



  Beispiel 2 ("bösartig")

  $$f(x) = \text{sign}(x-2)\sqrt{|x-2|}$$

  - Startintervall [1 3]
  - f(1) = -1
  - f(3) = 1



- Bisektionsverfahren:

  stetiges f wechsle Vorzeichen auf [a b]

einfaches Verfahren mit ständiger Intervallhalbierung

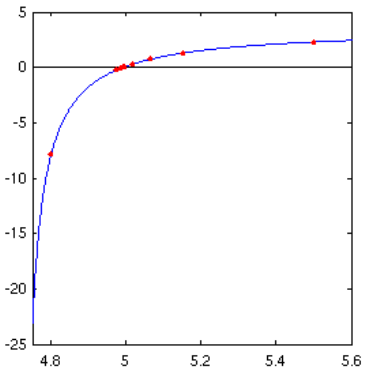
```
wiederhole bis Intervallbreite < delta
   bestimme Mittelpunkt xm = (a + b)/2
   berechne Vorzeichen von f(xm)
   wenn f(b) f(xm) < 0
      neues Intervall [xm b]
   sonst
      neues Intervall [a xm]
```

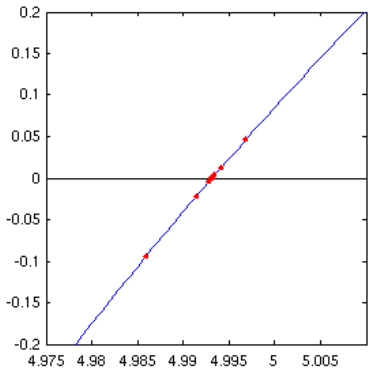
funktioniert immer (bei exakter Arithmetik)

beim Standardbeispiel (nachrechnen mit kap04.m)

a	b	xm	f(xm)
4.800000000000000	5.500000000000000	5.150000000000000	+1.28e+00
4.800000000000000	5.150000000000000	4.975000000000000	-2.47e-01
4.975000000000000	5.150000000000000	5.062500000000000	+7.08e-01
4.975000000000000	5.062500000000000	5.018750000000000	+2.99e-01
4.975000000000000	5.018750000000000	4.996874999999999	+4.69e-02
4.975000000000000	4.996874999999999	4.985937499999999	-9.40e-02
4.985937499999999	4.996874999999999	4.991406249999999	-2.21e-02
4.991406249999999	4.996874999999999	4.994140624999999	+1.27e-02
4.991406249999999	4.994140624999999	4.992773437499999	-4.63e-03
4.992773437499999	4.994140624999999	4.993457031249999	+4.07e-03
4.992773437499999	4.993457031249999	4.993115234374999	-2.75e-04
4.993115234374999	4.993457031249999	4.993286132812499	+1.90e-03
4.993115234374999	4.993286132812499	4.993200683593749	+8.12e-04
4.993115234374999	4.993200683593749	4.993157958984374	+2.69e-04
4.993115234374999	4.993157958984374	4.993136596679687	-2.80e-06
4.993136596679687	4.993157958984374	4.993147277832030	+1.33e-04
4.993136596679687	4.993147277832030	4.993141937255858	+6.51e-05
4.993136596679687	4.993141937255858	4.993139266967773	+3.12e-05
4.993136596679687	4.993139266967773	4.993137931823730	+1.42e-05
4.993136596679687	4.993137931823730	4.993137264251708	+5.69e-06
4.993136596679687	4.993137264251708	4.993136930465697	+1.45e-06
4.993136596679687	4.993136930465697	4.993136763572692	-6.76e-07
4.993136763572692	4.993136930465697	4.993136847019194	+3.86e-07
4.993136763572692	4.993136847019194	4.993136805295944	-1.45e-07
4.993136805295944	4.993136847019194	4.993136826157569	+1.20e-07
4.993136805295944	4.993136826157569	4.993136815726756	-1.23e-08
4.993136815726756	4.993136826157569	4.993136820942162	+5.40e-08
4.993136815726756	4.993136820942162	4.993136818334460	+2.08e-08
4.993136815726756	4.993136818334460	4.993136817030608	+4.26e-09
4.993136815726756	4.993136817030608	4.993136816378682	-4.03e-09
4.993136816378682	4.993136817030608	4.993136816704645	+1.13e-10
4.993136816378682	4.993136816704645	4.993136816541663	-1.96e-09
4.993136816541663	4.993136816704645	4.993136816623155	-9.24e-10
4.993136816623155	4.993136816704645	4.993136816663900	-4.06e-10
4.993136816663900	4.993136816704645	4.993136816684272	-1.46e-10
4.993136816684272	4.993136816704645	4.993136816694459	-1.69e-11
4.993136816694459	4.993136816704645	4.993136816699552	+4.79e-11
4.993136816694459	4.993136816699552	4.993136816697005	+1.55e-11
4.993136816694459	4.993136816697005	4.993136816695731	-6.79e-13
4.993136816695731	4.993136816697005	4.993136816696368	+7.42e-12
4.993136816695731	4.993136816696368	4.993136816696049	+3.37e-12
4.993136816695731	4.993136816696049	4.993136816695890	+1.34e-12
4.993136816695731	4.993136816695890	4.993136816695811	+3.38e-13
4.993136816695731	4.993136816695811	4.993136816695771	-1.70e-13

```
4.993136816695771   4.993136816695811   4.993136816695792  +8.93e-14
4.993136816695771   4.993136816695792   4.993136816695781  -4.66e-14
4.993136816695781   4.993136816695792   4.993136816695786  +2.13e-14
4.993136816695781   4.993136816695786   4.993136816695784  -1.24e-14
4.993136816695784   4.993136816695786   4.993136816695785  -8.88e-16
4.993136816695785   4.993136816695786   4.993136816695785  +1.02e-14
```



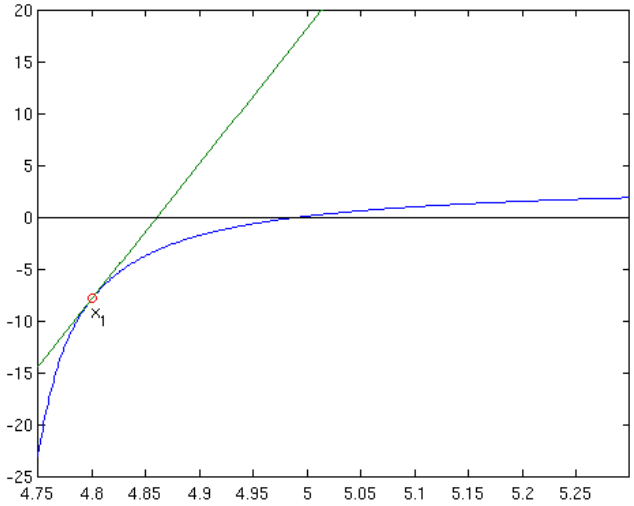


Problem mit Rechnerarithmetik

- Wie klein darf Abbruchbreite δ (= Genauigkeit des Ergebnisses) sein?
- sinnvoller Minimalwert: δ = ε*abs(a)
- bei "problematischen" Funktionen viel früher
- vgl. Funktion B im Polynom-Plot

- Newton-Verfahren:

  Idee: nähere Funktion durch Tangente



  Tangente durch x_n

$$y = f'(x_n)(x - x_n) + f(x_n)$$

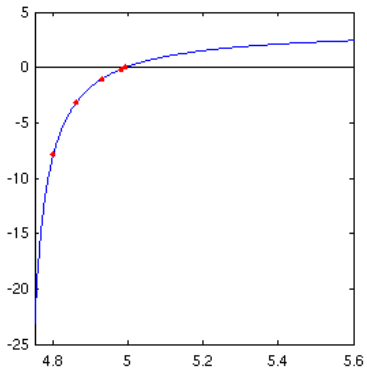
  x_{n+1} = Nullstelle der Tangente →

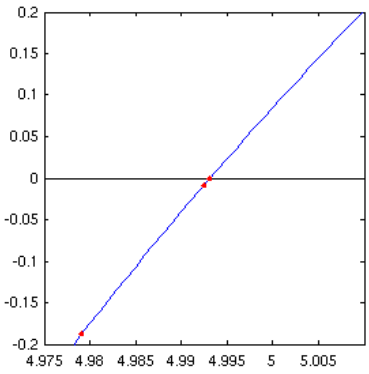
$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

  Standardbeispiel 1

```
4.800000000000000
4.860313493747295
4.930340369122104
4.979104035027125
```

```
4.992436030128772
4.993135068876532
4.993136816684912
4.993136816695785
4.993136816695785
```

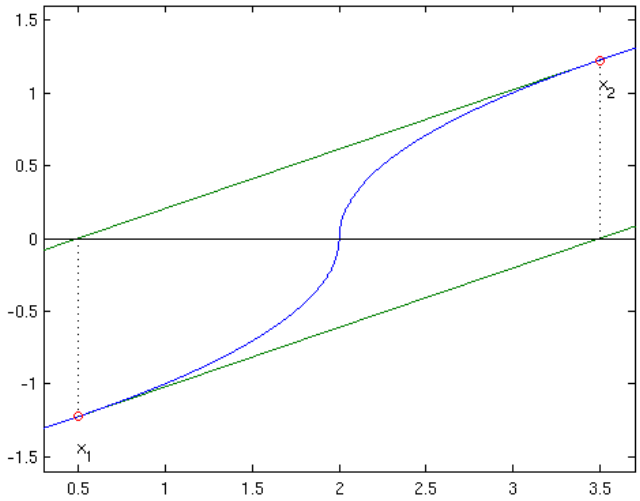




konvergiert hier sehr viel schneller als Bisektionsverfahren

Standardbeispiel 2

```
3.5
0.5
3.5
0.5
...
```



Probleme

- konvergiert nur in der Nähe der Lösung ("lokal")
- Ableitung von f wird benötigt

- Konvergenz-Geschwindigkeit:

  Sei x die exakte Lösung, $\varepsilon_i := |x_i - x|$ der Fehler im i-ten Schritt. Gibt es ein $p \geq 1$ und ein c mit $0 \leq c < 1$, so dass gilt

  $$\limsup_{i \to \infty} \frac{\varepsilon_{i+1}}{\varepsilon_i^p} \leq c$$

  dann heißt p die **Konvergenzordnung** der Folge x_i.

  etwas laxer ausgedrückt: Der Fehler sinkt etwa wie

  $$\varepsilon_{i+1} \leq c\,\varepsilon_i^p$$

Beispiel Bisektionsverfahren

- Fehler wird in jedem Schritt halbiert

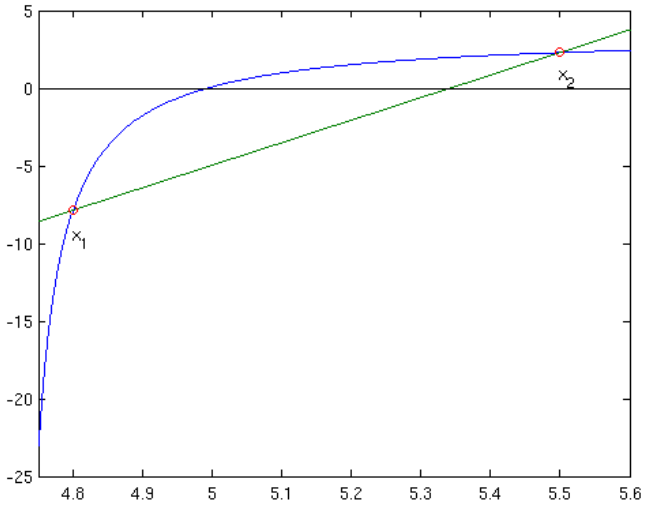
$$\epsilon_{i+1} \leq \frac{1}{2}\epsilon_i$$

- also p = 1, c = 1/2

bei Newton ist p = 2 (falls f'(x) ≠ 0) (Beweis: [3])

- Sekanten-Verfahren:

Idee: nähere die Tangente bei Newton durch die Sekante durch die letzten zwei Punkte

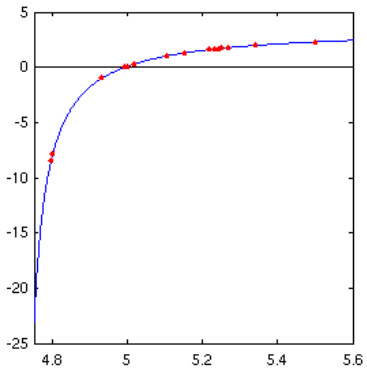


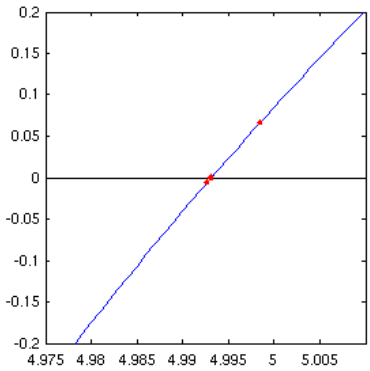
konkret

$$\begin{aligned} m &= \frac{f(x_{n-1}) - f(x_n)}{x_{n-1} - x_n} \\ x_{n+1} &= x_n - \frac{f(x_n)}{m} \end{aligned}$$

Standardbeispiel 1

```
5.500000000000000
4.800000000000000
5.341962754641359
5.233078480512697
4.692715063982089
5.249938048452647
5.267980603914145
4.740153341717683
5.240757795434711
5.216227584826059
4.795417173225895
5.150185032751038
5.103681829226098
4.931222941473443
5.017500486888689
4.998505040389419
4.992671292678041
4.993145710865901
4.993136831431582
4.993136816695318
4.993136816695785
4.993136816695785
```





Konvergenzordnung p = 1.618 (vgl. [2])

Probleme

- konvergiert nur in der Nähe der Lösung ("lokal")
- Sekantensteigung wird ungenau (Subtraktion ähnlicher Zahlen)

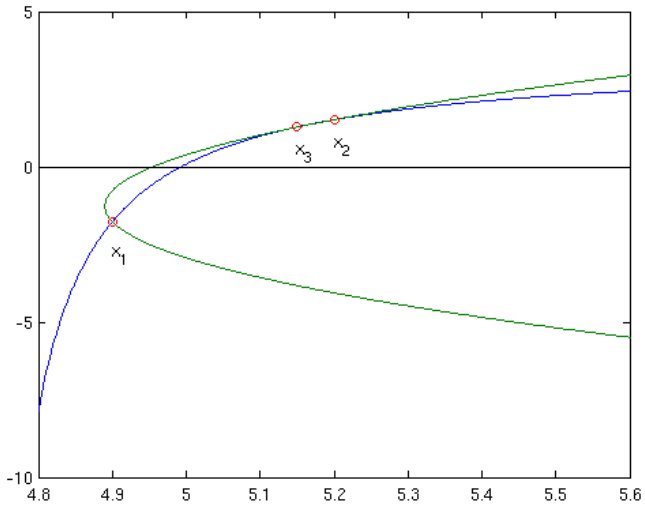
- Inverse quadratische Interpolation (IQI):

  Idee: statt einer Geraden durch die letzten zwei Punkte bestimme eine Parabel durch die letzten drei Punkte

  Problem: Parabel hat evtl. keine Nullstelle

  Lösung: wähle "inverse Parabel"

  $$x = ay^2 + by + c$$



  konkret

  - ggb. letzte drei Werte x_1, x_2, x_3 und Funktionswerte $y_i = f(x_i)$
  - Einsetzen der Punkte in Parabelgleichung liefert lineares System für (a b c)

    $$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} y_1^2 & y_1 & 1 \\ y_2^2 & y_2 & 1 \\ y_3^2 & y_3 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix}$$

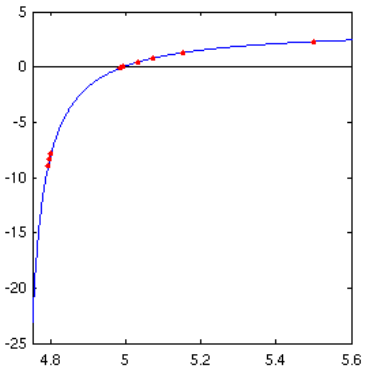
  - nächster Punkt = Schnittpunkt von Parabel und x-Achse = c

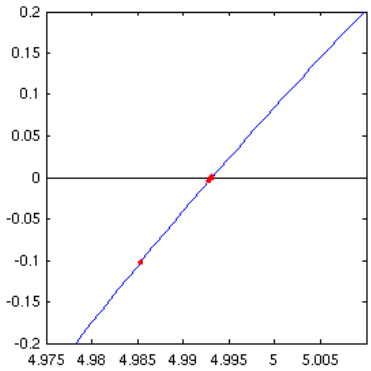
    x_4 = c

  Standardbeispiel 1

```
4.800000000000000
5.500000000000000
5.150000000000000
```

```
4.796299397999127
4.792545722045462
5.071630116446963
5.032407352330408
4.985297936065732
4.992830056897817
4.993138014176794
4.993136816732284
4.993136816695785
4.993136816695785
```





Konvergenzordnung p = 1.839 (vgl. [2])

Probleme

- konvergiert nur in der Nähe der Lösung ("lokal")
- Probleme, wenn zwei der drei Funktionswerte $f(x_i)$ fast gleich sind

- Aufgaben:

  Aufgabe 5

  Aufgabe 6

Das Dekker-Brent-Verfahren

- Dekker-Brent-Verfahren

  Problem:
  - Sekantenverfahren und IQI schnell, aber (zumindest am Anfang) unzuverlässig
  - Bisektion langsam, aber sicher

  Lösung: kombiniere die Verfahren geschickt mit Hilfe dreier Werte a, b, c, wobei
  - a und b liegen auf verschiedenen Seiten der Nullstelle, sie bilden immer das kleinste bisher gefundene Intervall
  - b ist immer der beste bisher gefundene Wert, d.h. der mit dem betragsmäßig kleinsten Funktionswert
  - c ist der vorherige Wert von b

  konkreter

```
Ausgangssituation: Startintervall [a b] mit
   f(a) f(b) < 0
   |f(b)| ≤ |f(a)|  (ggf. a, b vertauschen)
finde neuen Wert c aus [a b] mit Sekantenschritt
sortiere a, b, c um
wiederhole bis Intervallbreite < delta
   wenn (c ≠ a) und (c ≠ b)
      probiere IQI-Schritt
   sonst
      probiere Sekantenschritt
   end
   wenn neuer Wert im alten Intervall [a b]
     nimm ihn als neues c
   sonst
     bestimme neues c mit Bisektion
   end
   sortiere a, b, c um
```

  einige komplexere Details des Verfahrens fehlen ([12])
  - Kontrolle möglicher numerischer Probleme
  - Methode zur Beschleunigung bei "bösartigen" Funktionen
  - Programmiertricks zur allgemeinen Beschleunigung

  genaueres Abbruchverfahren
  - bestimme Ergebnis $x_0 = (a + b)/2$
  - bestimme absoluten Fehler $\varepsilon_{abs} = |b - a|/2$ und relativen Fehler $\varepsilon_{rel} = \varepsilon_{abs}/|x_0|$
  - Abbruch, wenn $\varepsilon_{rel} < \delta$

  Verfahren ist so zuverlässig wie Bisektion, aber in der Regel deutlich schneller

- Matlab-Routine `fzero`:

  basiert auf Dekker-Brent-Verfahren

  falls nur ein Startwert x_0 (kein Intervall):
  - suche Intervall mit Vorzeichenwechsel durch allmähliches Vergrößern um x_0 herum

  das kann schiefgehen, vor allem in der Nähe von Polstellen (mit Standardbeispiel 1)

```
fzero(f1, 4.8)
ans = 4.712388980384693
```

  fzero wird gesprächig bei Angabe einer entsprechenden Option

```
options = optimset("Display", "iter");
fzero(@cos, [1 3], options)
```

- Aufgaben:

  Aufgabe 7

Mehrdimensionale Nullstellensuche

- Problemstellung:

  gegeben sind n Funktionen F_i mit n Unbekannten x_i

  gesucht sind Nullstellen x_i aller Gleichungen

  $$\begin{aligned} F_1(x_1,\ldots,x_n) &= 0 \\ &\vdots \\ F_n(x_1,\ldots,x_n) &= 0 \end{aligned}$$

  - bzw. vektoriell geschrieben

    $$\vec{F}(\vec{x}) = 0$$

  ohne weitere Kenntnisse von F oder guten Anfangswerten i. a. nahezu unlösbar (vgl. [6, Kap. 9.6)]) !

- Graphische Veranschaulichung im 2d-Fall:

  $F_1(x, y) = z \triangleq$ Fläche in 3d

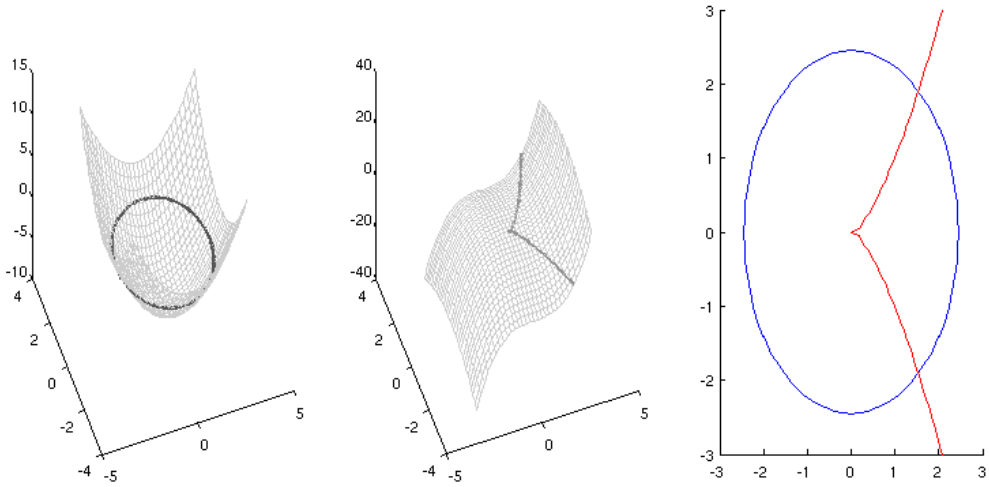
  $F_1(x, y) = 0 \triangleq$ Schnittkurve dieser Fläche mit der xy-Ebene

  Lösung beider Gleichungen ≙ Schnittpunkte der beiden Schnittkurven

  Beispiel

  $$\begin{aligned} F_1(x,y) &= x^2+y^2-6=0 \\ F_2(x,y) &= x^3-y^2=0 \end{aligned}$$

  - im Bild



  Erstellung dieser Graphik in Matlab mit Hilfe der Funktion plotZeros.m

```
% Beispielfunktion
f1 = @(x,y) x.^2 + y.^2 - 6;
f2 = @(x,y) x.^3 - y.^2;
% Bereich der Koordinaten
xvals = -3:.2:3;
yvals = -3:.2:3;
plotZeros(f1, f2, xvals, yvals)
```

- Lösen mit dem Newton-Verfahren:

  mehrdimensionale Taylorentwicklung in 1. Ordnung

  $$\vec{F}(\vec{x}_0+\vec{h}) = \vec{F}(\vec{x}_0) + \left(\frac{\partial \vec{F}}{\partial \vec{x}}\right)_{\vec{x}_0} \cdot \vec{h} + O(\vec{h}^2)$$

mit der Funktionalmatrix (Jacobimatrix)

$$\left(\frac{\partial \vec{F}}{\partial \vec{x}}\right)_{\vec{x}_k} = \begin{pmatrix} \frac{\partial F_1}{\partial x_1}(\vec{x}_k) & \cdots & \frac{\partial F_1}{\partial x_n}(\vec{x}_k) \\ \cdots & & \cdots \\ \frac{\partial F_n}{\partial x_1}(\vec{x}_k) & \cdots & \frac{\partial F_n}{\partial x_n}(\vec{x}_k) \end{pmatrix}$$

nähert Funktion durch "Tangentialebene"

Funktionen beim Näherungswert $\mathbf{x}_k$ durch lineare Näherung ersetzen und auflösen

$$\vec{F}(\vec{x}) \approx \vec{F}(\vec{x}_k) + \left(\frac{\partial \vec{F}}{\partial \vec{x}}\right)_{\vec{x}_k} (\vec{x} - \vec{x}_k) \stackrel{!}{=} 0$$

Nullstelle **x** des linearen Gleichungssystems als nächsten Wert $\mathbf{x}_{k+1}$ wählen

$$\left(\frac{\partial \vec{F}}{\partial \vec{x}}\right)_{\vec{x}_k} (\vec{x} - \vec{x}_k) = -\vec{F}(\vec{x}_k)$$

iterieren, bis gewünschte Genauigkeit erreicht

- Beispiel:

  löse System des obigen Beispiels

  (schlechter) Schätzwert

  $x_0 = 1$, $y_0 = 1$

  Berechnen der Jacobimatrix

  $$\frac{\partial F_1}{\partial x} = 2x \qquad \frac{\partial F_1}{\partial y} = 2y$$
  $$\frac{\partial F_2}{\partial x} = 3x^2 \qquad \frac{\partial F_2}{\partial y} = -2y$$

  - also

  $$\left(\frac{\partial \vec{F}}{\partial \vec{x}}\right) = \begin{pmatrix} 2x & 2y \\ 3x^2 & -2y \end{pmatrix}$$

  Bestimmung von **z** := **x** - $\mathbf{x}_k$ aus dem linearen Gleichungssystem

  $$\left(\frac{\partial \vec{F}}{\partial \vec{x}}\right)_{\vec{x}_0} \cdot \vec{z} = -\vec{F}(\vec{x}_0)$$

  - Wert von $\mathbf{x}_0$ einsetzen →

  $$\begin{pmatrix} 2 & 2 \\ 3 & -2 \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} 4 \\ 0 \end{pmatrix}$$

  - Ergebnis

  $$\vec{z} = \begin{pmatrix} 0.8 \\ 1.2 \end{pmatrix}$$

  daraus nächster Schätzwert

  $$\vec{x}_1 = \vec{x}_0 + \vec{z} = \begin{pmatrix} 1.8 \\ 2.2 \end{pmatrix}$$

  Wiederholen liefert

$$\vec{x}_2 = \begin{pmatrix} 1.5694 \\ 1.9160 \end{pmatrix}$$
$$\vec{x}_3 = \begin{pmatrix} 1.5382 \\ 1.9066 \end{pmatrix}$$

  - korrekt auf 3 Nachkommastellen

- Implementierung in Matlab:

  keine Standardfunktion für Newton-Verfahren vorhanden

  einfache Version (ohne Tests) in solveNewton.m

  Aufruf

  ```
  x = solveNewton(F, DF, x0, tol)
  ```

  Parameter

Name	Bedeutung	Typ
`F`	Funktion, die die Gleichung definiert	Funktion `y = F(x)` mit Vektoren `x`, `y`
`DF`	Jacobimatrix von `F`	Funktion `Dy = DF(x)` mit Vektor `x` und Matrix `Dy`
`x0`	Schätzwert	Vektor
`tol`	gewünschte Genauigkeit	Zahl

  Anwendung im Beispiel mit testNewton.m

- Aufgaben:

  Aufgabe 8

Interpolation und Approximation

- Interpolation
- Ausgleichsrechnung
- Fourieranalyse

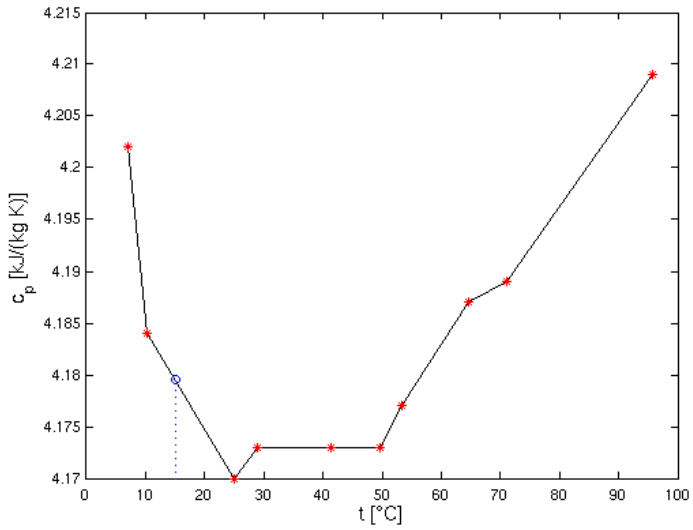
Interpolation

- Spezifische Wärmekapazität c_p von Wasser:

  einige Messwerte wurden bestimmt

  daraus soll Wert für t_1 = 15 °C ermittelt werden

  am einfachsten: lineare Interpolation



  Ergebnis: $c_p(t_1)$ = 4.1795 kJ/(kg K)

  kann man einen "besseren" Wert bekommen?

- Problem der Interpolation:

  gegeben N Punkte

  (x_i, y_i), i = 1, .. N, mit $x_i \neq x_j$ für $i \neq j$

  gesucht Funktion f einer vorgegebenen Klasse (z. B. Polynom) mit

  $f(x_i) = y_i$, i = 1, .. N

- Polynom-Interpolation:

  Es gibt genau ein Interpolationspolynom P vom Grad N-1

$$P(x) = \sum_{k=0}^{N-1} a_k x^k$$

  Einsetzen der Punkte liefert N Gleichungen für die N Koeffizienten a_k

$$P(x_i) = \sum_{k=0}^{N-1} a_k x_i^k = y_i \qquad i = 1 \dots N$$

  in Matrixform

$$\begin{pmatrix} 1 & x_1 & x_1^2 & \dots & x_1^{N-1} \\ 1 & x_2 & x_2^2 & \dots & x_2^{N-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_N & x_N^2 & \dots & x_N^{N-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{N-1} \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix}$$

  Systemmatrix

  - **Vandermonde-Matrix**

- nicht singulär (bei $x_i \neq x_j$ für $i \neq j$)
- schlechte Kondition bei kleinen Abständen zwischen den x_i

- Lagrangeform der Lösung:

  Lösung direkt hinschreibbar, etwa quadratisches Polynom für N = 3

  $$P(x) = \frac{(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)}y_1 + \frac{(x-x_1)(x-x_3)}{(x_2-x_1)(x_2-x_3)}y_2 + \frac{(x-x_1)(x-x_2)}{(x_3-x_1)(x_3-x_2)}y_3$$

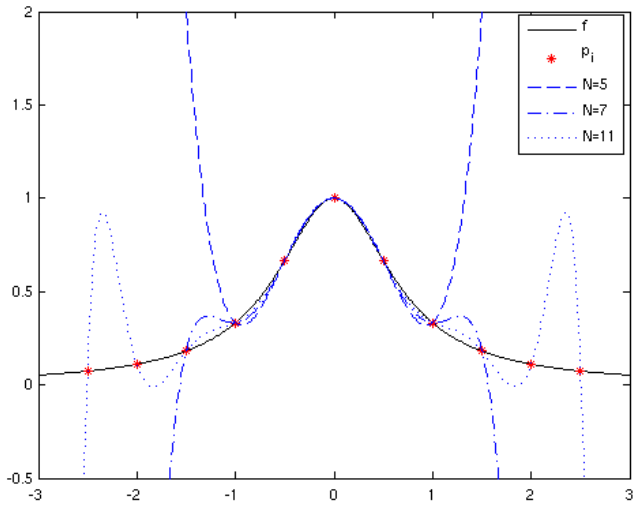
  Überprüfen durch Einsetzen von x_1, x_2, x_3

  allgemein

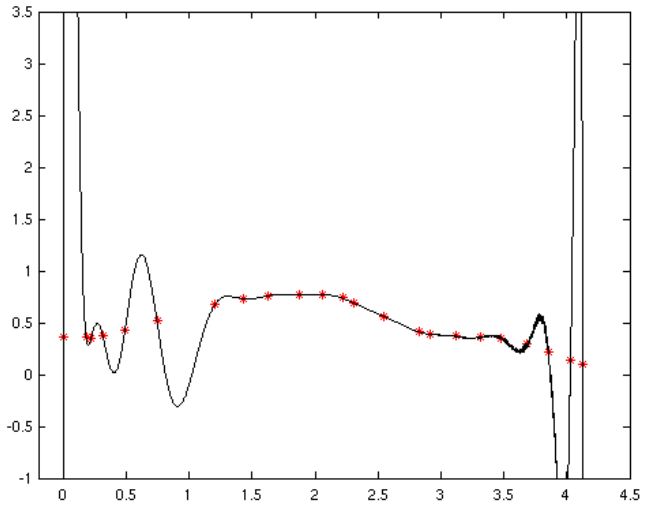
  $$P(x) = \sum_{k=1}^{N} \left(\prod_{j \neq k} \frac{x-x_j}{x_k-x_j} \right) y_k$$

- Beispiele:

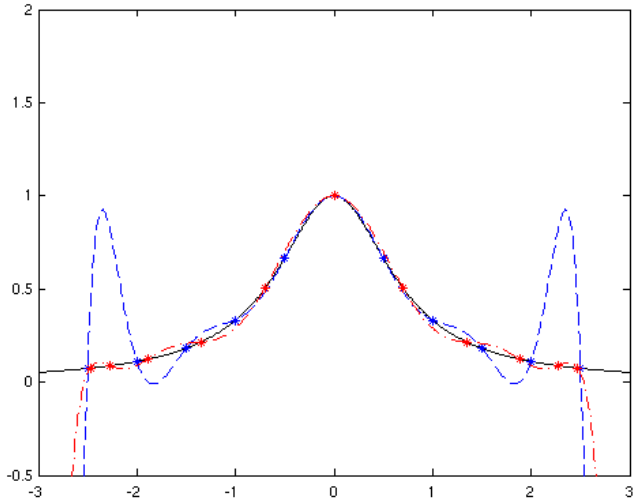
  "sinnvoll" nur bei kleinem N



  starke Überschwinger bei großem N



  bei bekannter Funktion f optimierbar durch geschickte Wahl der Stützstellen x_i (**Tschebycheff-Knoten**)

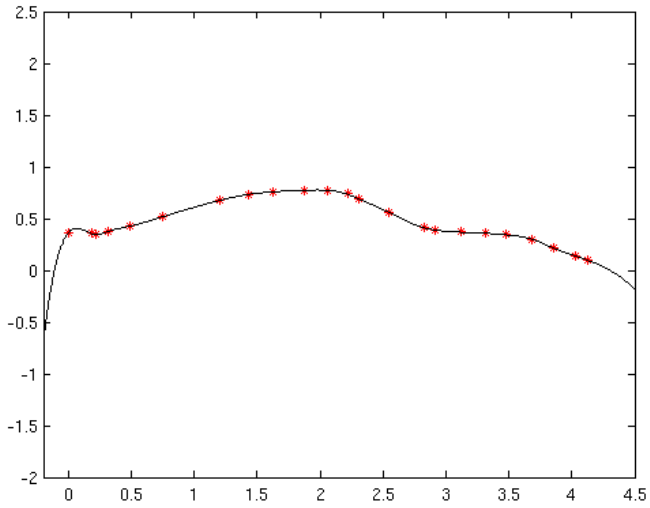


- Kubische Splines:

  Alternative: stückweise Polynome $P_k(x)$ niedriger Ordnung

  kubische Splines definiert durch
  - Polynom 3. Ordnung zwischen je zwei aufeinanderfolgenden Punkten
  - 1. und 2. Ableitungen sind stetig

  Beispiel



  Splines gegeben durch je 4 Koeffizienten a_k, b_k, c_k, d_k

$$P_k(x) = a_k x^3 + b_k x^2 + c_k x + d_k, \qquad k = 1 \ldots N-1$$

  Bedingungen

$$\begin{aligned}
P_k(x_k) &= y_k && k = 1 \ldots N-1 && (1)\\
P_k(x_{k+1}) &= y_{k+1} && k = 1 \ldots N-1 && (2)\\
P'_k(x_{k+1}) &= P'_{k+1}(x_{k+1}) && k = 1 \ldots N-2 && (3)\\
P''_k(x_{k+1}) &= P''_{k+1}(x_{k+1}) && k = 1 \ldots N-2 && (4)
\end{aligned}$$

  Anzahl der Koeffizienten: 4N - 4

  Anzahl der Bedingungen: (N-1) + (N-1) + (N-2) + (N-2) = 4N-6

  bleibt je eine zusätzliche Bedingung an den Endpunkten x_1, x_N

- Wahl der Randbedingungen:

  natürliche Splines

  - Extrapolationen (Kurven außerhalb [x_1 x_N]) verlaufen linear

$$\begin{aligned} P_1''(x_1) &= 0 \\ P_{N-1}''(x_N) &= 0 \end{aligned}$$

  "not-a-knot"-Splines

  - bei x_2 und x_{N-1} dreimal stetig differenzierbar

$$\begin{aligned} P_1'''(x_2) &= P_2'''(x_2) \\ P_{N-2}'''(x_{N-1}) &= P_{N-1}'''(x_{N-1}) \end{aligned}$$

  - ein einziges kubisches Polynom durch (x_1, x_2, x_3) bzw. (x_{N-2}, x_{N-1}, x_N)

  periodische Splines

$$\begin{aligned} P_1'(x_1) &= P_{N-1}'(x_N) \\ P_1''(x_1) &= P_{N-1}''(x_N) \end{aligned}$$

  - wichtig bei geschlossenen Kurven

  Ableitungen M_1, M_N an den Endpunkten bekannt

$$\begin{aligned} P_1'(x_1) &= M_1 \\ P_{N-1}'(x_N) &= M_N \end{aligned}$$

- Bestimmung der Koeffizienten:
  - die (noch unbekannten!) Steigungen an den Punkten seien M_k, also

$$P'(x_k) = M_k, \qquad k = 1 \ldots N$$

  - mit den Abkürzungen

$$\begin{aligned} h_k &:= x_{k+1} - x_k \quad k = 1 \ldots N-1 \\ s_k &:= x - x_k \quad\;\; k = 1 \ldots N-1 \end{aligned}$$

  hat das Polynom

$$\begin{aligned} P_k(x) &= \frac{1}{h_k^3}\left((3h_k - 2s_k)s_k^2 y_{k+1} + (h_k + 2s_k)(h_k - s_k)^2 y_k \right. \\ &\quad \left. + s_k^2(s_k - h_k)h_k M_{k+1} + s_k(s_k - h_k)^2 h_k M_k\right) \end{aligned}$$

  die Eigenschaften

$$\begin{aligned} P_k(x_k) &= y_k \qquad\quad k = 1 \ldots N-1 \\ P_k(x_{k+1}) &= y_{k+1} \qquad k = 1 \ldots N-1 \\ P_k'(x_k) &= M_k \qquad\quad k = 1 \ldots N-1 \\ P_k'(x_{k+1}) &= M_{k+1} \qquad k = 1 \ldots N-1 \end{aligned}$$

  - aus der Stetigkeit der 2. Ableitung (4) erhält man N-2 lineare Gleichungen für die M_k

$$\begin{aligned} &h_{k+1}M_k + 2(h_k + h_{k+1})M_{k+1} + h_k M_{k+2} = \\ &\quad 3\left(-\frac{h_{k+1}}{h_k}y_k + \left(\frac{h_{k+1}}{h_k} - \frac{h_k}{h_{k+1}}\right)y_{k+1} + \frac{h_k}{h_{k+1}}y_{k+2}\right) \qquad k = 1 \ldots N-2 \end{aligned}$$

  - 2 ergänzende Gleichungen für natürliche Splines

$$\begin{aligned} 2M_1 + M_2 &= \frac{3}{h_1}(y_2 - y_1) \\ 2M_N + M_{N-1} &= \frac{3}{h_{N-1}}(y_N - y_{N-1}) \end{aligned}$$

- 2 ergänzende Gleichungen für "not-a-knot"-Splines

$$
\begin{aligned}
h_2^2 M_1 + (h_2^2 - h_1^2) M_2 - h_1^2 M_3 &= 2\frac{h_1^2}{h_2}(y_2 - y_3) - 2\frac{h_2^2}{h_1}(y_1 - y_2) \\
h_{N-1}^2 M_{N-2} + (h_{N-1}^2 - h_{N-2}^2) M_{N-1} - h_{N-2}^2 M_N &= 2\frac{h_{N-2}^2}{h_{N-1}}(y_{N-1} - y_N) - 2\frac{h_{N-1}^2}{h_{N-2}}(y_{N-2} - y_{N-1})
\end{aligned}
$$

  - Details der Rechnung im Anhang

- Matlab-Funktionen:

  Interpolation bei Punkten x, y, an den Stellen xi
  - `yi = interp1(x, y, xi, Methode)`
  - Methoden (u.a.) `"linear"`, `"spline"`
  - `"spline"` verwendet not-a-knot-Randbedingungen

  Interpolationspolynom bestimmen
  - `poly = polyfit(x, y, length(x)-1);`

  Zwischenwerte des Polynoms berechnen
  - `yi = polyval(poly, xi);`

- B-Splines:

  interpolieren beliebige Raumkurve

  zusätzliche Kontrollpunkte legen Tangentenrichtungen fest

  Basis von Freiformkurven in 2D-CAD

  Verallgemeinerung auf Flächen
  - NURBS (Non Uniform Rational B-Splines)
  - Basis von Freiformflächen in 3D-CAD

- Aufgaben:

  Aufgabe 9

  Aufgabe 10

Ausgleichsrechnung

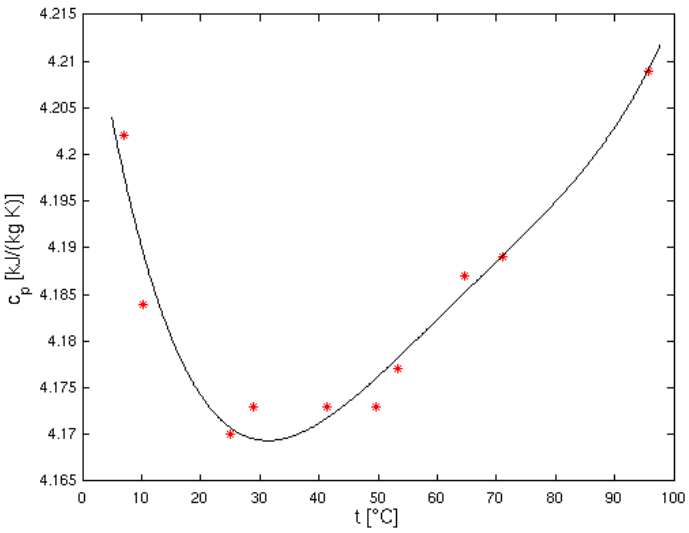
- Spezifische Wärmekapazität cp von Wasser:

  Werte wie vorher

  Idee:

  - Messwerte haben Fehler
  - Daten sollten durch Polynom 4. Ordnung gut beschrieben werden

  gesucht: "bestes" Polynom zu den gegebenen Messwerten



- Problem der Approximation:

  gegeben N Punkte (xi, yi), i = 1, .. N, mit xi ≠ xj für i ≠ j

  gesucht Funktion f aus einer vorgegebenen Klasse (z. B. Polynom fester Ordnung), die "möglichst genau" durch die Punkte geht

  Fehler von f bei Messung i

  $$r(i) = f(x_i) - y_i$$

  Gesamtfehler ("Methode der kleinsten Quadrate", "least square fit")

  $$r_2 = \sqrt{\sum_{i=1}^{N} r(i)^2} = \sqrt{\sum_{i=1}^{N} (f(x_i) - y_i)^2}$$

  auch andere Gesamtfehler werden verwendet

  $$r_1 := \sum_{i=1}^{N} |r(i)|$$
  $$r_\infty := \max_i |r(i)|$$

  Vorteil von r_2

  - führt auf numerisch gut zu lösende Gleichungen
  - liefert für normalverteilte Messwerte den "wahrscheinlichsten" Fit ("maximum likelihood")

- Lineare Ausgleichsrechnung:

  lineares Gleichungssystem mit m Gleichungen für n Unbekannte (m > n)

  A x = b

  A mxn-Matrix, x n-Vektor, b m-Vektor

hat in der Regel keine Lösung

Beispiel

$$\begin{pmatrix} 1 & 2 & -1 \\ 2 & 0 & 2 \\ -2 & 2 & -3 \\ 4 & 1 & 4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 2 \\ 3 \\ -1 \\ 2 \end{pmatrix}$$

gesucht ist x mit minimalem quadratischen Fehler

kann berechnet werden als Lösung der Normalengleichung

$(A^T A)\, x = A^T b$

Beweis im Anhang

Lösung des nxn-Systems mit Gauß-Verfahren ist numerisch problematisch (schlecht konditioniert)

besser über QR-Zerlegung

- Matrix A wird zerlegt in

  $A = Q R$

- mit

  R = obere Dreiecksmatrix (unter der Diagonalen nur 0)

  Q orthogonal (also $Q^T Q = 1$)

- QR-Zerlegung am Beispiel:

  Prinzip

  - Durch Spiegelung an einer geeigneten Ebene (gegeben durch ihren Normalenvektor v) wird ein Spaltenvektor von A in Richtung eines Koordinatenvektors e_i gebracht

  Basisvektoren: e_1, e_2, e_3

  Spaltenvektoren der (transformierten) Systemmatrix: s_1, s_2, s_3,

  1. Schritt

     a. berechne Hilfsvektor

     $$v_1 = s_1 - |s_1| e_1$$

     konkret

     $$v_1 = \begin{pmatrix} 1 \\ 2 \\ -2 \\ 4 \end{pmatrix} - 5 \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} -4 \\ 2 \\ -2 \\ 4 \end{pmatrix}$$

     b. berechne Householdermatrix

     $$Q_1 = 1 - \frac{2}{v_1^T v_1}(v_1 v_1^T)$$

     konkret

     $$Q_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} - \frac{1}{20} \begin{pmatrix} 16 & -8 & 8 & -16 \\ -8 & 4 & -4 & 8 \\ 8 & -4 & 4 & -8 \\ -16 & 8 & -8 & 16 \end{pmatrix} = \frac{1}{10} \begin{pmatrix} 2 & 4 & -4 & 8 \\ 4 & 8 & 2 & -4 \\ -4 & 2 & 8 & 4 \\ 8 & -4 & 4 & 2 \end{pmatrix}$$

     c. transformiere A

$$A_1 = Q_1 A = \begin{pmatrix} 5 & 0.4 & 5 \\ 0 & 0.8 & -1 \\ 0 & 1.2 & 0 \\ 0 & 2.6 & -2 \end{pmatrix}$$

2. Schritt:

a. berechne Hilfsvektor, ersetze dafür 1. Wert von s_2 durch Nullen →

$$v_2 = \begin{pmatrix} 0 \\ 0.8 \\ 1.2 \\ 2.6 \end{pmatrix} - 2.9732 \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ -2.1732 \\ 1.2 \\ 2.6 \end{pmatrix}$$

b. berechne Householdermatrix

$$Q_2 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} - 0.1548 \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 4.7229 & -2.6079 & -5.6504 \\ 0 & -2.6079 & 1.4400 & 3.1200 \\ 0 & -5.6504 & 3.1200 & 6.7600 \end{pmatrix}$$
$$= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0.2691 & 0.4036 & 0.8745 \\ 0 & 0.4036 & 0.7771 & -0.4829 \\ 0 & 0.8745 & -0.4829 & -0.0462 \end{pmatrix}$$

c. transformiere A

$$A_2 = Q_2 A_1 = \begin{pmatrix} 5 & 0.4 & 5 \\ 0 & 2.9732 & -2.0180 \\ 0 & 0 & 0.5621 \\ 0 & 0 & -0.7821 \end{pmatrix}$$

3. Schritt:

a. berechne Hilfsvektor, ersetze dafür 1. und 2. Wert von s_3 durch Nullen →

$$v_3 = \begin{pmatrix} 0 \\ 0 \\ 0.5621 \\ -0.7821 \end{pmatrix} - 0.9631 \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ -0.4010 \\ -0.7821 \end{pmatrix}$$

b. berechne Householdermatrix

$$Q_3 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} - 2.5893 \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0.1608 & 0.3136 \\ 0 & 0 & 0.3136 & 0.6116 \end{pmatrix}$$
$$= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0.5837 & -0.8120 \\ 0 & 0 & -0.8120 & -0.5837 \end{pmatrix}$$

c. transformiere A

$$A_3 = Q_3 A_2 = \begin{pmatrix} 5 & 0.4 & 5 \\ 0 & 2.9732 & -2.0180 \\ 0 & 0 & 0.9631 \\ 0 & 0 & 0 \end{pmatrix}$$

damit ist eine obere Dreiecksmatrix erreicht, also

$R = A_3$

Da Q_1, Q_2 und Q_3 orthogonal sind (nachrechnen!), ist es auch

$$Q = Q_1^T Q_2^T Q_3^T = \begin{pmatrix} 0.2000 & 0.6458 & -0.7235 & 0.1397 \\ 0.4000 & -0.0538 & -0.1128 & -0.9080 \\ -0.4000 & 0.7265 & 0.4839 & -0.2794 \\ 0.8000 & 0.2287 & 0.4792 & 0.2794 \end{pmatrix}$$

und es gilt

A = Q R

- Lösung der Normalengleichung:

  Mit der QR-Zerlegung von A kann man schreiben

  $$\begin{aligned} Ax &= QRx \approx b \\ \Rightarrow \quad Rx &\approx Q^T b \end{aligned}$$

  Da nur die oberen n Zeilen von R von 0 verschieden sind, teilt man das System in die oberen n und die unteren m-n Gleichungen

  $$Rx = \begin{pmatrix} R_1 x \\ 0 \cdot x \end{pmatrix} = Q^T b =: \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

  Die unteren Gleichungen kann man nicht lösen, sie liefern die Fehlerterme.

  Die oberen Gleichungen liefern durch Rückwärtssubstitution die Lösung für x

  Dieses x löst auch die Normalengleichung (Beweis im Anhang)

  im Beispiel

  - Berechnung der rechten Seite

    $$Q^T b = \begin{pmatrix} 3.6000 \\ 0.8610 \\ -1.3108 \\ -1.6064 \end{pmatrix} \quad \Rightarrow \quad b_1 = \begin{pmatrix} 3.6000 \\ 0.8610 \\ -1.3108 \end{pmatrix}$$

  - Lösen des Dreiecks-Systems

    $$R_1 x = b_1$$

    $$\Rightarrow \quad \begin{pmatrix} 5 & 0.4 & 5 \\ 0 & 2.9732 & -2.0180 \\ 0 & 0 & 0.9631 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 3.6000 \\ 0.8610 \\ -1.3108 \end{pmatrix}$$

    $$\Rightarrow \quad \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 2.1317 \\ -0.6341 \\ -1.3610 \end{pmatrix}$$

- Anwendung Polynomfit:

  gegeben m Punkte (xi, yi), i = 1, .. m, mit xi ≠ xj für i ≠ j

  gesucht: Polynom vom Grad n < m-1

  $$P(x) = \sum_{i=0}^{n} a_i x^i$$

  mit

  $$P(x_k) \approx y_k \qquad k = 1 \ldots m$$

  Einsetzen der Punkte in die Polynomdefinition liefert m Gleichungen für die n+1 unbekannten Koeffizienten a_i

  $$\sum_{i=0}^{n} a_i x_k^i = y_k \qquad k = 1 \ldots m$$

überbestimmtes lineares Gleichungssystem mit Vandermonde-Matrix

$$\begin{pmatrix} 1 & x_1 & x_1^2 & \dots & x_1^n \\ 1 & x_2 & x_2^2 & \dots & x_2^n \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_m & x_m^2 & \dots & x_m^n \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix}$$

- Was kann schief gehen:

  A hat nicht Rang n
  - zu wenig Daten
  - Daten sind nicht unabhängig (schlechtes Experiment!)
  - Modell passt nicht

  Modell macht keinen Sinn
  - z.B. kein Polynom, sondern ganz anderer Zusammenhang
  - Ausgleichsrechnung liefert gut aussehende Kurve
  - Extrapolation oder Interpolation ergibt trotzdem unsinnige Werte
  - Modell kommt aus der Theorie oder Erfahrung, Mathematik kann da nicht helfen!

- Matlabfunktionen

  QR-Zerlegung der Matrix A

  ```
  [Q, R] = qr(A)
  ```

  Lösung des linearen Ausgleichsproblems A x ≈ b

  ```
  x = A \ b
  ```

  Ausgleichspolynom n-ter Ordnung zu Datenpunkten xi, yi

  ```
  poly = polyfit(xi, yi, n)
  ```

  Anwenden des Polynoms auf Werte x

  ```
  y = polyval(poly, x)
  ```

- Messwerte mit verschiedener Genauigkeit:

  gegeben seinen m Messwerte
  - (xi, yi), i = 1, .. m, mit xi ≠ xj für i ≠ j
  - jeder mit einer Genauigkeit σ_i für yi

  Die Koeffizienten a_i des Fit-Polynoms n-ten Grades erhält man durch Lösung des Ausgleichsproblems

  $$Aa \approx b$$

  mit

  $$\begin{aligned} A_{ki} &= \frac{x_k^i}{\sigma_k} \qquad k = 1 \dots m, i = 0 \dots n \\ b_k &= \frac{y_k}{\sigma_k} \qquad k = 1 \dots m \end{aligned}$$

  Beweis: [6]

  Statt Polynomen kann man auch beliebige andere Grundfunktionen verwenden:

  $$F(x) = \sum_{i=1}^{n} a_i X_i(x)$$

  Polynom wäre dann der Spezialfall

  $X_i(x) = x^{i-1}$

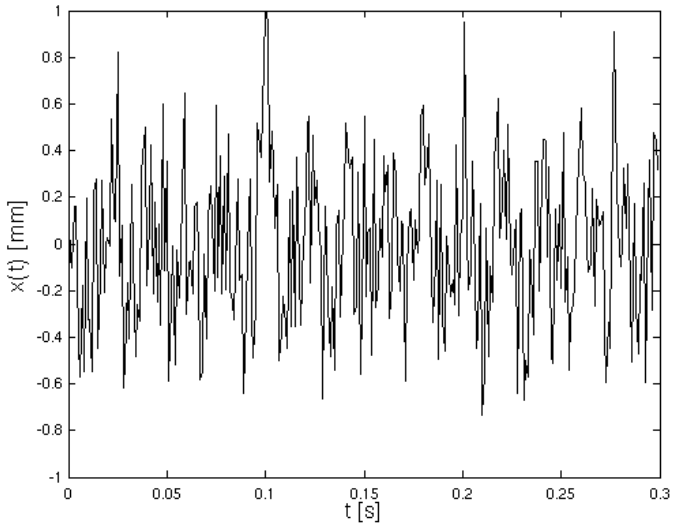
- Aufgaben:
    - Aufgabe 11
    - Aufgabe 12

Fourieranalyse

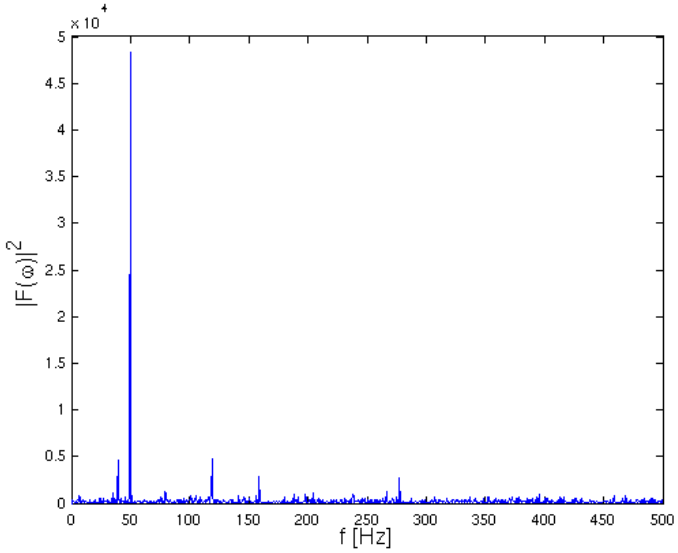
- Analyse von Störschwingungen:

  seltsame Störungen in einer Maschine, hervorgerufen durch Vibrationen unbekannter Herkunft

  Messung der Vibrationen ergibt



  Spektralanalysator zeigt



  - 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
  - besondere Störung mit Grundfrequenz 39.6 Hz

- Fourierreihe:

  Zerlegung einer periodischen Funktion f(t) mit Schwingungsdauer T in Sinus- und Kosinus-Schwingungen

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} \left(a_n \cos(n\omega t) + b_n \sin(n\omega t)\right)$$

mit ω := 2 π/T

Berechnung der Koeffizienten

$$a_n = \frac{2}{T}\int_0^T f(t)\cos(n\omega t)\,dt \qquad n = 0, 1, 2, \ldots$$
$$b_n = \frac{2}{T}\int_0^T f(t)\sin(n\omega t)\,dt \qquad n = 1, 2, 3, \ldots$$

- liefert bei endlicher Reihe beste Approximation "im quadratischen Mittel" [11]
- konvergiert für stückweise stetiges beschränktes f im Mittel [11]

Zusammenfassung als komplexe e-Funktion

$$f(t) = \sum_{n=-\infty}^{\infty} c_n e^{in\omega t}$$

Berechnung der komplexen Koeffizienten

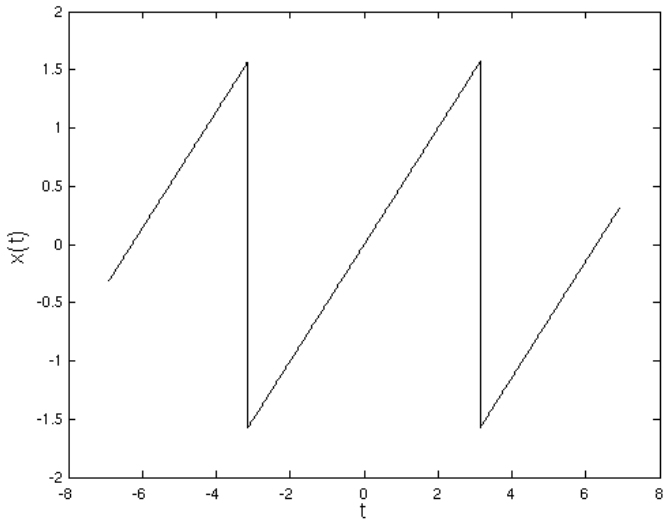
$$c_n = \frac{1}{T}\int_0^T f(t)e^{-in\omega t}\,dt \qquad n = 0, \pm 1, \pm 2, \ldots$$

Bestimmung von a_n und b_n aus c_n

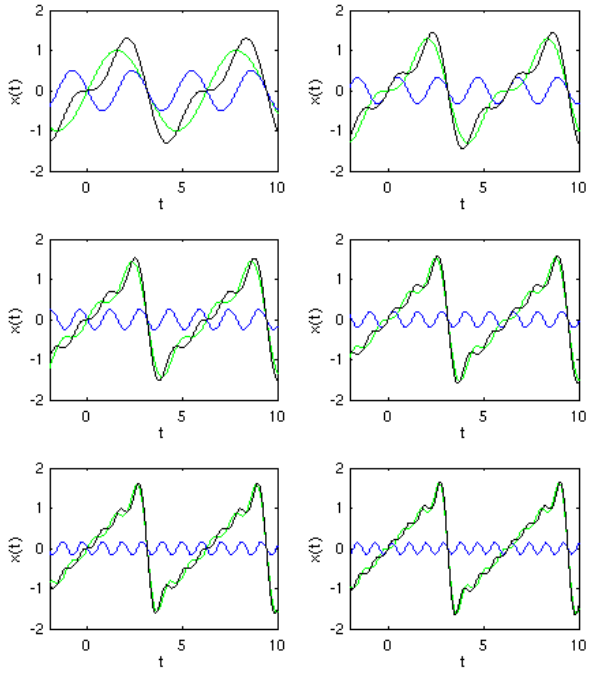
$$a_n = c_n + c_{-n}$$
$$b_n = i(c_n - c_{-n})$$

- Beispiel Sägezahnschwingung:

  f(t) = t/2 für t = -π .. π



Aufbau aus Grund- und Oberschwingungen



als Applet zum Experimentieren

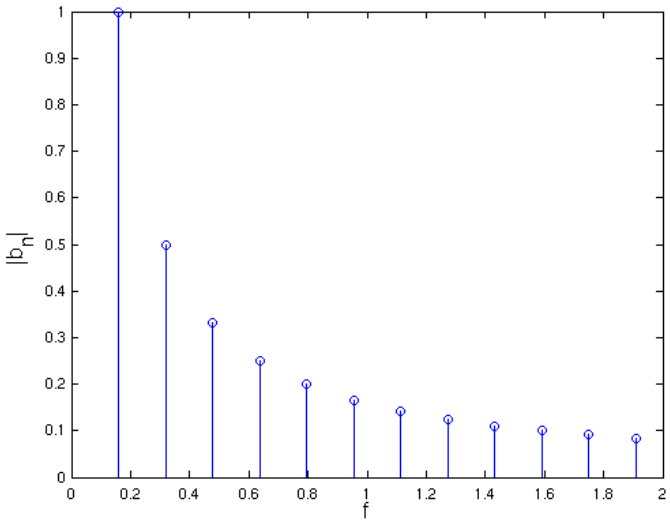
Koeffizienten

$$a_n = 0 \qquad n = 0, 1, 2, \ldots$$
$$b_n = (-1)^{n+1} \frac{1}{n} \qquad n = 1, 2, 3, \ldots$$

Fourierreihe also

f(t) = sin(t) - 1/2 sin(2t) + 1/3 sin(3t) - 1/4 sin(4t) ...

Spektrum (Darstellung der Größe der Koeffizienten über der Frequenz)

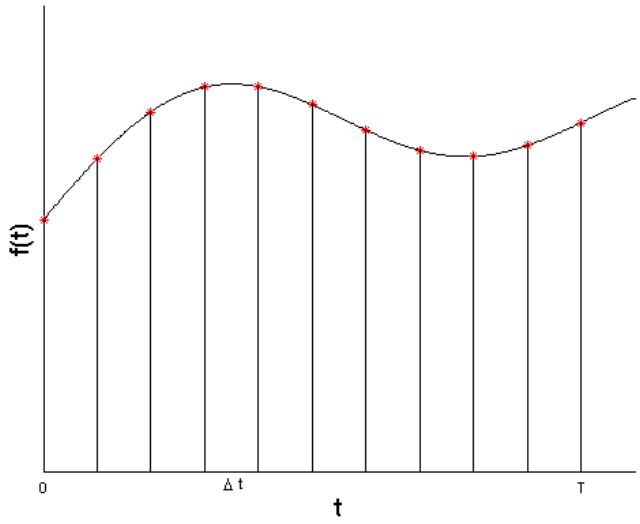


- Abtasten einer Funktion (**Sampling**):

  in der Praxis Funktion f meistens nicht explizit bekannt

  stattdessen werden Werte der Funktion f in festen Zeitabständen Δt gemessen

  $$x_n = f(n\Delta t) \qquad n = 0, 1, 2, 3, \ldots N-1$$



z.B. Messung im Schwingstand alle 1/100 s

Abtastfrequenz (**Sampling rate**)

$$f_S = \frac{1}{\Delta t}$$

insgesamt N Werte, also Messdauer

T = (N-1) Δt

- Diskrete Fouriertransformation:

  Ausgangspunkt sind N Werte x_n, n = 0, .. N - 1 (gemessen in festen Zeitabständen Δt)

  Berechnung der diskreten Fouriertransformierten X(k) (analog zu c_k) mit

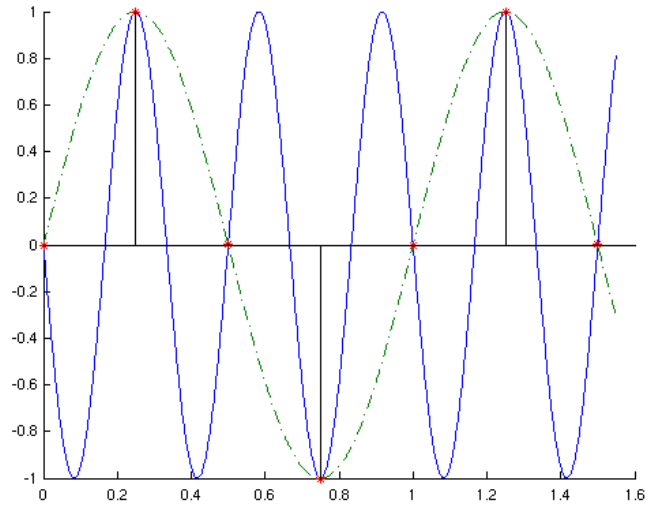
$$X_k = \sum_{n=0}^{N-1} x_n e^{-2\pi i n k/N} \qquad k = 0 \ldots N-1$$

  Rücktransformation

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{2\pi i n k/N} \qquad n = 0 \ldots N-1$$

  Achtung:

  - nur Schwingungen bis zur Nyquist-Frequenz $f_N = f_S/2$ messbar
  - Schwingungen höherer Frequenz f tauchen bei niedrigeren Werten $2 f_N - f$ auf (**Aliasing**)



- Fast Fourier Transformation (**FFT**):

  Zahl der Operationen ops(N) (Multiplikationen + Additionen) bei direkter Berechnung von X_k

  - N Multiplikationen, N-1 Additionen für ein X_k
  - N^2 Multiplikationen, N*(N-1) Additionen für alle X_k
  - → ops(N) = $2*N^2$ - N

  starke Reduktion des Aufwands bei geradem N mit "Umsortier-Trick"

  Beispiel N = 4

  - Mit der Abkürzung

    $$\omega := e^{-2\pi i/4} \quad \Rightarrow \quad \omega^4 = 1$$

    erhält man die Fouriertransformierte X_n

    $$\begin{aligned} X_0 &= \sum_{n=0}^{3} x_n \omega^{0n} = x_0 + x_1 + x_2 + x_3 \\ X_1 &= \sum_{n=0}^{3} x_n \omega^{1n} = x_0 + \omega x_1 + \omega^2 x_2 + \omega^3 x_3 \\ X_2 &= \sum_{n=0}^{3} x_n \omega^{2n} = x_0 + \omega^2 x_1 + x_2 + \omega^2 x_3 \\ X_3 &= \sum_{n=0}^{3} x_n \omega^{3n} = x_0 + \omega^3 x_1 + \omega^2 x_2 + \omega x_3 \end{aligned}$$

    was sich umsortieren lässt zu

    $$\begin{aligned} X_0 &= (x_0 + x_2) + \omega^0 (x_1 + x_3) \\ X_1 &= (x_0 + \omega^2 x_2) + \omega^1 (x_1 + \omega^2 x_3) \\ X_2 &= (x_0 + x_2) + \omega^2 (x_1 + x_3) \\ X_3 &= (x_0 + \omega^2 x_2) + \omega^3 (x_1 + \omega^2 x_3) \end{aligned}$$

  - Die Fouriertransformierte Y^g_n der "geraden Werte" x_0 und x_2 berechnet man mit

    $$\widetilde{\omega} := e^{-2\pi i/2} = \omega^2$$

    zu

    $$Y^g_0 = x_0 + x_2, \quad Y^g_1 = x_0 + \widetilde{\omega} x_2 = x_0 + \omega^2 x_2$$

  - Analog ist die Fouriertransformierte Y^u_n der "ungeraden Werte" x_1 und x_3

    $$Y^u_0 = x_1 + x_3, \quad Y^u_1 = x_1 + \omega^2 x_3$$

  - Also kann man die Fouriertransformierte der 4 Werte auf die von jeweils 2 zurückführen

    $$\begin{aligned} X_0 &= Y^g_0 + \omega^0 Y^u_0 \\ X_1 &= Y^g_1 + \omega^1 Y^u_1 \\ X_2 &= Y^g_0 + \omega^2 Y^u_0 \\ X_3 &= Y^g_1 + \omega^3 Y^u_1 \end{aligned}$$

  - Für die Zahl der Operationen ergibt sich somit

    $$\mathrm{ops}(4) = 2\,\mathrm{ops}(2) + 2 \cdot 4$$

  ganz analog zeigt man für beliebiges gerade N (s. Anhang)

  $$\mathrm{ops}(N) = 2\,\mathrm{ops}(\frac{N}{2}) + 2N$$

  falls N eine Zweierpotenz ist, folgt aus dieser Beziehung durch wiederholte Anwendung des

"Umsortier-Tricks" (s. Anhang)

$$\mathrm{ops}(N) = 2N \log_2(N)$$

dies ist für große N eine dramatische Zeitersparnis, etwa bei 1 ns pro Operation

N	ops normal	ops FFT
1024	2 ms	20 µs
1048576	2199 s	0.042 s
1073741824	73 a	64 s

1000x1000 Fouriertranformationen häufig (z.B. Bildbearbeitung)

man wählt (fast) immer N als Zweierpotenz, notfalls mit Nullen auffüllen

- FFT mit Matlab:

  grundlegende Funktionen

  - `X = fft(x);`
  - `x = ifft(X);`

  da X komplex, häufig nur Betrag oder Betragsquadrat (**Leistungsspektrum**) interessant

  ```
  F = abs(X)
  ```

  Darstellung des Spektrums

  - nur bis zur zulässigen Maximalfrequenz
  - x-Achse in richtigen Einheiten (z.B. in Hz)
  - Matlabvektoren beginnen bei 1, Frequenzen fangen bei 0 an
  - → Indizes um 1 verschieben

- Anwendungsbeispiel in Matlab:

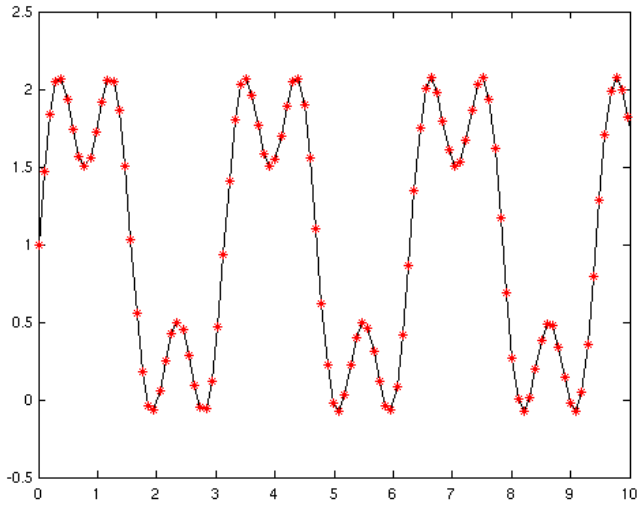
  Ausgangsfunktion

  ```
  f = @(t) 2 + sin(2*t) + 0.5*sin(6*t);
  ```

  gesampelt über Zeit T, N Werte

  ```
  T = 100;
  N = 1024;
  Delta_t = T/(N-1);
  t = [0:N-1]*Delta_t;
  x = f(t);
  ```

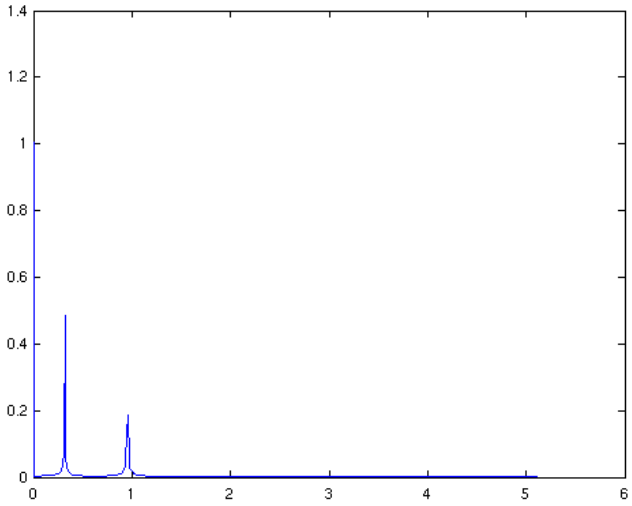
  Samplewerte (Ausschnitt!)



FFT

```
X = fft(x);
F = abs(X(1:N/2))/N;
freq = [0:(N/2)-1]/T;
plot(freq, F);
```

Spektrum



maximale Frequenz (Nyquist-Frequenz)

$$f_N = \frac{1}{2\Delta t} = 5.115$$

Wert bei f = 0 ist Mittelwert

- Aufgaben:

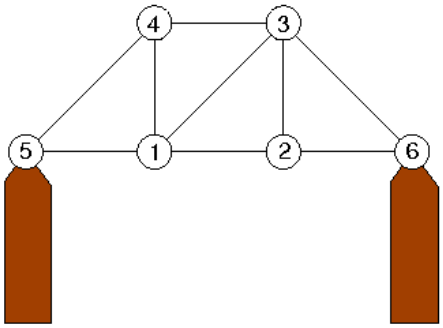
  Aufgabe 13

  Aufgabe 14

Bestimmung von Eigenwerten und Eigenvektoren

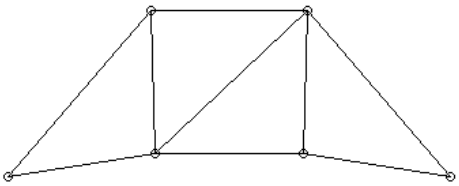
- Schwingungen eines Fachwerks:

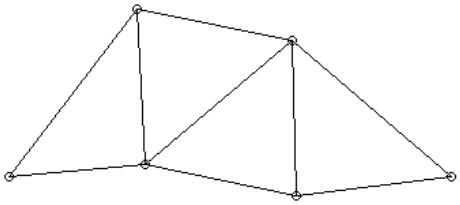
  Fachwerk



  - in den Punkten 5 und 6 fest gelagert
  - alle Knoten haben Masse m
  - Balken als Zug-/Druckfedern mit gleicher Federkonstante c

  äußere Vibrationen wirken auf das Fachwerk (durch die Lager 5, 6)
  - bei einigen Frequenzen heftiges Mitschwingen (Eigenfrequenzen)
  - jeweils typische Schwingungsformen (Eigenschwingungen)





- Definition der Eigenwertaufgabe:

  gegeben sei eine nxn-Matrix A

  gesucht sind Zahl λ und Vektor x ≠ 0 mit

  $A\,x = \lambda\,x$

  λ heißt **Eigenwert** von A, x **Eigenvektor** von A zum Eigenwert λ

  Länge von x beliebig, häufig auf 1 gesetzt (oder größte Komponente auf 1)

  Beispiel 1 (Diagonalmatrix)

$$A = \begin{pmatrix} 2 & 0 \\ 0 & 4 \end{pmatrix}$$

$$\Rightarrow \quad \lambda_1 = 2, \qquad x_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$\lambda_2 = 4, \qquad x_2 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

  Beispiel 2 (symmetrische Matrix)

$$B = \begin{pmatrix} 2 & -1 \\ -1 & 4 \end{pmatrix}$$

$$\Rightarrow \lambda_1 = 4.4142, \quad x_1 = \begin{pmatrix} 0.3827 \\ -0.9239 \end{pmatrix}$$

$$\lambda_2 = 1.5858, \quad x_2 = \begin{pmatrix} 0.9239 \\ 0.3827 \end{pmatrix}$$

Beispiel 3 (unsymmetrische Matrix)

$$C = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$$

$$\Rightarrow \lambda_1 = i, \quad x_1 = \begin{pmatrix} 1 \\ i \end{pmatrix}$$

$$\lambda_2 = -i, \quad x_2 = \begin{pmatrix} 1 \\ -i \end{pmatrix}$$

Beispiel 4 (nur 1 Eigenvektor)

$$D = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$$

$$\Rightarrow \lambda_1 = 0, \quad x_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

kein weiterer Eigenwert und Eigenvektor (bis auf Faktor)

- Eigenschaften von Matrizen:

  besondere Matrizen

  - $A^T = A$ (symmetrisch)
  - $A^T A = 1$ (orthogonal, z.B. Drehungen und Spiegelungen)
  - $(x, Ax) \geq 0$ für bel. Vektor x (positiv)

  Ähnlichkeitstransformation

  $B = U A U^{-1}$ (U beliebig, aber invertierbar)

  Satz:

  - Ähnliche Matrizen A und B haben gleiche Eigenwerte
  - x Eigenvektor von A ⇒ Ux Eigenvektor von B
  - Beweis

$$\begin{aligned} Ax &= \lambda x \\ \Rightarrow \quad B(Ux) &= UAU^{-1} \cdot Ux \\ &= UAx \\ &= \lambda(Ux) \end{aligned}$$

  Satz:

  - Sei A symmetrisch. Dann ist A zu einer reellen Diagonalmatrix D ähnlich, wobei die Transformationsmatrix U orthogonal ist:

    $A = U D U^T$ mit $U^T U = 1$

  - Insbesondere sind alle Eigenwerte reell, der k-te Eigenvektor ist die k-te Spalte von U.
  - Beweis: z.B. in [8]

  etwa bei Beispiel 2

$$U = \begin{pmatrix} 0.3827 & 0.9239 \\ -0.9239 & 0.3827 \end{pmatrix}$$

$$D = \begin{pmatrix} 4.4142 & 0 \\ 0 & 1.5858 \end{pmatrix}$$

$$\Rightarrow \quad B = UDU^t, \quad U^tU = 1$$

betrachten i.F. nur symmetrische Matrizen

- Analytische Berechnung von Eigenwerten und -vektoren:

  A nxn-Matrix, dann gilt für ein x ≠ 0

$$\begin{aligned} & Ax = \lambda x \\ \Leftrightarrow \quad & (A-\lambda 1)x = 0 \\ \Leftrightarrow \quad & \det(A-\lambda 1) = 0 \end{aligned}$$

  - also Polynom der Ordnung n für λ (charakteristisches Polynom von A)
  - hat n Lösungen (davon können mehrere zusammenfallen)

  für Beispiel 2

$$\begin{aligned} 0 &= \det\left(\begin{pmatrix} 2 & -1 \\ -1 & 4 \end{pmatrix} - \lambda \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}\right) \\ &= \det\left(\begin{pmatrix} 2-\lambda & -1 \\ -1 & 4-\lambda \end{pmatrix}\right) \\ &= (2-\lambda)(4-\lambda) - 1 \\ &= \lambda^2 - 6\lambda + 7 \\ \Rightarrow \quad \lambda_{1,2} &= 3 \pm \sqrt{9-7} = 3 \pm \sqrt{2} \\ \Rightarrow \quad \lambda_1 &= 4.4142, \quad \lambda_2 = 1.5858 \end{aligned}$$

  Eigenvektor x_1 als Lösung des homogenen System

$$\begin{aligned} & (\mathbf{B} - \lambda_1 \mathbf{1})\,\mathbf{x} = \mathbf{0} \\ \Leftrightarrow \quad & \begin{pmatrix} -2.4142 & -1 \\ -1 & -0.4142 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \mathbf{0} \end{aligned}$$

  - 1. Komponente willkührlich auf 1 setzen →

$$\begin{aligned} -2.4142 \cdot 1 - 1 \cdot y &= 0 \\ \Rightarrow \quad y &= -2.4142 \\ \Rightarrow \quad x_1 &= \begin{pmatrix} 1 \\ -2.4142 \end{pmatrix} \end{aligned}$$

  - Vektor normieren (Länge 1) durch Multiplikation mit 0.3827 →

$$x_1 = \begin{pmatrix} 0.3827 \\ -0.9239 \end{pmatrix}$$

  - analog für x_2

  Verfahren für größeres n wenig brauchbar

  - keine Lösungsformeln für n > 4
  - Nullstellensuche von Polynomen i.a. schlecht konditioniert

- QR-Verfahren:

  Grundidee:

  - Transformiere A mit geschickt gewähltem orthogonalen U

    $A' = U^T A U$

  - so dass A' größere Diagonal- und kleinere Nichtdiagonalelemente hat als A
  - Wiederhole, bis A' nahezu diagonal

  Prinzip des QR-Verfahrens

  - Mache QR-Zerlegung von A

A_0 = Q R

- neues A ist A_1 = R Q
- dies ist eine Ähnlichkeitstransformation, denn

$$\begin{aligned} A &= QR \\ \Rightarrow \quad R &= Q^T A \\ \Rightarrow \quad A_1 &= RQ = Q^T A Q \end{aligned}$$

im Beispiel

$$\begin{aligned} B_0 &= \begin{pmatrix} 2 & -1 \\ -1 & 4 \end{pmatrix} = Q_0 R_0 = \begin{pmatrix} -0.8944 & 0.4472 \\ 0.4472 & 0.8944 \end{pmatrix} \cdot \begin{pmatrix} -2.2361 & 2.6833 \\ 0 & 3.1305 \end{pmatrix} \\ B_1 &= R_0 Q_0 = \begin{pmatrix} 3.2 & 1.4 \\ 1.4 & 2.8 \end{pmatrix} = Q_1 R_1 = \begin{pmatrix} -0.9162 & -0.4008 \\ -0.4008 & 0.9162 \end{pmatrix} \cdot \begin{pmatrix} -3.4928 & -2.4049 \\ 0 & 2.0041 \end{pmatrix} \\ B_2 &= R_1 Q_1 = \begin{pmatrix} 4.1639 & -0.8033 \\ -0.8033 & 1.8361 \end{pmatrix} \end{aligned}$$

- nach 10 Iterationen wird aus B

$$B_{10} = \begin{pmatrix} 4.4142 & -0.0002 \\ -0.0002 & 1.5858 \end{pmatrix}$$

- U (und somit die Eigenvektoren) erhält man aus

$$U = Q_0 \cdot Q_1 \cdot Q_2 \cdots$$

- nach 10 Iterationen

$$U_{10} = \begin{pmatrix} 0.3828 & 0.9238 \\ -0.9238 & 0.3828 \end{pmatrix}$$

Konvergenz-Beschleunigung durch Shift

- verschiebe Eigenwerte vor der QR-Zerlegung um σ_k

$$A_k - \sigma_k 1 = Q_k R_k$$

- schiebe hinterher wieder zurück

$$A_{k+1} = R_k Q_k + \sigma_k 1$$

- sinnvoller Wert für σ_k: rechte untere Ecke von A_k

$$\sigma_k = A_k(n,n)$$

im Beispiel

$$\begin{aligned} \sigma_0 &= 4 \\ B_0 - \sigma_0 1 &= \begin{pmatrix} -2 & -1 \\ -1 & 0 \end{pmatrix} = Q_0 R_0 = \begin{pmatrix} -0.8944 & -0.4472 \\ -0.4472 & 0.8944 \end{pmatrix} \cdot \begin{pmatrix} 2.2361 & 0.8944 \\ 0 & 0.4472 \end{pmatrix} \\ B_1 &= R_0 Q_0 + \sigma_0 1 = \begin{pmatrix} 1.6 & -0.2 \\ -0.2 & 4.4 \end{pmatrix} \end{aligned}$$

schon nach 3 Iterationen erhält man die Ergebnismatrizen auf 4 signifikante Stellen

- Symmetrische Hessenberg-Form:

QR-Verfahren konvergiert erheblich schneller für Tridiagonalmatrizen (symmetrische Hessenberg-Matrizen)

$$A = \begin{pmatrix} \alpha_1 & \beta_1 & & \\ \beta_1 & \alpha_2 & \ddots & \\ & \ddots & \ddots & \beta_{n-1} \\ & & \beta_{n-1} & \alpha_n \end{pmatrix}$$

kann durch Transformation mit n-2 Householdermatrizen (s. QR-Zerlegung) aus beliebiger symmetrischer Matrix erreicht werden

$$\begin{aligned} A_1 &= Q_1 A Q_1^T \\ Q_1 &= 1 - \frac{2}{v^T v}(vv^T) \end{aligned}$$

1. Schritt mit folgendem Vektor v

$$\begin{aligned} \alpha &:= -\text{sign}(a_{21}) \sqrt{\sum_{j=2}^{n} a_{j1}^2} \\ v &= \begin{pmatrix} 0 \\ a_{21} - \alpha \\ a_{31} \\ \vdots \\ a_{n1} \end{pmatrix} \end{aligned}$$

folgende Schritte mit Teilmatrizen

Beispiel

- 1. Schritt

$$\begin{aligned} A &= \begin{pmatrix} 1 & 2 & 3 & 4 \\ \boxed{2} & 3 & 4 & 5 \\ \boxed{3} & 4 & -1 & -2 \\ \boxed{4} & 5 & -2 & -3 \end{pmatrix} \\ \alpha &= -5.3852 \\ v &= (0,\, 7.3852,\, 3,\, 4)^T \\ Q_1 &= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -0.3714 & -0.5571 & -0.7428 \\ 0 & -0.5571 & 0.7737 & -0.3017 \\ 0 & -0.7428 & -0.3017 & 0.5977 \end{pmatrix} \\ A_1 &= \begin{pmatrix} 1 & -5.3852 & 0 & 0 \\ -5.3852 & 1.2069 & 3.9138 & 5.9612 \\ 0 & 3.9138 & -0.7741 & -1.2617 \\ 0 & 5.9612 & -1.2617 & -1.4328 \end{pmatrix} \end{aligned}$$

- 2. Schritt

$$\begin{aligned} A_1 &= \begin{pmatrix} 1 & -5.3852 & 0 & 0 \\ -5.3852 & 1.2069 & 3.9138 & 5.9612 \\ 0 & \boxed{3.9138} & -0.7741 & -1.2617 \\ 0 & \boxed{5.9612} & -1.2617 & -1.4328 \end{pmatrix} \\ \alpha &= -7.1312 \\ v &= (0,\, 0,\, 11.0450,\, 5.9612)^T \\ Q_2 &= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -0.5488 & -0.8359 \\ 0 & 0 & -0.8359 & 0.5488 \end{pmatrix} \\ A_2 &= \begin{pmatrix} 1 & -5.3852 & 0 & 0 \\ -5.3852 & 1.2069 & -7.1312 & 0 \\ 0 & -7.1312 & -2.3921 & -0.1995 \\ 0 & 0 & -0.1995 & 0.1852 \end{pmatrix} \end{aligned}$$

- zusammen

$$
\begin{aligned}
A_2 &= QAQ^T \\
Q &= Q_2 Q_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -0.3714 & -0.5571 & -0.7428 \\ 0 & 0.9267 & -0.1724 & -0.3340 \\ 0 & 0.0580 & -0.8124 & 0.5803 \end{pmatrix}
\end{aligned}
$$

- Verallgemeinertes Eigenwert-Problem (**VEP**):

  gegeben seien zwei nxn-Matrix A, B

  gesucht sind Zahl λ und Vektor x ≠ 0 mit

  $A\,x = \lambda\,B\,x$

  wichtiger Spezialfall: A, B symmetrisch, B positiv und nicht singulär

  dann auf normales Eigenwert-Problem zurückführbar

  - Cholesky-Zerlegung von B

    $B = L\,L^T$

  - dann gilt

$$
\begin{aligned}
\left(L^{-1}A(L^{-1})^T\right)\cdot\left(L^T x\right) &= L^{-1}Ax \\
&= L^{-1}\lambda Bx \\
&= \lambda L^{-1}(LL^T)x \\
&= \lambda(L^T x)
\end{aligned}
$$

  also:

  - Eigenwert λ und Eigenvektor x von $L^{-1}\,A\,(L^{-1})^T$ bestimmen
  - λ ist Eigenwert des VEP
  - $(L^{-1})^T\,x$ ist Eigenvektor des VEP

- Matlab-Funktionen:

  Lösung des Eigenwertproblems

  ```
  [U,D] = eig(A)
  ```

  - $A = U\,D\,U^T$, $U^T\,U = 1$, D diagonal
  - Eigenwerte = Werte aus `diag(D)`
  - Eigenvektoren = Spalten von U

  Lösung des verallgemeinerten Eigenwertproblems

  ```
  [U,D] = eig(A,B)
  ```

  - Bedeutung wie bei `eig`

  Hessenberg-Matrix zu A

  ```
  [Q,H] = hess(A)
  ```

  - $A = Q\,H\,Q^T$ mit $Q^T\,Q = 1$

- Anwendung auf Schwingungsprobleme:

  Bewegungsgleichung für kleine Schwingungen (frei und ungedämpft)

  $$M\ddot{x}(t) + Cx(t) = 0$$

  - x(t): Vektor der Koordinaten, Auslenkungen aus der Gleichgewichtslage
  - M: symmetrische, positive Matrix (**Massenmatrix**)
  - C: symmetrische, positive Matrix (**Steifigkeitsmatrix**)

  Ansatz

  $x(t) = \hat{x}\cdot e^{i\omega t}$

$x(t) = \hat{x} \cdot e$

liefert

$$\begin{aligned} -\omega^2 M\hat{x} + C\hat{x} &= 0 \\ \Leftrightarrow \qquad C\hat{x} &= \omega^2 M\hat{x} \end{aligned}$$

also verallgemeinertes Eigenwert-Problem mit

- Eigenwert ω^2
- Eigenvektor $\hat{x}$

Lösung liefert Schwingung mit Eigenfrequenz $\omega = 2\,\pi\,f$

- Aufgaben:

  Aufgabe 15

  Aufgabe 16

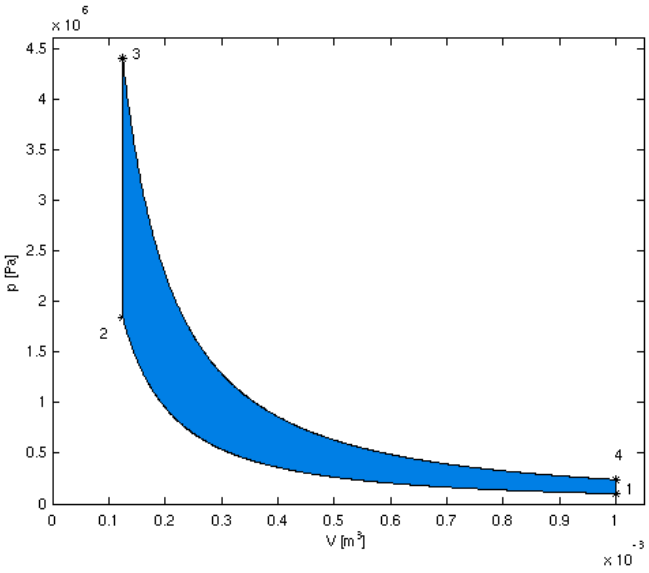
Numerische Integration

- Leistung eines Otto-Motors:

  vereinfachter Kreisprozess

  - 1 → 2 Kompression ohne Wärmezufuhr (**adiabatisch**)
  - 2 → 3 Zündung → sehr schnelle Drucksteigerung, Wärmezufuhr näherungsweise bei konstantem Volumen
  - 3 → 4 adiabatische Ausdehnung
  - 4 → 1 Druckminderung bei konstantem Volumen, Ausstoßen der Abgase, Ansaugen des neuen Gemischs

  Darstellung von Druck über Volumen (**p-V-Diagramm**)



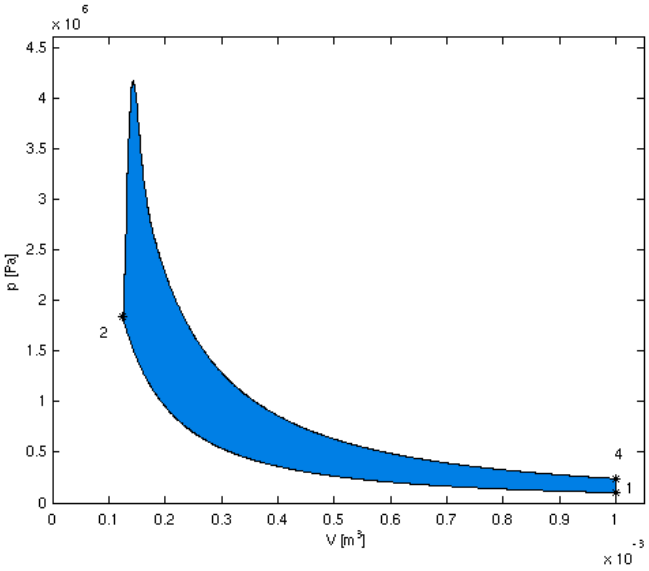
  umschlossene Fläche = abgegebene Arbeit W bei einem Umlauf

  Leistung P bei Motordrehzahl n

$$P = \frac{n}{2} W$$

  (Arbeitstakt nur jede 2. Umdrehung)

  realistischer mit Modell für Verbrennungsvorgang



- Problemstellung:

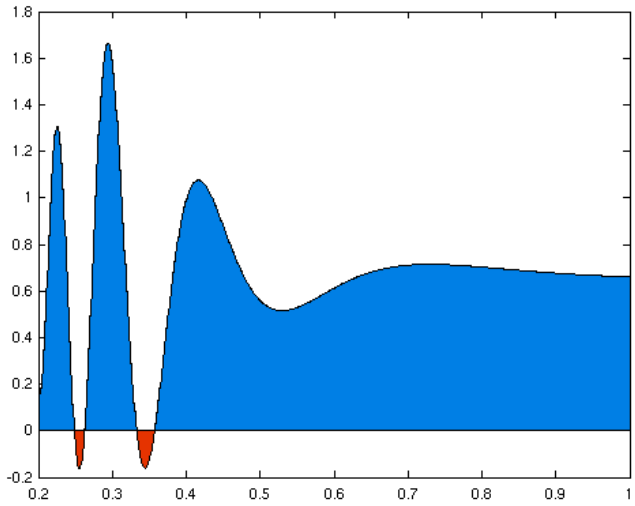
  berechne für gegebene Funktion f und gegebene Grenzen a, b das bestimmte Integral

$$I = \int_a^b f(x)dx$$

Standardbeispiel im Folgenden

$$I = \int_{0.2}^{1.0} \left(\frac{\sin\left(\frac{6}{x}\right)}{1+(10x-3)^2} + \frac{2}{3} \right) dx$$

im Bild



Wert

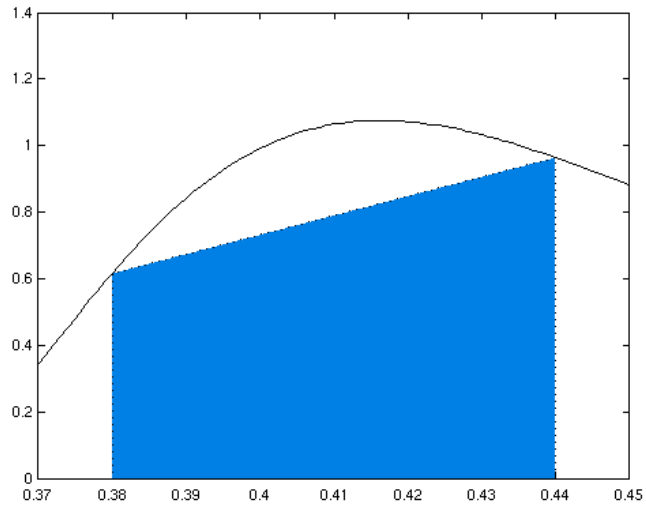
I = 0.535914332007100

- Trapezregel:

  Aufteilen des Intervalls [a, b] in N Teilintervalle $[x_i, x_{i+1}]$ ($x_0 = a$, $x_N = b$) der Länge

  $$h = \frac{b-a}{N}$$

  Approximation von f auf einem Teilintervall $[x_i, x_i + h]$ durch eine Gerade durch die Randpunkte $(x_i, f(x_i))$ und $(x_{i+1}, f(x_{i+1}))$



  Integralnäherung auf dem Intervall

$$I_h = \frac{h}{2}\left(f(x_i) + f(x_i + h)\right)$$

Näherung des gesamten Integrals durch Addition

$$\begin{aligned} I_T &= \frac{h}{2}\sum_{i=0}^{N-1}\left(f(x_i) + f(x_{i+1})\right) \\ &= \frac{h}{2}\left(f(a) + 2\sum_{i=1}^{N-1} f(x_i) + f(b)\right) \end{aligned}$$

Genauigkeit $O(h^2)$ [8]

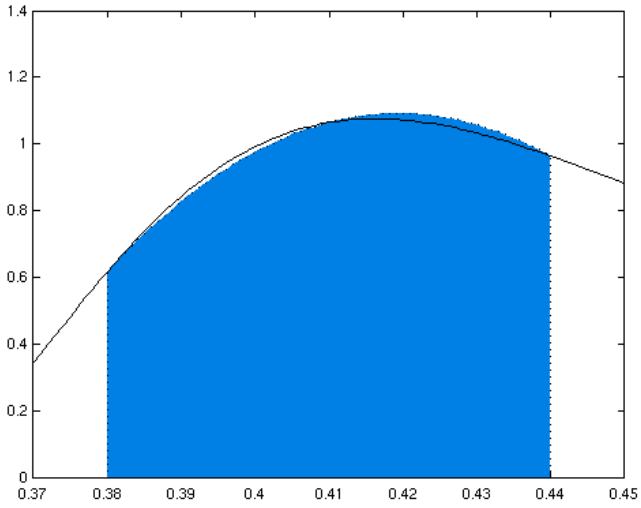
im Standardbeispiel

h	**I_T**	**rel. Fehler**
1.000000e-01	0.62504156147227	1.663087e-01
1.000000e-02	0.53605438650191	2.613375e-04
1.000000e-03	0.53591569981964	2.552297e-06
1.000000e-04	0.53591434568167	2.551634e-08
1.000000e-05	0.53591433214385	2.551738e-10
1.000000e-06	0.53591433200846	2.539420e-12
1.000000e-07	0.53591433200717	1.282347e-13

Verfahren leicht auf unterschiedliche Intervallgrößen erweiterbar
→ gut geeignet für durch Messwerte gegebene Funktionen

- Simpsonregel:

  Idee wie bei Trapezregel, aber f durch Parabel approximieren



  Parabel durch Punkte bei x_i, $x_i + h/2$, $x_i + h$

$$\begin{aligned} P(x) &= \frac{1}{h^2}\left((2y_1 - 4y_2 + 2y_3)x^2\right. \\ &\quad -((4x_i + 3h)y_1 - (8x_i + 4h)y_2 + (4x_i + h)y_3)x \\ &\quad \left.+((2x_i^2 + 3x_i h + h^2)y_1 + (2x_i^2 + x_i h)y_3 - 4(x_i^2 + x_i h)y_2)\right) \end{aligned}$$

  mit $y_1 = f(x_i)$, $y_2 = f(x_i + h/2)$, $y_3 = f(x_i + h)$

  Integralnäherung auf dem Intervall

$$I_h = \frac{h}{6}\left(f(x_i) + 4f(x_i + \frac{h}{2}) + f(x_i + h)\right)$$

Näherung des gesamten Integrals durch Addition

$$I_S = \frac{h}{6}\left(f(a) + 2\sum_{i=1}^{N-1} f(x_i) + 4\sum_{i=0}^{N-1} f(x_i + \frac{h}{2}) + f(b)\right)$$

Genauigkeit $O(h^4)$ (!) [8]

im Standardbeispiel

h	I_S	rel. Fehler
1.000000e-01	0.47345537553892	1.165465e-01
1.000000e-02	0.53591352330050	1.509022e-06
1.000000e-03	0.53591433191739	1.673978e-10
1.000000e-04	0.53591433200709	1.553732e-14
1.000000e-05	0.53591433200710	2.693136e-15
1.000000e-06	0.53591433200709	1.553732e-14

- Verbesserung der Verfahren:

  höhere Polynom-Ordnung bringt nichts, da höhere Interpolationspolynome stark schwingen

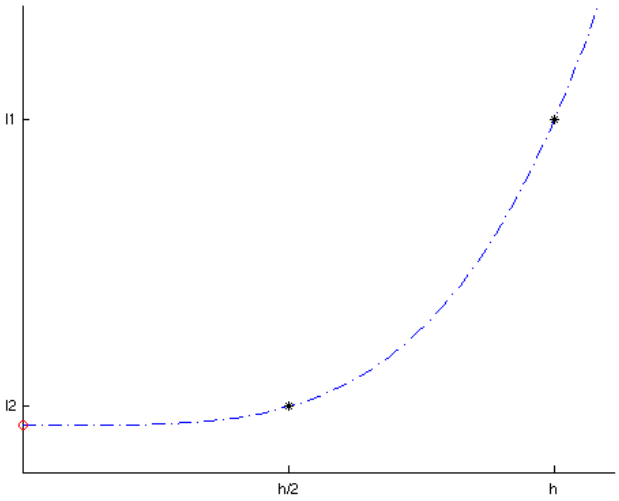
  zwei versch. Ansätze
  - Erhöhung der Ordnung durch Extrapolation
  - feste Ordnung, aber Anpassung der Schrittweite

- Adaptives Simpson-Verfahren:

  Grundidee: Intervalle dort verfeinern, wo Teilergebnis ungenauer als vorgegebene Toleranz δ

  Fehler-Abschätzung auf einem Teilintervall der Breite h
  - ein Simpson-Schritt → I_1
  - Intervall halbieren, zwei Simpson-Schritte → I_2
  - $| I_2 - I_1 | < \delta$ → Schritt ok
  - sonst: Intervall halbieren und mit jedem Teilintervall weitermachen

  Ergebnis I_2 kann noch verbessert werden durch Extrapolation



$$
\begin{aligned}
I_1 &= I + ah^4 && |\cdot(-1)\\
I_2 &= I + a\left(\frac{h}{2}\right)^4 && |\cdot 16\\
\Rightarrow \quad 16I_2 - I_1 &= 15I\\
\Rightarrow \quad I &= \frac{16I_2 - I_1}{15}
\end{aligned}
$$

extrapolierter Wert hat höhere Fehlerordnung (hier $O(h^6)$!)

Gesamtfehler a priori schwer abschätzbar

- Einzelfehler können viel kleiner sein als δ
- Anzahl der Intervalle variabel

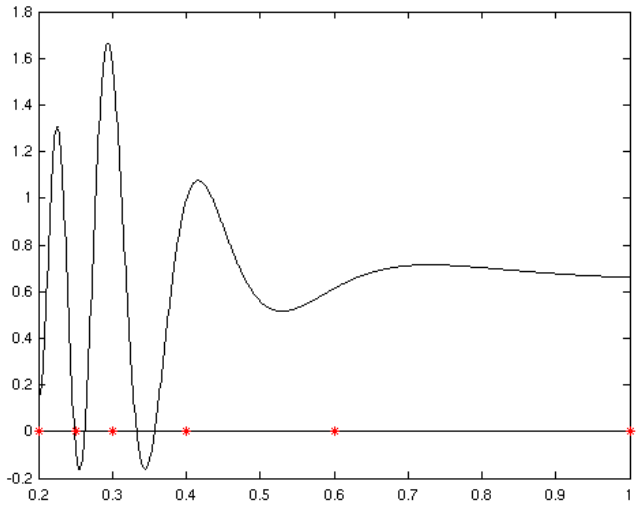
- Adaptive Integration am Beispiel:

  Standardbeispiel mit Genauigkeit δ = 0.02

  1. I = 0.200, r = 1.000 → I1 = 0.4377, I2 = 0.5891 halbieren
  2. I = 0.200, r = 0.600 → I1 = 0.3168, I2 = 0.3775 halbieren
  3. I = 0.600, r = 1.000 → I1 = 0.2723, I2 = 0.2743 ok
  4. I = 0.200, r = 0.400 → I1 = 0.2494, I2 = 0.0598 halbieren
  5. I = 0.400, r = 0.600 → I1 = 0.1280, I2 = 0.1393 ok
  6. I = 0.200, r = 0.300 → I1 = 0.0254, I2 = 0.0848 halbieren
  7. I = 0.300, r = 0.400 → I1 = 0.0345, I2 = 0.0458 ok
  8. I = 0.200, r = 0.250 → I1 = 0.0445, I2 = 0.0355 ok
  9. I = 0.250, r = 0.300 → I1 = 0.0403, I2 = 0.0391 ok

  Ergebnis: I = 0.5350 (exakt: 0.5359)

  Verteilung der Intervalle



- Probleme bei der Integration:

  Pseudo-Singularitäten (sin(x)/x)

  Singularitäten am Rand

  Singularitäten im Intervall → Integral aufspalten

  unendliche Intervalle

- Matlab-Funktionen:

  Integration mit adaptivem Verfahren

```
I = integral(fun, a, b)
```

mit Vorgabe der Genauigkeit

```
I = integral(fun, a, b, "RelTol", tol)
```

alternatives Verfahren, auch bei Singularitäten am Rand oder unendlichem Intervall

`I = quadgk(fun, a, b)` mit Standardgenauigkeit

`I = quadgk(fun, a, b, "AbsTol", tol)` mit absoluter Genauigkeit tol

`I = quadgk(fun, a, b, "RelTol", tol)` mit relativer Genauigkeit tol

Berechnung des Integrals zu Messpunkten X, Y mit Trapezregel

```
I = trapz(X,Y)
```

- Weitere Methoden:

  sukzessive Erhöhung der Ordnung durch Extrapolation (**Romberg-Verfahren**)

  Wahl von Stützpunkten mit verschiedenem Abstand (**Gauß-Formeln**)

  Integral als Lösung der Differentialgleichung

  y' = f(x), y(0) = a

  Lösung bis y(b)

  Mehrfach-Integrale
  - i.a. schwierig
  - komplizierte Integrationsgebiete statt einfachem Intervall
  - viele ganz spezifische Verfahren (z. B. Montecarlo-Integration)

- Aufgaben:

  Aufgabe 17

  Aufgabe 18

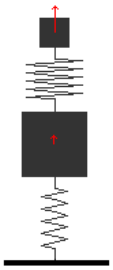
Lösung von Differentialgleichungen

- Schwingungstilger:

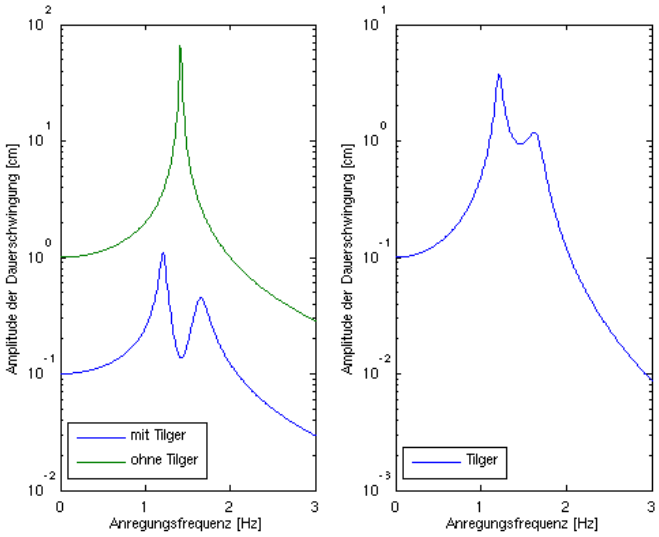
  Problem: schwach gedämpftes System gerät durch äußere Anregung in starke Schwingungen

  Beispiele
  - Lenksäule durch Motorvibrationen
  - Prüfmaschine durch Motorunwucht
  - freie Treppe durch Fußgänger
  - Mast einer Windkraftanlage durch Rotorbewegung

  Abhilfe: Einbau eines Tilgers = zusätzliche schwingende Masse, die Schwingungsenergie aufnimmt



  Dauerschwingungen von Masse und Tilger mit Standardverfahren der Schwingungslehre berechenbar



  aber: wie stark schwingt der Tilger am Anfang (Einschwingverhalten)?

  ausprobieren mit Applet

  Lösung der Bewegungsgleichung erforderlich

- Existenz und Eindeutigkeit von Lösungen [9]:
  - Anfangswertproblem

    Gesucht ist eine vektorwertige Funktion **y**(t), die die Differentialgleichung

    $$\dot{\vec{y}}(t) = \vec{f}(t, \vec{y}(t))$$

    löst und die Anfangsbedingung

    $$\vec{y}(t_0) = \vec{y}_0$$

erfüllt.

- Satz von Peano:

  Ist f stetig, so hat das Anfangswertproblem eine Lösung (für eine Umgebung von t_0).

- Satz von Picard-Lindelöf

  Erfüllt f die lokale Lipschitz-Bedingung

  $$|\vec{f}(t,\vec{y}_1) - \vec{f}(t,\vec{y}_2)| \leq L\,|\vec{y}_1 - \vec{y}_2|$$

  dann hat das Anfangswertproblem eine eindeutige Lösung.

  Hinreichend für die Lipschitz-Bedingung ist die Existenz und Beschränktheit der partiellen Ableitung $\frac{\partial \vec{f}}{\partial \vec{y}}$.

  Beispiel

  Das Anfangswertproblem

  $$\dot{y} = \sqrt{y}, \quad y(0) = 0$$

  hat (u.a.) die zwei Lösungen

  $$\begin{aligned} y_1(t) &= \frac{1}{4}t^2 \\ y_2(t) &= 0 \end{aligned}$$

- Differentialgleichungen höherer Ordnung:

  lassen sich zurückführen auf 1. Ordnung durch Einführen der niedrigeren Ableitungen als zusätzliche Variable

  Beispielproblem

  $$\ddot{x} = -\omega^2 x, \quad x(0) = x_0, \quad \dot{x}(0) = v_0$$

  $\dot{x}$ als zweite Variable einführen

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

  dann erhält man ein zweidimensionales System 1. Ordnung

  $$\dot{\vec{y}} = \begin{pmatrix} \dot{x} \\ \ddot{x} \end{pmatrix} = \begin{pmatrix} \dot{x} \\ -\omega^2 x \end{pmatrix} = \begin{pmatrix} y_2 \\ -\omega^2 y_1 \end{pmatrix} =: \vec{f}(t,\vec{y}(t))$$

  mit der Anfangsbedingung

  $$\vec{y}(0) = \begin{pmatrix} x_0 \\ v_0 \end{pmatrix}$$

- Euler-Verfahren:

  schrittweise von einem Punkt y(t) zu einem nächsten y(t + h) vorantasten

  für kleine Schrittweite h gilt in 1. Ordnung in h

  $$\begin{aligned} y(t+h) &= y(t) + h\dot{y}(t) \\ &= y(t) + hf(t,y(t)) \end{aligned}$$

  Beispiel "exponentieller Zerfall"

  $$\dot{y} = -\lambda y, \quad y(0) = 1$$

  mit λ = 0.1

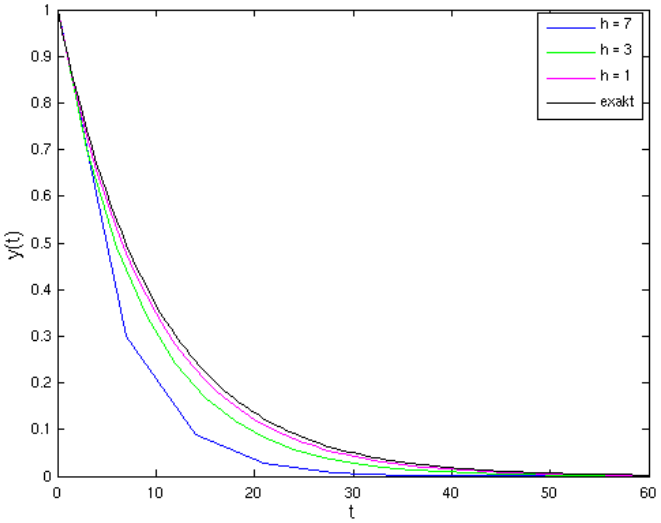
  mit der konkreten Funktion f(t,y) = -λ y

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

schrittweise erhält man die Näherungslösung zu

$$\begin{aligned} y(h) &= (1-h\lambda)\, y(0) \\ y(2h) &= (1-h\lambda)\, y(h) = (1-h\lambda)^2\, y(0) \\ y(3h) &= (1-h\lambda)\, y(2h) = (1-h\lambda)^3\, y(0) \\ y(Nh) &= (1-h\lambda)^N\, y(0) \end{aligned}$$

im Bild



- Methode von Heun:

  Verbesserung der Genauigkeit durch Mittelwert über mehrere Steigungen

  Mittelwert über Steigung k_1 bei t und k_2 bei t+h

  $$y(t+h) = y(t) + h\,\frac{k_1+k_2}{2}$$

  Steigung k_1 wie bei Euler direkt aus der Gleichung

  $$k_1 = f(t,y)$$

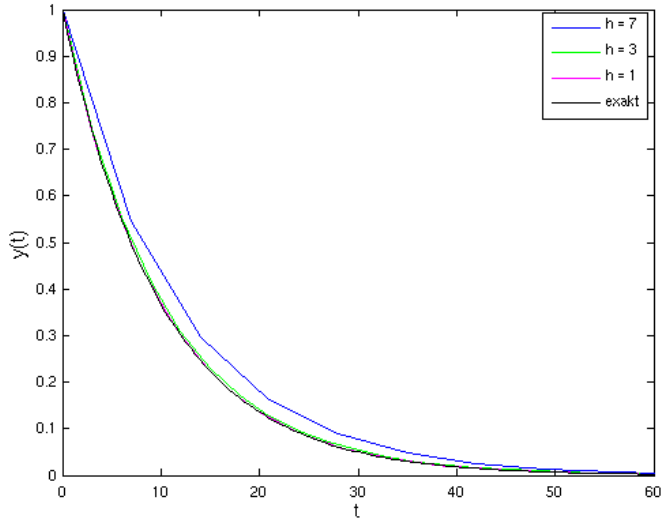
  Steigung k_2 bei t+h geschätzt mit Eulerschritt

  $$k_2 = f(t+h, y+h\,k_1)$$

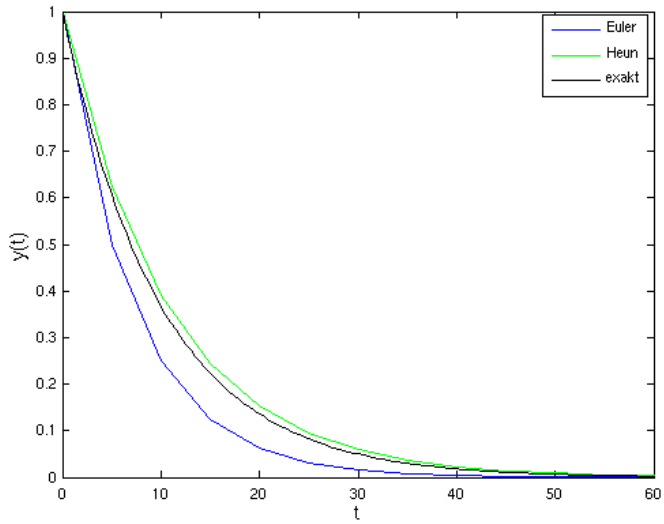
  im Beispiel "exponentieller Zerfall"

  $$\begin{aligned} k_1 &= f(t,y) = -\lambda y \\ k_2 &= f(t+h, y+h\,k_1) = -\lambda(y+h\,k_1) = -\lambda y\,(1-\lambda h) \\ y(t+h) &= y(t) + h\,\frac{k_1+k_2}{2} = \left(1-\lambda h + \frac{1}{2}(\lambda h)^2\right) y(t) \end{aligned}$$

  im Bild



Vergleich mit Euler bei h = 5



- Genauigkeit des Heun-Verfahrens:
  - Taylorreihe für y(t+h) bis zur 2. Ordnung in h

    $$y(t+h) = y(t) + h\dot{y}(t) + \frac{1}{2}h^2\ddot{y}(t)$$

    Einsetzen der DGL →

    $$y(t+h) = y(t) + hf + \frac{1}{2}h^2\dot{f}$$

    wobei die Argumente (t, y(t)) jeweils weggelassen werden
  - Berechnen der Ableitung von f mit den partiellen Ableitungen

    $$\begin{aligned}\frac{\partial f}{\partial t}(t,y) &=: f_t \\ \frac{\partial f}{\partial y}(t,y) &=: f_y\end{aligned}$$

    und der Kettenregel

    $$\begin{aligned}\dot{f}(t,y(t)) &= f_t + f_y\dot{y}(t) \\ &= f_t + f_y f\end{aligned}$$

    ergibt

$$y(t+h) = y(t) + hf + \frac{1}{2}h^2(f_t + f_y f)$$

- dagegen ist beim Heun-Verfahren, bis zur 2. Ordnung gerechnet

$$\begin{aligned} y_H(t+h) &= y(t) + \frac{h}{2}k_1 + \frac{h}{2}k_2 \\ &= y(t) + \frac{h}{2}f + \frac{h}{2}f(t+h, y(t)+hf)) \\ &= y(t) + \frac{h}{2}f + \frac{h}{2}[f + hf_t + hff_y] \\ &= y(t) + hf + \frac{1}{2}h^2(f_t + f_y f) \end{aligned}$$

- Genauigkeit eines Schritts also 2. Ordnung in h

- Runge-Kutta-Verfahren:

  Mittelung über mehrere Zwischensteigungen k_i, so dass hohe Ordnungen in h erreicht werden

  allgemeines Schema bei s Stufen

$$\begin{aligned} k_i &= f(t + a_i h, y(t) + h\sum_{j=1}^{i-1} b_{ij}k_j), \quad i = 1, 2, \ldots, s \\ y(t+h) &= y(t) + h\sum_{i=1}^{s} c_i k_i \end{aligned}$$

  benötigte Werte

a_i	Zwischenpositionen	immer $a_1 = 0$
b_{ij}	Bestimmen von k_i aus älteren k_j	untere Dreiecksmatrix
c_i	Kombination der k_i	

  angegeben als Butcher-Diagramm

$$\begin{array}{c|c} a & B \\ \hline & c^T \end{array}$$

  etwa bei Heun

$$\begin{array}{c|cc} 0 & & \\ 1 & 1 & \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$$

  "klassisches" Runge-Kutta-Verfahren 4. Ordnung

$$\begin{array}{c|cccc} 0 & & & & \\ \frac{1}{2} & \frac{1}{2} & & & \\ \frac{1}{2} & 0 & \frac{1}{2} & & \\ 1 & 0 & 0 & 1 & \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array}$$

  Berechnung der Koeffizienten für höhere Ordnungen sehr aufwändig

- Schrittweitensteuerung:

  Idee: passe Schrittweite h an Problembereich an
  - kleines h bei schnellen Änderungen
  - großes h bei Bereichen mit wenig Schwankungen

  Vorteile

- sehr viel schneller
- lokaler Fehler an vorgegebene Genauigkeit tol anpassbar

einfachstes Verfahren

- 1 Schritt mit h
- 2 Schritte mit h/2
- daraus lokalen Fehler ε abschätzen
- ε ≈ tol → genaueren Wert nehmen, h lassen
- ε < tol → genaueren Wert nehmen, h vergrößern
- ε > tol → Wert nicht nehmen, h verkleinern und Schritt wiederholen

viele Details müssen geklärt werden

- Anfangswert für h
- Bereich für ε, so dass h gleich bleibt
- konkrete Vergrößerung/Verkleinerung von h
- wie klein darf h werden ?

- Eingebettete Runge-Kutta-Verfahren RKp(q):

  effizientere Methode

  - Schritt h mit zwei Ordnungen p und q rechnen
  - daraus Fehler ε abschätzen

  liefern neuen Wert der Ordnung p und Fehlerschätzung

  verwenden dafür jeweils im wesentlichen die gleichen k_i-Werte

  Beispiel RK3(2) von Bogacki und Shampine

  - Tableau für den neuen Wert

$$\begin{array}{c|ccc} 0 & & & \\ \frac{1}{2} & \frac{1}{2} & & \\ \frac{3}{4} & 0 & \frac{3}{4} & \\ \hline & \frac{2}{9} & \frac{3}{9} & \frac{4}{9} \end{array}$$

  - Berechnung des Fehlers ε verwendet neuen Wert y(t+h)

$$\begin{aligned} k_4 &= f(t+h, y(t+h)) \\ \varepsilon &= \frac{h}{72}|-5k_1+6k_2+8k_3-9k_4| \end{aligned}$$

  bestes bekanntes Verfahren RK5(4) von Dormand und Prince

- Algorithmus von ode23 [1]:

  leicht vereinfacht

  - einige mögliche Fehlersituationen nicht abgeprüft
  - Annahme $t_1 > t_0 \geq 0$

  Vorgaben

$$\begin{aligned} \varepsilon_{\text{rel}} &= 1 \cdot 10^{-3} \\ \delta &= 1 \cdot 10^{-3} \qquad \text{(kleinster Nenner)} \end{aligned}$$

  Anfangswert für h

$$\begin{aligned} k_1 &= f(t_0, y_0) \\ r &= \frac{|k_1|}{\max(|y_0|, \delta)} \\ h &= \frac{0.8}{\cdot} \sqrt[3]{\varepsilon_{\text{rel}}} \end{aligned}$$

r

h-Schritt

y(t+h) und ε mit RK3(2) berechnen

Prüfen des lokalen Fehlers

$$y_{\text{abs}} = \min(|y(t)|, |y(t+h)|)$$
$$\varepsilon_{\text{lok}} = \frac{\varepsilon}{\max(y_{\text{abs}}, \delta)}$$

- $\varepsilon_{\text{lok}} \leq \varepsilon_{\text{rel}} \rightarrow$

  y-Wert wird akzeptiert

  $k_1 = k_4$ (braucht nicht neu berechnet zu werden)

Berechnung des neuen h

$$h = h \min\left(5, 0.8 \sqrt[3]{\frac{\varepsilon_{\text{rel}}}{\varepsilon_{\text{lok}}}}\right)$$

- Matlab-Funktionen:

  rechte Seite f(t,y) der Differentialgleichung
  - als Matlabfunktion erstellen
  - Ergebnis ist ein Spaltenvektor!

  diverse DGL-Solver
  - `ode23`, `ode45`, `ode113`, `ode15s`, `ode23s`, `ode23t`, `ode23tb`
  - sinnvoller Standardsolver: `ode45`

  Aufruf jeweils

  ```
  [t,y] = odeXY(func, tSpan, y0 [,options]);
  ```

  - mit

`func`	Funktion f(t,y)
`tSpan = [t0 t1]`	Start- und Endzeit
`y0`	Vektor mit Startwerten y(t0)
`options`	optionale Struktur mit Optionen
`t`	Ergebnisvektor mit Zeiten t_i
`y`	Ergebnismatrix mit Funktionswerten $y(t_i)$ eine Spalte pro Komponente von y

  Verändern der Standardoptionen

  ```
  options = odeset("name1",value1,"name2",value2,...)
  ```

  - einige Optionen

Name	**Standardwert**	**Bedeutung**
`RelTol`	`1e-3`	relativer Fehler
`AbsTol`	`1e-6`	absoluter Fehler
`Refine`	`1`	Faktor für zusätzliche Ausgabezeitpunkte
`Stats`	`"off"`	Ausgabe von Statistikdaten bei `"on"`

- Aufgaben:

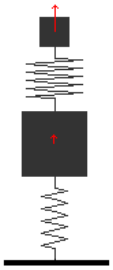
  Aufgabe 19

  Aufgabe 20

Steife Probleme

- Steifer Tilger:

  betrachtet wird wieder das System aus Schwinger und Tilger von oben

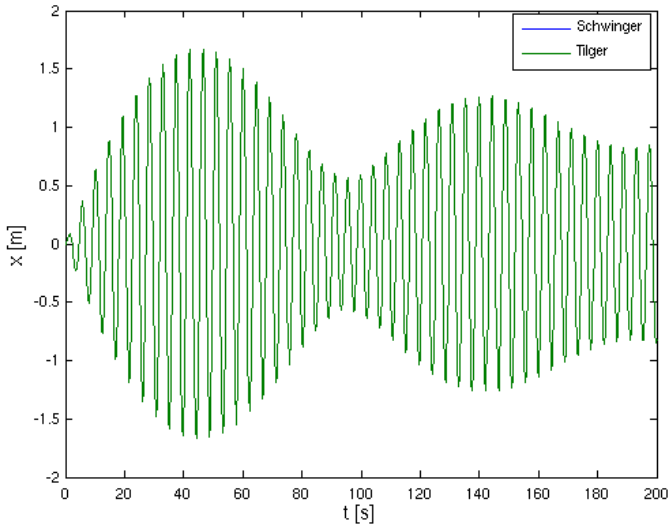


  was geschieht, wenn die Feder zum Tilger immer steifer wird (d.h. ihre Federkonstante und Dämpfung nehmen stark zu)?

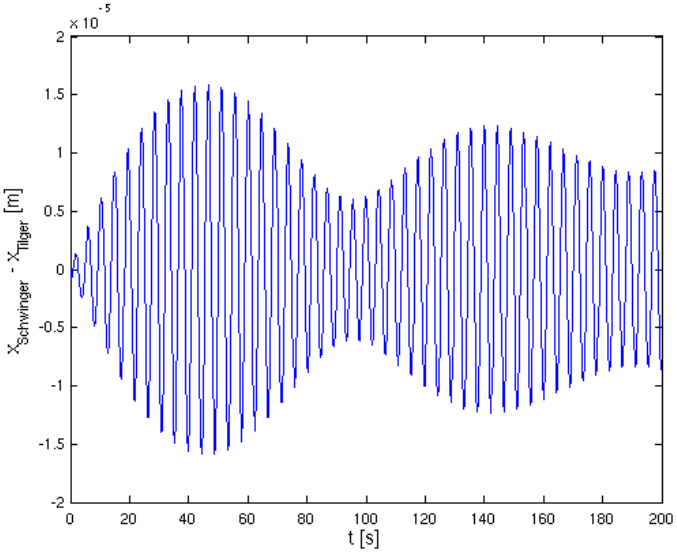
  Erwartung

  - Tilger und Schwinger sind quasi "fest" miteinander verbunden
  - Schwinger bewegt sich wie ohne Tilger (bis auf dessen kleine Zusatzmasse)
  - Tilger bewegt sich in festem Abstand mit dem Schwinger mit

  Ergebnis der Simulation für Faktor 100000



  - Schwinger und Tilger schwingen genau im Takt, minimaler Unterschied



aber: Rechenzeit des Solvers (`ode45`) ist dramatisch angestiegen (Faktor 1000!)

- Schrittweiten beim Beispiel "Radioaktiver Zerfall":

  System gegeben durch

  $$\dot{y} = \lambda y, \quad y(0) = 1$$

  mit λ < 0

  i. F. grundlegendes Beispiel

  Simulation mit adaptivem Solver `ode45` mit verschiedenen Genauigkeiten `tol` und für verschiedene Werte des Parameters λ

  Zeitbereich jeweils von t = 0 bis t = 60

  gemessen wird Zahl der Zeitschritte mit `length(t)`

  Ergebnisse

tol	**1e-3**	**1e-4**	**1e-5**	**1e-6**
-λ				
0.1	45	53	69	81
10	773	793	809	817
1000	72365	72381	72401	72409

  1. Beobachtung:
     - Zahl der Zeitschritte steigt nur langsam mit `tol`
     - Ursache: einfache glatte Lösungsfunktion exp(λ t)

  2. Beobachtung:
     - Zahl der Zeitschritte wächst stark für (betragsmäßig) großes λ
     - unverständlich, denn Lösung ist im Rahmen der geforderten Toleranz fast = 0!

  typisches Verhalten "steifer" Systeme

- Untersuchung der Ursache beim Eulerverfahren:

  Eulerverfahren liefert für N-ten Zeitschritt

  $$y(Nh) = (1 + h\lambda)^N y(0)$$

  für h λ < -1 wechselndes Vorzeichen

  für h λ < -2 sogar betragsmäßig exponentieller Anstieg

  also für großes negatives λ

- Lösung ist völlig harmlos (i.w. = 0)
- Verfahren funktioniert nur für sehr kleine Schrittweiten

ähnliches Verhalten bei allen bisher behandelten Verfahren

Grundproblem:

- winzig kleine Störungen erzwingen winzige Schrittweiten
- → lange Rechenzeiten
- → ungenau wegen Aufschaukeln kleiner Fehler

- A-Stabilitätsgebiet eines Solvers:

  Bereich für z = h λ (als komplexe Variable gedacht), in dem ein Verfahren für das Beispiel eine nicht-ansteigende Lösung liefert, d.h. es gilt

  $$\frac{|y((n+1)h)|}{|y(nh)|} \leq 1 \quad \text{mit } n = 1, 2, 3, \ldots$$

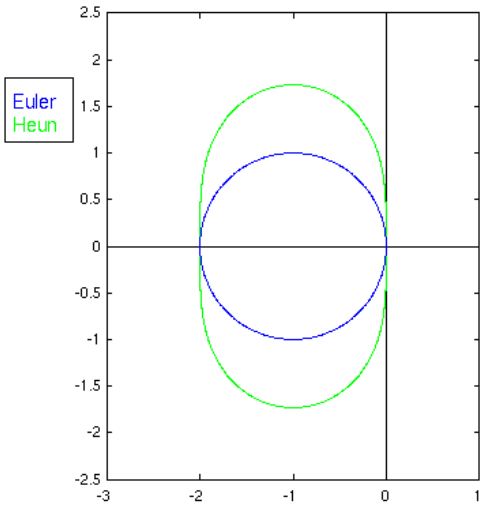
  Beispiel Euler

  $$\begin{aligned} \frac{|y((n+1)h)|}{|y(nh)|} &= |1 + h\lambda| \\ \Rightarrow \qquad |1+z| &\leq 1 \end{aligned}$$

  Beispiel Heun

  $$|1 + z + \frac{1}{2}z^2| \leq 1$$

  im Bild



- Implizites Eulerverfahren:

  bei Euler: Ableitung am Startpunkt

  $$\begin{aligned} y(t+h) &= y(t) + h\dot{y}(t) \\ &= y(t) + hf(t, y(t)) \end{aligned}$$

  - bei exponentiellem Abfall dort noch zu groß → Überschießen

  statt dessen: Ableitung am Endpunkt

  $$\begin{aligned} y(t+h) &= y(t) + h\dot{y}(t+h) \\ &= y(t) + hf(t+h, y(t+h)) \end{aligned}$$

  - dort Ableitung schon klein → ok

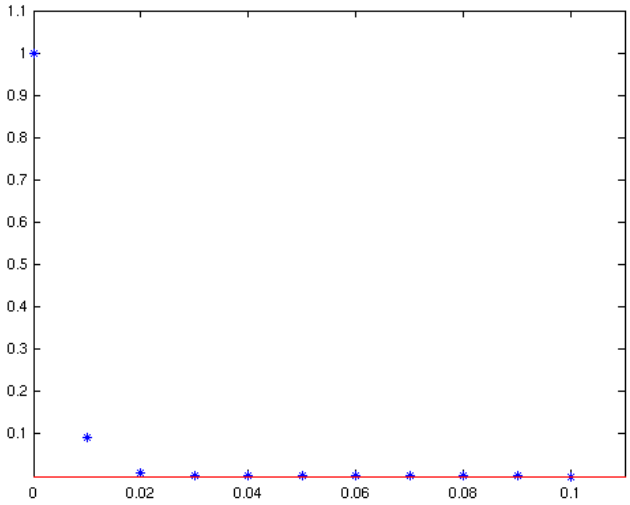
  Problem

- gesuchte Größe y(t + h) auf beiden Seiten der Gleichung (nur implizit gegeben)
- steht in Funktion f → Auflösung eines nichtlinearen Gleichungssystems nötig
- iterativ, etwa mit Newton (mehrdimensional - i. a. sehr aufwändig!)
- Vorteil: guter Startwert der Iteration vom letzten Zeitschritt

im Beispiel einfach explizit auflösen

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

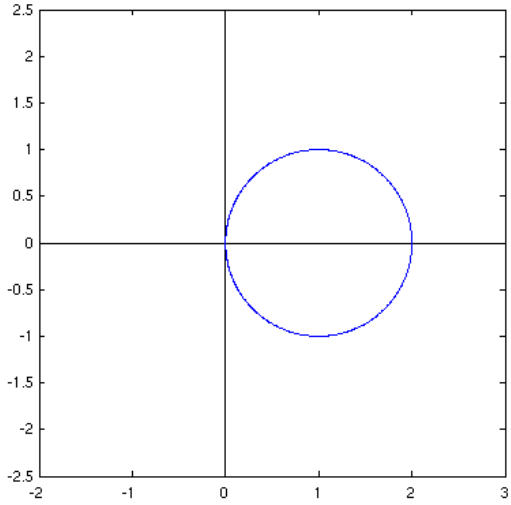
Ergebnis für λ = -1000 und Schrittweite 0.01



Stabilitätsgebiet

$$\frac{1}{|1-z|} \leq 1$$

- im Bild



- insbesondere für alle λ< 0 stabil
- auch für viele λ > 0 stabil (unerwünscht!)

- Implizite Verfahren höherer Ordnung:

  Trapez-Verfahren (2. Ordnung)
  - Integriere Differentialgleichung

$$\dot{\vec{y}}(t) = \vec{f}(t, \vec{y}(t))$$

über [t, t+h] →

$$\begin{aligned} \vec{y}(t+h) - \vec{y}(t) &= \int_t^{t+h} \vec{f}(t', \vec{y}(t'))dt' \\ &\approx \frac{h}{2}\left[\vec{f}(t+h, \vec{y}(t+h)) + \vec{f}(t, \vec{y}(t))\right] \end{aligned}$$

mit der Trapezregel

- Stabilitätsgebiet: Im(z) < 0 (optimal)

weitere Verfahren als Runge-Kutta-Schema mit vollbesetzter Matrix **B**

Newton-Verfahren zur Lösung

- braucht Jacobimatrix **J** der Differentialgleichung

$$J(t, y) = \frac{\partial \vec{f}}{\partial \vec{y}}(t, y)$$

- alternativ Ableitung numerisch nähern

- Beispiel van-der-Pool-Oszillator:

  gegeben durch

  $$\ddot{x} - \mu(1 - x^2)\dot{x} + x = 0$$

  als System

  $$\dot{\vec{y}} = \begin{pmatrix} \dot{y}_1 \\ \dot{y}_2 \end{pmatrix} = \begin{pmatrix} y_2 \\ \mu(1 - y_1^2)y_2 - y_1 \end{pmatrix} = \vec{f}(t, \vec{y})$$

  zugehörige Jacobimatrix

  $$\vec{J}(t, \vec{y}) = \begin{pmatrix} \frac{\partial f_1}{\partial y_1} & \frac{\partial f_1}{\partial y_2} \\ \frac{\partial f_2}{\partial y_1} & \frac{\partial f_2}{\partial y_2} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -2\mu y_1 y_2 - 1 & \mu(1 - y_1^2) \end{pmatrix}$$

  in Matlab

  - Funktionen **f**(t,**y**) und **J**(t,**y**) definieren
  - Jacobimatrix als Option bekanntmachen
  - `options = odeset("Jacobian", J);`
  - mit steifem Solver lösen, z.B. `ode15s`

- Aufgaben:

  Aufgabe 21

  Aufgabe 22

Aufgaben

Aufgabe 1

- Berechnen Sie die Fließkommanäherung fl(x) der Zahlen $x_1 = 0.03125$, $x_2 = 10000$ und $x_3 = 0.2$
  a. für eine Numerik mit $\beta = 10$, $t = 3$, $e_{min} = -3$, $e_{max} = 3$
  b. für double-Werte gemäß IEEE 754
  c. Geben Sie die genaue Bitdarstellung (der Übersichtlichkeit wegen hexadezimal) der Werte aus b. an.
- Lösung

Aufgabe 2

a. Verwenden Sie die Taylorreihe der Exponentialfunktion

$$e^x \approx \sum_{k=0}^{N} \frac{x^k}{k!}$$

um die Werte von exp(x) und exp(-x) für x = 10 zu bestimmen. Probieren Sie verschiedene Werte für N und untersuchen Sie die erhaltenen relativen Genauigkeiten.

b. Vergleichen Sie mit dem Verhalten bei x = 20. Was geschieht?

c. Wie könnte man den Wert für exp(-x) genauer bestimmen?

- Lösung

Aufgabe 3

a. Bestimmen Sie die LU-Zerlegung der Matrix

$$A = \begin{pmatrix} 50 & -20 & 91 \\ -29 & -59 & 10 \\ 110 & 90 & 81 \end{pmatrix}$$

durch Ausführen des Gaußschen Algorithmus mit Spalten-Pivotisierung mit Matlab als "Taschenrechner". Runden Sie dabei alle Zwischenergebnisse auf vier signifikante Stellen. Überprüfen Sie explizit, dass

L U = P A

b. Verwenden Sie das Ergebnis aus a., um die Lösung der Gleichung Ax = b für b = (89, -185, 449)^T zu bestimmen. Vergleichen Sie mit dem exakten Ergebnis x = (4, 1, -1)^T: Wie groß sind der relative Fehler und das Residuum?

c. Lösen Sie das Problem nun mit Matlab (mit voller Genauigkeit). Bestimmen Sie auch hier Fehler und Residuum sowie die Konditionszahl von A. Ist die Ungleichung

$$\frac{|x - x_*|}{|x|} \leq \mathrm{cond}(A) \cdot \frac{|r|}{|b|}$$

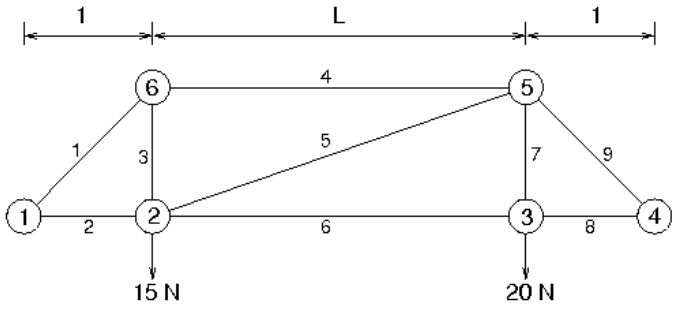
erfüllt? Stimmt sie auch für die Ergebnisse von b. ?

- Lösung

Aufgabe 4

a. Bestimmen Sie die Balkenkräfte für das Fachwerk vom Anfangsbeispiel. Wie groß ist die Konditionszahl der Systemmatrix?

b. Der Abstand zwischen den Gelenken 2-3 und 5-6 sei um einen Faktor L größer als vorher:



Stellen Sie das zugehörige Gleichungssystem auf und lösen Sie es für L = 1000. Wie groß ist nun die Konditionszahl des Systems?

- Lösung

Aufgabe 5

- Ermitteln Sie die Nullstelle der Funktion

$$f(x) = \sin\left(\frac{1}{x}\right)$$

im Intervall [0.107, 0.28] auf 5 signifikante Stellen

a. mit dem Bisektionsverfahren,

b. mit dem Sekantenverfahren,

c. mit dem IQI-Verfahren,

Führen Sie alle Verfahren schrittweise durch. Natürlich lässt sich alles mit dem Taschenrechner erledigen, es geht aber deutlich schneller, wenn Sie sich kleine Hilfsfunktionen in Matlab schreiben, die jeweils einen einzelnen Bisektions-, Sekanten- oder IQI-Schritt durchführen.

- Lösung

Aufgabe 6

a. Stellen Sie die Kräftebilanz für das Eingangsbeispiel (Feder-Masse-System) auf und verifizieren Sie die angegebene Gleichung.

b. Schreiben Sie je eine Matlabroutine für das Bisektions- und das Sekantenverfahren und bestimmen Sie damit die Gleichgewichtsposition für c = 20 N/m, m = 1 kg, l = 0.3 m, g = 9.81 m/s^2. Vergleichen Sie die Ergebnisse mit dem der `fzero`-Funktion von Matlab.

c. Reduzieren Sie die Zahl der Parameter durch geschickte Umformung und Definition neuer Hilfsgrößen von 4 auf 1. Bestimmen Sie die Gleichgewichtsposition für c = 40 N/m, m = 6 kg, l = 0.9 m, g = 9.81 m/s^2 (ohne erneutes Lösen der Gleichung).

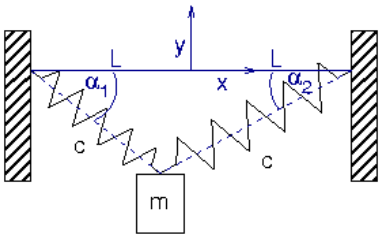
- Lösung

Aufgabe 7

- Lösen Sie Aufgabe 5 mit dem Dekker-Brent-Verfahren auf 4 Dezimalen Genauigkeit.
- Lösung

Aufgabe 8

- Gesucht ist die Gleichgewichtsposition einer Masse zwischen zwei Federn wie in Aufgabe 6, wobei diesmal die Federn unterschiedliche Ruhelängen L_1, L_2 haben :



a. Stellen Sie die Kräftebilanz auf und zeigen Sie, dass sich die Gleichgewichtsposition (x, y) als Lösung des folgenden Systems ergibt:

$$F_1(x,y) = \left(1 - \frac{L_1}{\sqrt{(x+L)^2+y^2}}\right)(x+L) + \left(1 - \frac{L_2}{\sqrt{(x-L)^2+y^2}}\right)(x-L) = 0$$

$$F_2(x,y) = \left(1 - \frac{L_1}{\sqrt{(x+L)^2+y^2}}\right)y + \left(1 - \frac{L_2}{\sqrt{(x-L)^2+y^2}}\right)y + \frac{mg}{c} = 0$$

b. Bestimmen Sie die Lösung für die Werte

   c = 20 N/m, m = 1 kg, g = 9.81 m/s^2, L = 0.3 m, L_1 = 0.25 m, L_2 = 0.2 m

c. Mit den Ruhelängen $L_1 = L_2$ = 0.5 m und c = 70 N/m gibt es mehrere Gleichgewichtspositionen. Wie viele? Berechnen Sie sie.

- Lösung

Aufgabe 9

- Gegeben seien folgende Punkte:

  $x_i = [1, 2, 3, 4, 5]$, $y_i = [1, 2, 1, 1, 4]$

  a. Bestimmen Sie das Interpolationspolynom durch diese Punkte.

  b. Berechnen Sie die Spline-Interpolationsfunktionen
     1. als natürlichen Spline,
     2. als not-a-knot-Spline.

- Lösung

Aufgabe 10

- Bestimmen Sie mit Matlab zu den Daten aus cp.dat das Interpolationspolynom. Plotten Sie in einem Diagramm die Messpunkte, die lineare, Polynom- und Spline-Interpolation.
- Wie groß ist der c_p-Wert bei t = 15 °C für die einzelnen Interpolationsverfahren?
- Lösung

Aufgabe 11

- Bestimmen Sie die Ausgleichsparabel 2. Ordnung zu den Messwerten

  $x_i = [1, 2, 3, 4]$, $y_i = [3, 2, 3, 7]$

  indem Sie die Schritte des Verfahrens nachvollziehen:

  a. Aufstellen des überbestimmten Gleichungssystems
  b. QR-Zerlegung der Systemmatrix (mit Matlab oder "per Hand")
  c. Lösen der Normalengleichung und Angabe des Polynoms

- Plotten Sie die Punkte und das Polynom in einem Diagramm.
- Lösung

Aufgabe 12

a. Berechnen Sie mit Matlab zu den cp-Messwerten von Wasser (s. cp.dat) die Ausgleichspolynome 1. - 4. und 8. Ordnung. Plotten Sie die Daten und die Ausgleichskurven in einem Diagramm. Welche Kurve erscheint Ihnen am besten geeignet?

b. Wiederholen Sie a. für die 4. Ordnung, wobei Sie zusätzlich zu den Messwerten noch deren Messgenauigkeiten σ_i verwenden (s. cps.dat). Vergleichen Sie die Ergebnisse aus a und b.

- Lösung

Aufgabe 13

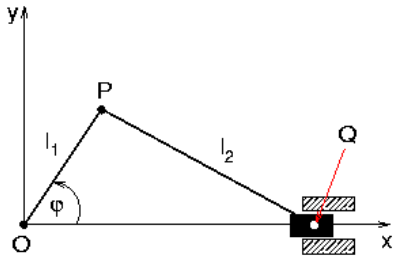
- Die Datei stoerung.dat enthält die Messwerte der Störschwingung des Eingangsbeispiels in der Form

  Zeitwert [s] Messwert [mm]

  Reproduzieren Sie mit Matlab daraus das abgebildete Leistungsspektrum.
- Wie groß ist die Nyquist-Frequenz? Gibt es Probleme mit Aliasing?
- Lösung

Aufgabe 14

- Zur Umsetzung einer Drehbewegung in eine Schubbewegung kann ein Schubkurbelgetriebe verwendet werden:



  a. Bestimmen Sie die Ortskurve x(t) des Endpunkts Q bei konstanter Antriebsfrequenz ω und plotten Sie sie für die Werte λ = 0.4 und λ = 0.99, wobei λ := l_1/l_2 das Schubstangenverhältnis bezeichnet.

  b. Bestimmen Sie das Spektrum der Schubkurbelschwingung für die beiden Werte von λ. Experimentieren Sie mit den Werten für die Messzeit T und die Zahl N der Punkte. Welche Werte sind minimal zu wählen?

- Lösung

Aufgabe 15

- Führen Sie das QR-Verfahren mit folgender Beispielmatrix durch:

$$A = \begin{pmatrix} 9 & 6 & -5 & -2 \\ 6 & 9 & -5 & 2 \\ -5 & -5 & 7 & 0 \\ -2 & 2 & 0 & 7 \end{pmatrix}$$

  Iterieren Sie jeweils solange, bis die Matrix "hinreichend diagonal" ist, konkret, bis

$$\delta := \frac{\sum_{i \neq j} |a_{ij}|}{\sum_{i} |a_{ii}|} < \varepsilon$$

  mit $\varepsilon = 10^{-4}$. Dabei sind die a_{ij} die Elemente der Matrix A.

- Wie viele Schritte sind nötig
  a. ohne Shift,
  b. ohne Shift, aber mit vorheriger Hessenberg-Transformation (mit Matlabs `hess`-Funktion),
  c. mit Shift ?
- Vergleichen Sie jeweils die erhaltenen Eigenwerte und Eigenvektoren mit dem von Matlab berechneten Ergebnis.
- Lösung

Aufgabe 16

- Um die Eigenschwingungen des Beispiel-Fachwerks zu berechnen, werden die zweidimensionalen Auslenkungen der vier Knotenpunkte zu einem Vektor zusammengefasst:

  $$x = (x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4)^T$$

- Die Bewegungsgleichung kann dann wie üblich geschrieben werden als

  $$M\ddot{x}(t) + Cx(t) = 0$$

  wobei die Matrizen gegeben sind durch

$$M = m \cdot 1$$

$$C = c \cdot \begin{pmatrix} 2.5 & 0.5 & -1 & 0 & -0.5 & -0.5 & 0 & 0 \\ 0.5 & 1.5 & 0 & 0 & -0.5 & -0.5 & 0 & -1 \\ -1 & 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 \\ -0.5 & -0.5 & 0 & 0 & 2 & 0 & -1 & 0 \\ -0.5 & -0.5 & 0 & -1 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 1.5 & 0.5 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0.5 & 1.5 \end{pmatrix}$$

- Bestimmen Sie die Eigenfrequenzen und Eigenschwingungen des Fachwerks für m = 1 kg und c = 1 N/m.
- Zeichnen Sie die 1. Eigenschwingung ein, indem Sie die zweidimensionalen Verschiebungsvektoren an den Knotenpunkten eintragen (per Hand oder mit Matlab). Skizzieren Sie ebenso die 2. Eigenschwingung (in einer zweiten Zeichnung).
- Lösung

Aufgabe 17

- Berechnen Sie das Integral

$$I = \int_{0.4}^{4.0} \cos(10\, e^{-2(x-1)^2}) dx$$

  mit Hilfe des adaptiven Simpson-Verfahren für eine Genauigkeit δ = 0.01. Geben Sie die verwendeten Zwischenpunkte an.

- Lösung

Aufgabe 18

a. Bestimmen Sie die Leistung des Ottomotors vom Eingangsbeispiel bei einer Drehzahl von 3000 U/min für folgende Werte:

Punkt	V [dm^3]	p [bar]
1	1.000	1.000
2	0.125	18.379
3	0.125	43.979
4	1.000	2.393

Die Kurven 1 → 2 und 3 → 4 werden jeweils durch die Adiabatengleichung beschrieben

$$p\,V^{\kappa} = \text{const.}$$

wobei der Adiabatenkoeffizient den Wert $\kappa = 1.40$ hat.

b. Wiederholen Sie a., wobei die Kurven 2 → 3 und 3 → 4 ersetzt werden durch folgende Kurve p(V):

$$\begin{aligned} \alpha &= 30 \\ m &= 1.7 \\ x &= \frac{V - V_2}{V_2} \\ y(x) &= 1 + (\alpha m x^m - 1) e^{-\alpha x^m} \\ p(V) &= p_2(1-y) + y p_3 \left(\frac{V_3}{V}\right)^{\kappa} \end{aligned}$$

- Lösung

Aufgabe 19

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

  $$m\ddot{x} = -mg - b\dot{x}^2 \operatorname{sign}(\dot{x})$$

  mit den Parameterwerten m = 75 kg, g = 9.81 m/s^2, b = 0.1 kg/m und den Anfangsbedingungen x(0) = 5000 m, v(0) = 0.

  Verwenden Sie dazu die Kuttasche 3/8-Regel, ein Runge-Kutta-Verfahren, das durch das folgende Diagramm definiert ist:

  $$\begin{array}{c|cccc} 0 & & & & \\ \frac{1}{3} & \frac{1}{3} & & & \\ \frac{2}{3} & -\frac{1}{3} & 1 & & \\ 1 & 1 & -1 & 1 & \\ \hline & \frac{1}{8} & \frac{3}{8} & \frac{3}{8} & \frac{1}{8} \end{array}$$

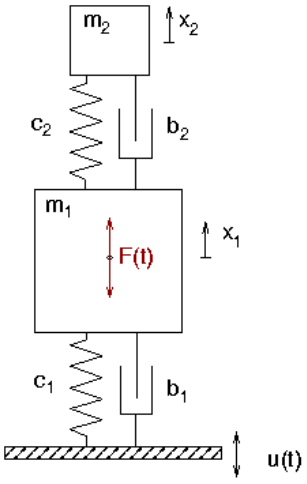
  Benutzen Sie eine feste Schrittweite von h = 10 und lösen Sie bis t = 70 s.

- Ermitteln Sie zum Vergleich die Lösung an den gleichen Stellen mit Hilfe von Matlabs Solver ode45 und stellen Sie die Ergebnisse für x(t) tabellarisch gegenüber.

- Lösung

Aufgabe 20

- Ein System aus schwingender Masse und Tilger sei durch folgende Werte beschrieben:



  - m_1 = 5 kg
  - m_2 = 0.5 kg
  - c_1 = 10 N/m
  - c_2 = 1 N/m
  - b_1 = 0.1 Ns/m
  - b_2 = 0.1 Ns/m

- Durch Bodenvibrationen u(t) wird auf den Schwinger eine harmonische Kraft F(t) übertragen:

$$F(t) = F_1 \cos \Omega t \qquad \text{mit} \quad F_1 = 1\,\text{N}\,,\ \Omega = 1.414\,\frac{1}{\text{s}}$$

- Die Bewegungsgleichung des Systems lautet

$$\vec{M}\ddot{\vec{x}} + \vec{B}\dot{\vec{x}} + \vec{C}\vec{x} = \vec{F}(t)$$

$$\vec{M} = \begin{pmatrix} m_1 & 0 \\ 0 & m_2 \end{pmatrix}, \quad \vec{C} = \begin{pmatrix} c_1 + c_2 & -c_2 \\ -c_2 & c_2 \end{pmatrix}, \quad \vec{B} = \begin{pmatrix} b_1 + b_2 & -b_2 \\ -b_2 & b_2 \end{pmatrix},$$

$$\vec{F}(t) = \begin{pmatrix} F_1 \cos \Omega t \\ 0 \end{pmatrix}, \quad \vec{x} = \begin{pmatrix} x_1(t) \\ x_2(t) \end{pmatrix}$$

  Anfangswerte können zu 0 angenommen werden.

- Bestimmen Sie die Amplitude der Dauerschwingung von m_1 ohne und mit Tilger sowie die größte Auslenkung des Tilgers.

- Lösung

Aufgabe 21

- Lösen Sie das Anfangswertproblem

$$\ddot{x} + \omega^2 x = 0, \quad x(0) = 1,\ \dot{x}(0) = 0$$

  für $\omega = 8$ bis zur Zeit $t_1 = 0.5$, indem Sie den obigen Algorithmus von ode23 für einen relativen Fehler $\varepsilon_{rel} = 0.1$ durchspielen.

- Vergleichen Sie Ihr Ergebnis mit dem von Matlabs Solver `ode23` bei gleicher relativer Genauigkeit.

- Lösung

Aufgabe 22

- Berechnen Sie die Lösung der van-der-Pool-Gleichung

$$\ddot{x} - \mu(1 - x^2)\dot{x} + x = 0$$

für μ = 10 und die Anfangsbedingung x(0) = 1, $\dot{x}$(0) = 0 für t = [0 60] in festen Schritten h = 0.01. Verwenden Sie dazu

  a. den Solver `ode23t` ohne explizite Angabe der Jacobimatrix,

  b. den Solver `ode23t` mit expliziter Angabe der Jacobimatrix,

  c. einen eigenen Solver, der die Trapezregel mit Newtonverfahren implementiert. (*)

- Lösung

Anhang

Literatur

1. C. Moler: Numerical Computing With Matlab
   Society for Industrial & Applied Mathematics, 2. Aufl. 2010, ISBN 978-0898716603
2. M. Knorrenschild: Numerische Mathematik: Eine beispielorientierte Einführung
   Carl Hanser Verlag, 6. Aufl. 2017, ISBN 978-3446451612
3. Hans-Rudolf Schwarz: Numerische Mathematik
   Vieweg+Teubner Verlag, 8. Aufl. 2011, ISBN: 978-3834815514
4. Burden, Faires, Burden: Numerical Analysis.
   Cengage Learning Inc, 10. Ed. 2015, ISBN 978-1305253667
5. Ramin S. Esfandiari: Numerical Methods for Engineers and Scientists Using MATLAB
   Taylor & Francis, 2. Ed.. 2017, ISBN: 978-1498777421
6. Press, Flannery, Teukolsky: Numerical Recipes: The Art of Scientific Computing
   Cambridge University Press, 3. Aufl. 2007, ISBN 978-0521880688
7. G. Opfer: Numerische Mathematik für Anfänger
   Vieweg+Teubner, 5. Aufl. 2008, ISBN 978-3834804136
8. Deuflhard, Hohmann: Numerische Mathematik I
   Gruyter, 5. Aufl 2018, ISBN 978-3110614213
9. Deuflhard, Bornemann: Numerische Mathematik II
   Gruyter, 4. Aufl. 2013, ISBN 978-3110316339
10. Bronstein, Semendjajew, Musiol: Taschenbuch der Mathematik
    Europa-Lehrmittel-Verlag, 11. Aufl. 2010, ISBN 978-3-8085-5792-1
11. A. Meister: Numerik linearer Gleichungssysteme: Eine Einführung in moderne Verfahren.
    Springer Spektrum, 5. Aufl. 2015, ISBN: 978-3658071998
12. R. P. Brent, An algorithm with guaranteed convergence for finding a zero of a function
    Computer Journal 14 (1971), 422-425.

Herleitungen

- Relativer Fehler beim linearen Gleichungssystem
- Bestimmung der Koeffizienten für kubische Splines
- Herleitung der Normalengleichung
- Lösung der Normalengleichung
- Aufteilungsschritt beim FFT-Verfahren
- Zahl der Operationen beim FFT-Verfahren

Relativer Fehler beim linearen Gleichungssystem

- Sei A eine nicht-singuläre n x n -Matrix, b ein n-elementiger Spaltenvektor. Für das Gleichungsystem

  A x = b

  sei x_* eine Näherungslösung mit dem Residuum r. Dann gilt für den relativen Fehler

  $$\frac{|x - x_*|}{|x|} \leq \text{cond}(A) \cdot \frac{|r|}{|b|}$$

- Beweis

  Nach der Definition des Residuums gilt

  $$\begin{aligned} Ax &= b \qquad (1) \\ Ax_* &= b - r \qquad (2) \end{aligned}$$

  Subtraktion der beiden Gleichungen und Linksmultiplikation mit A^{-1} liefert

  $$x - x_* = A^{-1}r$$

  Daher

  $$\begin{aligned} |x - x_*| \cdot |b| &= |A^{-1}r| \cdot |Ax| \\ &\leq \|A^{-1}\| \cdot |r| \cdot \|A\| \cdot |x| \\ &= \text{cond}(A) \cdot |r| \cdot |x| \\ \Rightarrow \quad \frac{|x - x_*|}{|x|} &\leq \text{cond}(A) \cdot \frac{|r|}{|b|} \end{aligned}$$

Bestimmung der Koeffizienten für kubische Splines

- Das Polynom $P_k(x)$ zwischen den Punkten (x_k, y_k) und (x_{k+1}, y_{k+1}) sei gegeben durch

$$\begin{aligned} P_k(x) &= \frac{1}{h_k^3}\left((3h_k - 2s_k)s_k^2 y_{k+1} + (h_k + 2s_k)(h_k - s_k)^2 y_k \right. \\ &\quad \left. + s_k^2(s_k - h_k)h_k M_{k+1} + s_k(s_k - h_k)^2 h_k M_k\right) \end{aligned}$$

wobei wieder

$$\begin{aligned} h_k &:= x_{k+1} - x_k \quad k = 1 \ldots N-1 \\ s_k &:= x - x_k \quad\quad k = 1 \ldots N-1 \end{aligned}$$

Seine Ableitungen lassen sich leicht berechnen zu

$$\begin{aligned} P_k'(x) &= \frac{1}{h_k^3}(6s_k(h_k - s_k)y_{k+1} - 6s_k(h_k - s_k)y_k \\ &\quad + s_k(3s_k - 2h_k)h_k M_{k+1} + (s_k - h_k)(3s_k - h_k)h_k M_k) \\ P_k''(x) &= \frac{2}{h_k^3}(3(h_k - 2s_k)y_{k+1} - 3(h_k - 2s_k)y_k + (3s_k - h_k)h_k M_{k+1} + (3s_k - 2h_k)h_k M_k) \\ P_k'''(x) &= \frac{6}{h_k^3}(2(y_{k+1} - y_k) + h_k(M_{k+1} + M_k)) \end{aligned}$$

Durch Einsetzen prüft man sofort nach:

$$\begin{aligned} P_k(x_k) &= y_k \quad\quad k = 1 \ldots N-1 \\ P_k(x_{k+1}) &= y_{k+1} \quad\quad k = 1 \ldots N-1 \\ P_k'(x_k) &= M_k \quad\quad k = 1 \ldots N-1 \\ P_k'(x_{k+1}) &= M_{k+1} \quad\quad k = 1 \ldots N-1 \end{aligned}$$

- Setzt man in die Stetigkeitsbedingung der 2. Ableitungen

$$P_k''(x_{k+1}) = P_{k+1}''(x_{k+1}) \qquad k = 1 \ldots N-2 \qquad (4)$$

die explizite Form von P_k'' ein, erhält man nach einfachem Sortieren der Terme das lineare Gleichungssystem für die M_k

$$\begin{aligned} &h_{k+1}M_k + 2(h_k + h_{k+1})M_{k+1} + h_k M_{k+2} = \\ &\quad 3\left(-\frac{h_{k+1}}{h_k}y_k + \left(\frac{h_{k+1}}{h_k} - \frac{h_k}{h_{k+1}}\right)y_{k+1} + \frac{h_k}{h_{k+1}}y_{k+2}\right) \qquad k = 1 \ldots N-2 \end{aligned}$$

- Für natürliche Splines gilt

$$\begin{aligned} P_1''(x_1) &= 0 \\ P_{N-1}''(x_N) &= 0 \end{aligned}$$

Einsetzen der expliziten Form von P_k'' liefert sofort die angegebenen Beziehungen

$$\begin{aligned} 2M_1 + M_2 &= \frac{3}{h_1}(y_2 - y_1) \\ 2M_N + M_{N-1} &= \frac{3}{h_{N-1}}(y_N - y_{N-1}) \end{aligned}$$

- Ebenso erhält man für "Not-a-Knot"-Splines aus den Bedingungen

$$\begin{aligned} P_1'''(x_2) &= P_2'''(x_2) \\ P_{N-2}'''(x_{N-1}) &= P_{N-1}'''(x_{N-1}) \end{aligned}$$

durch Einsetzen in die explizite Form von P_k''' die Gleichungen

$$
\begin{aligned}
h_2^2 M_1 + (h_2^2 - h_1^2) M_2 - h_1^2 M_3 &= 2\frac{h_1^2}{h_2}(y_2 - y_3) - 2\frac{h_2^2}{h_1}(y_1 - y_2) \\
h_{N-1}^2 M_{N-2} + (h_{N-1}^2 - h_{N-2}^2) M_{N-1} - h_{N-2}^2 M_N &= 2\frac{h_{N-2}^2}{h_{N-1}}(y_{N-1} - y_N) - 2\frac{h_{N-1}^2}{h_{N-2}}(y_{N-2} - y_{N-1})
\end{aligned}
$$

Herleitung der Normalengleichung

- Sei A eine mxn-Matrix (m > n) von maximalem Rang n, b ein m-Vektor. Dann gibt es genau einen n-Vektor x, der den quadratischen Fehler r_2 des linearen Gleichungssystem

  $A\,x = b$

  minimiert. Er ist gegeben als Lösung der Normalengleichung

  $(A^T A)\,x = A^T b$

- Beweis:
  - Statt r_2 kann man auch r_2^2 minimieren. Es gilt

    $$\begin{aligned} r_2^2 &= (Ax-b)^T(Ax-b) \\ &= (x^TA^T - b^T)(Ax-b) \\ &= x^T(A^TA)x - x^TA^Tb - b^TAx + b^Tb \end{aligned}$$

    in Komponenten

    $$\begin{aligned} r_2^2 &= \sum_{i,j} x_i(A^TA)_{ij}x_j - \sum_{i,j} x_iA_{ji}b_j - \sum_{i,j} b_iA_{ij}x_j + \sum_i b_i^2 \\ &= \sum_{i,j} x_i(A^TA)_{ij}x_j - 2\sum_{i,j} x_iA_{ji}b_j + \sum_i b_i^2 \end{aligned}$$

  - Notwendige Bedingung für ein Minimum ist, dass die Ableitung verschwindet, also

    $$\begin{aligned} 0 = \frac{\partial r_2^2}{\partial x_k} &= \sum_j (A^TA)_{kj}x_j + \sum_i x_i(A^TA)_{ik} - 2\sum_j A_{jk}b_j \\ &= 2\sum_j (A^TA)_{kj}x_j - 2\sum_j A_{jk}b_j \end{aligned}$$

    wobei der letzte Schritt aus der Symmetrie von $A^T A$ folgt:

    $$(A^TA)_{ij} = (A^TA)_{ji}$$

    Man erhält also

    $$\sum_j (A^TA)_{kj}x_j = \sum_j A_{jk}b_j$$

    bzw. in Matrix-Schreibweise

    $(A^T A)\,x = A^T b$

  - Hinreichende Bedingung für ein Minimum ist die positive Definitheit der 2. Ableitung. Nun ist

    $$\frac{\partial^2 r_2^2}{\partial x_i \partial x_k} = 2(A^TA)_{ki}$$

    Die Matrix $A^T A$ ist aber positiv definit, da sie positiv ist und maximalen Rang n hat.

Lösung der Normalengleichung

- Die Lösung von

  $R_1 x = b_1$

  ist auch eine Lösung der Normalengleichung

  $(A^T A) x = A^T b$

- Beweis:

  Mit Hilfe der Zerlegungen

  $$A = QR, \qquad R = \begin{pmatrix} R_1 \\ 0 \end{pmatrix}$$

  rechnet man nach

  $$\begin{aligned} A^T A x &= R^T Q^T Q R x \\ &= R^T R x \\ &= (R_1^T 0) \begin{pmatrix} R_1 \\ 0 \end{pmatrix} x \\ &= R_1^T R_1 x \\ &= R_1^T b_1 \\ &= (R_1^T 0) \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \\ &= R^T Q^T b \\ &= A^T b \end{aligned}$$

Aufteilungsschritt beim FFT-Verfahren

- Mit der Abkürzung

  $$\omega := e^{-2\pi i/N}$$

  kann man für gerades N die Summe in gerade und ungerade Indizes aufspalten

  $$\begin{aligned} X_k &= \sum_{n=0}^{N-1} \omega^{nk} x_n \\ &= \sum_{n\,\text{gerade}}^{N-1} \omega^{nk} x_n + \sum_{n\,\text{ungerade}}^{N-1} \omega^{nk} x_n \\ &= \sum_{n=0}^{N/2-1} \omega^{2nk} x_{2n} + \omega^k \sum_{n=0}^{N/2-1} \omega^{2nk} x_{2n+1} \end{aligned}$$

- Für halb so viele Werte ist der Faktor bei der Fouriertransformation gerade

  $$\widetilde{\omega} := e^{-2\pi i/(N/2)} = \left(e^{-2\pi i/N}\right)^2 = \omega^2$$

  also gilt für k = 0, ... N/2 - 1

  $$\begin{aligned} X_k &= \sum_{n=0}^{N/2-1} \widetilde{\omega}^{nk} x_{2n} + \omega^k \sum_{n=0}^{N/2-1} \widetilde{\omega}^{nk} x_{2n+1} \\ &= Y_k^g + \omega^k Y_k^u \end{aligned}$$

  mit den Fouriertransformationen Y^g_n der Werte x_{2n} mit geradem Index bzw. Y^u_n der Werte x_{2n+1} mit ungeradem Index.

- Für k > N/2 - 1 schreiben wir

  $$\begin{aligned} X_{k+N/2} &= \sum_{n=0}^{N/2-1} \widetilde{\omega}^{n(k+N/2)} x_{2n} + \omega^k \sum_{n=0}^{N/2-1} \widetilde{\omega}^{n(k+N/2)} x_{2n+1} \\ &= \sum_{n=0}^{N/2-1} \widetilde{\omega}^{nN/2} \widetilde{\omega}^{nk} x_{2n} + \omega^k \sum_{n=0}^{N/2-1} \widetilde{\omega}^{nN/2} \widetilde{\omega}^{nk} x_{2n+1} \end{aligned}$$

  Nun ist aber

  $$\widetilde{\omega}^{nN/2} = \omega^{nN} = \left(e^{-2\pi i/N}\right)^{Nn} e^{-2\pi i n} = 1$$

  also folgt sofort

  $$X_{k+N/2} = Y_k^g + \omega^{k+N/2} Y_k^u$$

- Man hat nun als Operationen für die Fouriertransformation von N Punkten 2 Fouriertransformationen für N/2 Punkte sowie je eine Multiplikation und Addition pro Index k, zusammen

  $$\text{ops}(N) = 2\,\text{ops}(\frac{N}{2}) + 2N$$

Zahl der Operationen beim FFT-Verfahren

- Ist N eine Zweierpotenz ist, so beträgt die Zahl der Operationen (Multiplikationen und Additionen) beim FFT-Verfahren

$$\mathrm{ops}(N) = 2N \log_2(N)$$

- Beweis:
  - Aus der Aufspaltung der Fourierreihe in je eine Fourierreihe für gerade und ungerade Indizes ergab sich die Rekursionsbeziehung

$$\mathrm{ops}(N) = 2\,\mathrm{ops}(\frac{N}{2}) + 2N$$

  - Induktionsanfang:

    Für N = 1 ist die Fourierreihe einfach

    $X_0 = x_0$

    also gilt

    $\mathrm{ops}(1) = 0 = 2*1*\log_2(1)$

  - Induktionsschritt:

    Mit der Rekursionsformel und der Induktionsvoraussetzung erhält man durch einfache Rechnung

$$\begin{aligned} \mathrm{ops}(2N) &= 2\,\mathrm{ops}(N) + 4N \\ &= 2\,(2N \log_2 N) + 4N \\ &= 4N(\log_2 N + 1) \\ &= 4N(\log_2 N + \log_2 2) \\ &= 2(2N)\log_2(2N) \end{aligned}$$

Matlab-Beispiele

- kap04.m
- plotZeros.m
- solveNewton.m
- testNewton.m
- ex01.m
- ex02.m
- ex03.m
- runde.m
- ex04.m
- ex05a.m
- bisektionsSchritt.m
- ex05b.m
- sekantenSchritt.m
- ex05c.m
- iqiSchritt.m
- ex06.m
- ex07.m
- ex08b.m
- ex08c.m
- ex09.m
- ex10.m
- ex11.m
- ex12.m
- ex13.m
- ex14.m
- ex15.m
- ex16.m
- ex17.m
- ex18.m
- ex19.m
- ex20.m
- ex21.m
- ex22.m
- odeTrapez.m

Beispieldaten

- cp.dat
- cps.dat
- stoerung.dat

Lösung von Aufgabe 1

a. Fließkommanäherung bei β = 10, t = 3, e_{min} = -3, e_{max} = 3:

$$\begin{aligned} x_1 &= 0.03125 = 3.125 \cdot 10^{-2} \\ \mathrm{fl}(x_1) &= \begin{cases} 3.12 \cdot 10^{-2} & \text{bei round to even} \\ 3.13 \cdot 10^{-2} & \text{bei Aufrunden} \end{cases} \end{aligned}$$

$$\begin{aligned} x_2 &= 10000 = 1 \cdot 10^4 \\ \mathrm{fl}(x_2) &= \infty \qquad \text{(Overflow)} \end{aligned}$$

$$\begin{aligned} x_3 &= 0.2 = 2 \cdot 10^{-1} \\ \mathrm{fl}(x_3) &= 2 \cdot 10^{-1} = x_3 \end{aligned}$$

b. Fließkommanäherung bei β = 2, t = 53, e_{min} = -1022, e_{max} = 1023:

$$x_1 = 0.03125 = \frac{1}{32} = 1 \cdot 2^{-5} = \mathrm{fl}(x_1)$$

$$\begin{aligned} x_2 &= 10000 = 2710_{16} \\ &= 10\,0111\,0001\,0000_2 = 1.0011100010000_2 \cdot 2^{13} = \mathrm{fl}(x_2) \end{aligned}$$

$$\begin{aligned} x_3 &= 0.2 = 0.\overline{0011}_2 = 1.\overline{1001}_2 \cdot 2^{-3} \\ &= 1.1001(13\mathrm{x})1001_2 \ldots \cdot 2^{-3} \end{aligned}$$

bei 53 Binärziffern (incl. der führenden 1) wird hier also aufgerundet, somit

$$\begin{aligned} \mathrm{fl}(x_3) &= 1.1001(12\mathrm{x})1010_2 \cdot 2^{-3} \\ &= \frac{1}{8}\left(1 + 9\sum_{k=1}^{12}\left(\frac{1}{16}\right)^k + 10\left(\frac{1}{16}\right)^{13}\right) \\ &= 0.200\,000\,000\,000\,000\,111\,022\ldots \end{aligned}$$

c. Zur Berechnung der IEEE-Darstellung muss man beachten, dass die 1 vor dem Punkt weggelassen wird, die Mantisse also aus den 52 Binärziffern nach dem Punkt besteht. Außerdem muss der Exponent um 1023 erhöht werden. Damit erhält man:

$$\begin{aligned} \mathrm{fl}(x_1) &= 1.0_2 \cdot 2^{-5} \\ M &= 0, \quad E = 1018 = 3fa_{16} \\ \mathrm{fl}(x_1) &\hat{=} 3fa0000000000000 \end{aligned}$$

$$\begin{aligned} \mathrm{fl}(x_2) &= 1.001110001000_2 \cdot 2^{13} \\ M &= 3880000000000, \quad E = 1036 = 40c_{16} \\ \mathrm{fl}(x_2) &\hat{=} 40c3880000000000 \end{aligned}$$

$$\begin{aligned} \mathrm{fl}(x_3) &= 1.1001(12\mathrm{x})1010_2 \cdot 2^{-3} \\ M &= 999999999999a, \quad E = 1020 = 3fc_{16} \\ \mathrm{fl}(x_3) &\hat{=} 3fc999999999999a \end{aligned}$$

Mit Hilfe von Matlabfunktionen, darunter auch solchen aus der Symbolic Toolbox, lassen sich die Rechnungen automatisch erledigen, wie das Skript ex01.m zeigt.

Lösung von Aufgabe 2

Alle Berechnungen können mit dem Matlab-Skript ex02.m ausgeführt werden.

a. Mit dem Skript erhält man für die relativen Fehler $err_{1,2}$ von exp(10) und exp(-10) folgende Werte:

N	err_1	err_2
30	7.9838e-08	2.0373e+01
40	1.7788e-13	5.3128e-05
50	3.3033e-16	7.2223e-09

Die Fehler bei exp(-10) sind dramatisch viel größer; umgekehrt braucht man viel mehr Terme, um eine vergleichbare Genauigkeit zu erhalten. Ursache ist die Auslöschung bei der Taylorreihe von exp(-10), deren Terme jedesmal das Vorzeichen wechseln.

b. Für x = 20 ergeben sich folgende Werte:

N	err_1	err_2
30	1.3475e-02	7.7612e+14
50	4.8287e-09	5.0793e+08
70	2.4571e-16	1.1302e+00
100	2.4571e-16	1.0249e+00

Die Reihe konvergiert generell langsamer, d. h. man braucht mehr Terme für einen vorgegebenen Fehler. Bei exp(-20) sind die numerischen Fehler durch Auslöschung so groß, dass auch mit beliebig vielen Termen kein sinnvolles Ergebnis erzielt wird.

c. Das Problem der Auslöschung lässt sich hier leicht umgehen, indem man exp(-x) nicht direkt über die Taylorreihe bestimmt, sondern durch exp(-x) = 1/exp(x). Die sich dabei für exp(-x) ergebenden relativen Fehler sind etwa so groß wie die von exp(x).

Lösung von Aufgabe 3

Alle Berechnungen können mit dem Matlab-Skript ex03.m ausgeführt werden. Das Runden kann sehr einfach erledigt werden mit `round(x, 4, "significant")`, allerdings wird dabei bei 0.5 immer aufgerundet. Die kleine Hilfsfunktion runde.m implementiert dagegen "round-to-even" und liefert, obwohl der Unterschied nur zweimal zum Tragen kommt, deutlich andere Werte für x.

a. LU-Zerlegung von A:

1. Vertauschung von 1. und 3. Zeile

$$P_1 = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

$$P_1 A = \begin{pmatrix} \boxed{110} & 90 & 81 \\ -29 & -59 & 10 \\ 50 & -20 & 91 \end{pmatrix}$$

2. 1. Zeile mit q_1 = -29/110 ≈ -0.2636 multiplizieren und von der 2. Zeile subtrahieren. 1. Zeile mit q_2 = 50/110 ≈ 0.4545 multiplizieren und von der 3. Zeile subtrahieren

$$\begin{pmatrix} 110 & 90 & 81 \\ 0 & -35.28 & 31.35 \\ 0 & \boxed{\text{-60.90}} & 54.19 \end{pmatrix}$$

3. Vertauschung von 2. und 3. Zeile

$$P_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

$$\rightarrow \begin{pmatrix} 110 & 90 & 81 \\ 0 & \boxed{\text{-60.90}} & 54.19 \\ 0 & -35.28 & 31.35 \end{pmatrix}$$

4. 2. Zeile mit q_3 = 35.28/60.90 ≈ 0.5793 multiplizieren und von der 3. Zeile subtrahieren

$$U = \begin{pmatrix} 110 & 90 & 81 \\ 0 & -60.90 & 54.19 \\ 0 & 0 & -0.04 \end{pmatrix}$$

5. L aus den q's zusammensetzen, dabei Vertauschung in Schritt 3 berücksichtigen

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 0.4545 & 1 & 0 \\ -0.2636 & 0.5793 & 1 \end{pmatrix}$$

6. gesamte Permutationsmatrix als Produkt der Teilpermutationen

$$P = P_2 P_1 = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

Reihenfolge klar, da immer von links an A heranmultipliziert wird

7. Kontrolle der Zerlegung

$$PA = \begin{pmatrix} 110 & 90 & 81 \\ 50 & -20 & 91 \\ -29 & -59 & 10 \end{pmatrix}$$

$$LU = \begin{pmatrix} 110 & 90 & 81 \\ 50 & -19.99 & 91 \\ -29 & -59 & 10 \end{pmatrix}$$

b. Lösung von A x = b:

Berücksichtigung von P

$$Ax = b \quad \Rightarrow \quad PAx = Pb \quad \Rightarrow \quad LUx = Pb$$

also Vorwärts-/Rückwärtssubstitution für rechte Seite

$$Pb = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 89 \\ -185 \\ 449 \end{pmatrix} = \begin{pmatrix} 449 \\ 89 \\ -185 \end{pmatrix}$$

Vorwärtssubstitution

$$Ly = Pb$$

$$\begin{pmatrix} 1 & 0 & 0 \\ 0.4545 & 1 & 0 \\ -0.2636 & 0.5793 & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 449 \\ 89 \\ -185 \end{pmatrix}$$

ergibt (bei Rundung in jedem Rechenschritt!)

$$\begin{aligned} y_1 &= 449 \\ 0.4545y_1 + y_2 &= 89 \\ \Rightarrow \quad y_2 &= -115.1 \\ -0.2636y_1 + 0.5793y_2 + y_3 &= -185 \\ \Rightarrow \quad y_3 &= 0.0800 \end{aligned}$$

Rückwärtssubstitution

$$Ux = y$$

$$\begin{pmatrix} 110 & 90 & 81 \\ 0 & -60.90 & 54.19 \\ 0 & 0 & -0.04 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 449 \\ -115.1 \\ 0.08 \end{pmatrix}$$

man erhält

$$\begin{aligned} -0.04x_3 &= 0.08 \\ \Rightarrow \quad x_3 &= -2.000 \\ -60.90x_2 + 54.19x_3 &= -115.1 \\ \Rightarrow \quad x_2 &= 0.110 \\ 110x_1 + 90x_2 + 81x_3 &= 449 \\ \Rightarrow \quad x_1 &= 5.465 \end{aligned}$$

Ergebnis der Rechnung also

$$x_* = \begin{pmatrix} 5.465 \\ 0.110 \\ -2.000 \end{pmatrix}$$

Residuum

$$|r| = |b - Ax_*| = 0.0750$$

relativer Fehler

$$|e_{\text{rel}}| = \frac{|x - x_*|}{|x|} = 0.4678$$

c. Lösung mit Matlab:

Mit

```
[L, U, P] = lu(A)
```

```
y = L \ (P*b)
xSc = U \ y
```

erhält man wesentlich genauere Werte für L, U, y und x

$$U = \begin{pmatrix} 110 & 90 & 81 \\ 0 & -60.91 & 54.18 \\ 0 & 0 & -0.0224 \end{pmatrix} \qquad L = \begin{pmatrix} 1 & 0 & 0 \\ 0.4545 & 1 & 0 \\ -0.2636 & 0.5791 & 1 \end{pmatrix}$$

$$y = \begin{pmatrix} 449.0 \\ -115. \\ 0.0224 \end{pmatrix} \qquad x = \begin{pmatrix} 4 \\ 1 \\ -1 \end{pmatrix}$$

Die kleinen Unterschiede in den letzten Zeilen zeigen, wie sich die Rechenfehler bei Rundung hier aufschaukeln.

Residuum und relativen Fehler bestimmt man mit

```
rC = norm(A*xSc - b)
```

```
eC = norm(x - xSc)/norm(x)
```

zu

rC = $2.8422 \cdot 10^{-14}$

eC = $4.4758 \cdot 10^{-13}$

Die rechte Seite der Ungleichung bestimmt man (für Teil c. bzw. b.) leicht zu

rhsC = $1.1914 \cdot 10^{-12}$

bzw.

rhsB = 3.1510

in beiden Fällen deutlich größer als der relative Fehler (um Faktor 5 - 10)

Lösung von Aufgabe 4

Alle Berechnungen können mit dem Matlab-Skript ex04.m ausgeführt werden.

a. Anfangsbeispiel:

Die Gleichungen G2 bis G6 lassen sich in die Standardform

A x = b

bringen mit

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & -a & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & a & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & a \\ 0 & 0 & 0 & 1 & a & 0 & 0 & 0 & -a \\ 0 & 0 & 0 & 0 & a & 0 & 1 & 0 & a \\ a & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ a & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$b = \begin{pmatrix} 0 & 15\,\mathrm{N} & 0 & 20\,\mathrm{N} & 0 & 0 & 0 & 0 & 0 \end{pmatrix}'$$

Mit Matlab bestimmt man sofort die Lösung für die Balkenkräfte

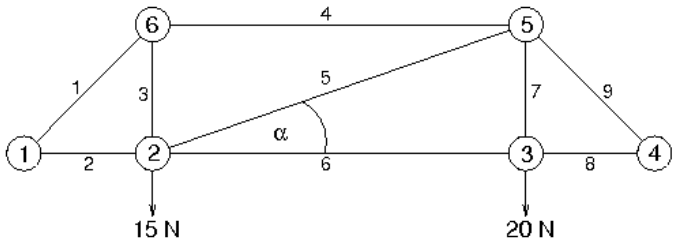
$$x_* = \begin{pmatrix} -23.5702 \\ 16.6667 \\ 16.6667 \\ -16.6667 \\ -2.3570 \\ 18.3333 \\ 20.0000 \\ 18.3333 \\ -25.9272 \end{pmatrix}$$

und die Konditionszahl

cond(A) = 6.8262

b. gestrecktes Beispiel:

Mit dem Winkel α aus dem Bild



und den Abkürzungen

$$a := \frac{1}{\sqrt{2}}$$

$$c := \cos\alpha = \frac{L}{\sqrt{L^2+1}}$$

$$s := \sin\alpha = \frac{1}{\sqrt{L^2+1}}$$

lauten die Gleichgewichtsbedingungen nun

$$
\begin{array}{lrcl}
G2 & f_2 & = & f_6 + c f_5 \\
 & f_3 + s f_5 & = & 15\,\mathrm{N} \\
 & & & \\
G3: & f_6 & = & f_8 \\
 & f_7 & = & 20\,\mathrm{N} \\
 & & & \\
G4: & f_8 + a f_9 & = & 0 \\
 & & & \\
G5: & f_4 + c f_5 & = & a f_9 \\
 & s f_5 + f_7 + a f_9 & = & 0 \\
 & & & \\
G6: & a f_1 & = & f_4 \\
 & a f_1 + f_3 & = & 0
\end{array}
$$

analog zu a. erhält man mit Matlab leicht die Balkenkräfte

$$
x_* = \begin{pmatrix}
-21.2203 \\
15.0050 \\
15.0050 \\
-15.0050 \\
-4.9900 \\
19.9950 \\
20.0000 \\
19.9950 \\
-28.2772
\end{pmatrix}
$$

Die Konditionszahl hat sich trotz der stark gestreckten Geometrie kaum geändert

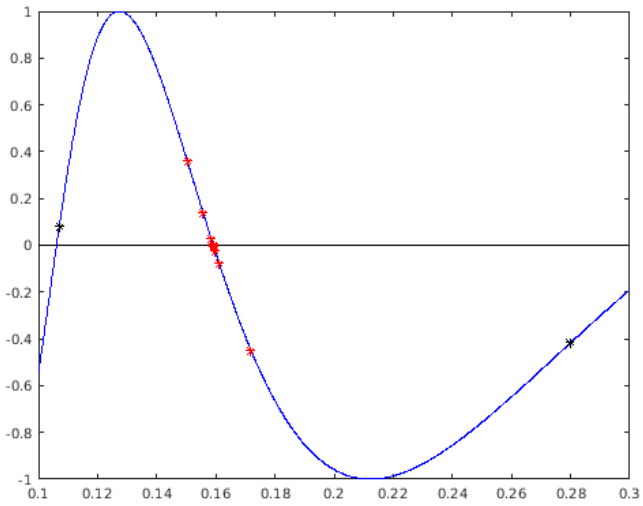
cond(A) = 8.2287

Lösung von Aufgabe 5

a. Einen einzelnen Schritt berechnet bisektionsSchritt.m. Mehrfache Ausführung bis zur gewünschten Genauigkeit liefert

```
[0.107000 0.193500]
[0.150250 0.193500]
[0.150250 0.171875]
[0.150250 0.161062]
[0.155656 0.161062]
[0.158359 0.161062]
[0.158359 0.159711]
[0.159035 0.159711]
[0.159035 0.159373]
[0.159035 0.159204]
[0.159120 0.159204]
[0.159120 0.159162]
[0.159141 0.159162]
[0.159151 0.159162]
[0.159151 0.159157]
[0.159154 0.159157]
```

im Bild



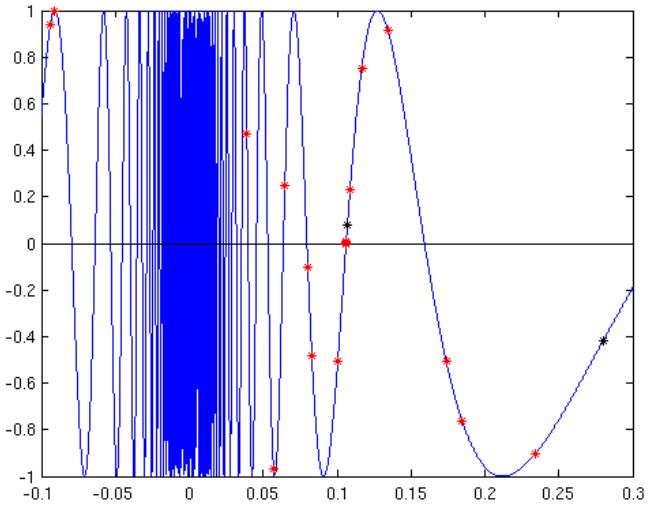
Alles zusammen erledigt ex05a.m.

b. Einen einzelnen Schritt berechnet sekantenSchritt.m. Mehrfache Ausführung bis zur gewünschten Genauigkeit liefert

```
 0.280000   0.134541
 0.134541   0.234399
 0.234399   0.184758
 0.184758  -0.091683
-0.091683   0.064695
 0.064695   0.116595
 0.116595   0.039023
 0.039023  -0.093903
-0.093903   0.173821
 0.173821   0.080220
 0.080220   0.057026
 0.057026   0.082907
 0.082907   0.108783
 0.108783   0.100439
```

```
0.100439   0.106178
0.106178   0.106104
0.106104   0.106103
```

im Bild

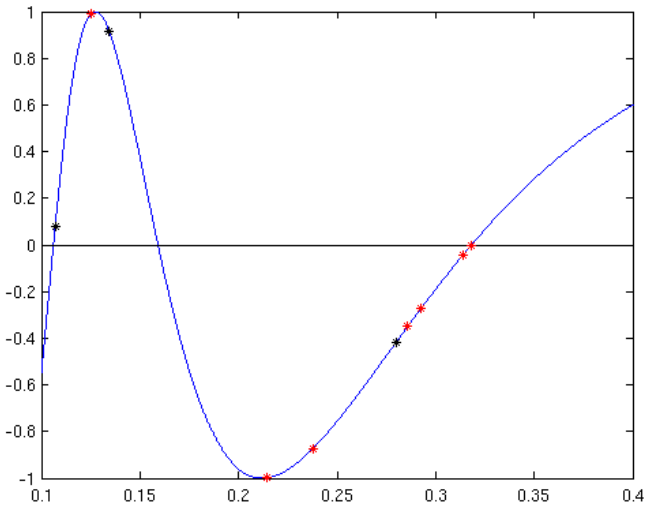


Alles zusammen erledigt ex05b.m.

c. Einen einzelnen Schritt berechnet iqiSchritt.m. Mehrfache Ausführung bis zur gewünschten Genauigkeit liefert

```
0.280000   0.134541   0.125090
0.134541   0.125090   0.237733
0.125090   0.237733   0.214633
0.237733   0.214633   0.292410
0.214633   0.292410   0.285798
0.292410   0.285798   0.313702
0.285798   0.313702   0.318215
0.313702   0.318215   0.318310
0.318215   0.318310   0.318310
```

im Bild



Alles zusammen erledigt ex05c.m.

Lösung von Aufgabe 6

Alle numerischen Berechnungen können mit dem Matlab-Skript ex06.m ausgeführt werden.

a. Kräftegleichgewicht:

Bei symmetrischer Lage der Masse heben sich die waagerechten Anteile der Federkräfte auf.

Auslenkung Δl einer Feder

$$\Delta l = \sqrt{x^2 + l^2} - l$$

entsprechende Federkraft in vertikaler Richtung

$$\begin{aligned} F_F &= -c\Delta l \sin\alpha \\ &= -c\Delta l \frac{x}{\sqrt{x^2+l^2}} \end{aligned}$$

Kräftebilanz

$$\begin{aligned} F_G + 2F_F &= mg - 2c\Delta l \frac{x}{\sqrt{x^2+l^2}} \\ &= mg - 2cx\left(1 - \frac{l}{\sqrt{x^2+l^2}}\right) \\ &= 0 \end{aligned}$$

Durch leichtes Vereinfachen erhält man

$$2\frac{c}{m}x\left(1 - \frac{l}{\sqrt{x^2+l^2}}\right) = g$$

b. Lösen der Gleichung:

Das Skript ex06.m enthält die Routinen `bisektion` und `sekante`. Damit und mit Matlabs `fzero`-Funktion erhält man jeweils das gleiche Ergebnis für die Gleichgewichtslage:

x_g = 0.502888810812037 m

c. Reduktion der Parameter

- Eine einfache Umformung ergibt

$$\begin{aligned} & 2\frac{c}{m}x\left(1 - \frac{l}{\sqrt{x^2+l^2}}\right) = g \\ \Leftrightarrow \quad & 2\frac{c}{m}x\left(1 - \frac{1}{\sqrt{\left(\frac{x}{l}\right)^2+1}}\right) = g \\ \Leftrightarrow \quad & \frac{x}{l}\left(1 - \frac{1}{\sqrt{\left(\frac{x}{l}\right)^2+1}}\right) = \frac{mg}{2cl} \end{aligned}$$

Zur Abkürzung setzen wir

$$\alpha := \frac{mg}{2cl} > 0$$

und führen eine "relative Länge" ein

$$y := \frac{x}{l}$$

Damit erhält man die reduzierte Gleichung zu

$$y\left(1 - \frac{1}{\sqrt{y^2+1}}\right) = \alpha$$

- Die Parameterwerte aus b. und c. liefern

$\alpha_b = 0.8175 = \alpha_c$

beide führen auf dieselbe reduzierte Gleichung, also auch auf das gleiche Ergebnis für y:

$y_g = 1.676296036040122$

Die Gleichgewichtslage für c. erhält man daher sofort aus

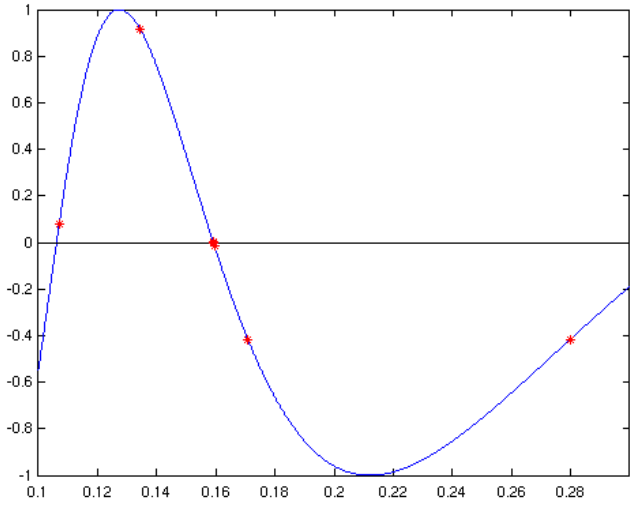
$x_g = y_g * l = 1.508666432436110$

Lösung von Aufgabe 7

- Das Matlab-Skript ex07.m vollzieht die einzelnen Schritte explizit nach (ohne Automatisierung durch Schleifen und Abfragen). Für die Berechnung eines Schrittes benutzt es die Funktionen `sekantenSchritt` und `iqiSchritt`. Die Umsortierung wird jeweils per Hand anhand der mit `zeigeWerte` ausgegebenen Funktionswerte vorgenommen:
  - a und b bilden das kleinste Intervall, das die Nullstelle umschließt (d.h. f(a) und f(b) haben unterschiedliches Vorzeichen)
  - b ist der bessere Wert (|f(b)| < |f(a)|)
  - c wird altes b
- Die Kommentare im Skript erläutern jeweils die Gründe für die Entscheidungen (welcher Schritt? wie umsortieren?). Die Ausgaben liefern die Entscheidungshilfen und dokumentieren den Fortschritt des Verfahrens:

```
f(a) =   0.078901, f(b) =  -0.416722
x:   0.280000   0.107000        NaN
f:  -0.416722   0.078901        NaN

1. Schritt
Sekantenschritt      0.134541,   0.912557
x:   0.134541   0.280000   0.107000
f:   0.912557  -0.416722   0.078901

2. Schritt
IQIschritt           0.125090,   0.990178
Bisektionsschritt    0.207270,  -0.993709
x:   0.207270   0.134541   0.280000
f:  -0.993709   0.912557  -0.416722

3. Schritt
IQIschritt           0.281375,  -0.400795
Bisektionsschritt    0.170906,  -0.418694
x:   0.134541   0.170906   0.134541
f:   0.912557  -0.418694   0.912557

4. Schritt
Sekantenschritt      0.159469,  -0.012358
x:   0.134541   0.159469   0.170906
f:   0.912557  -0.012358  -0.418694

5. Schritt
IQIschritt           0.159125,   0.001166
x:   0.159469   0.159125   0.159469
f:  -0.012358   0.001166  -0.012358

6. Schritt
Sekantenschritt      0.159155,  -0.000002
x:   0.159125   0.159155   0.159125
f:   0.001166  -0.000002   0.001166

Ergebnis:   0.159140
```

  im Bild:



- Man erkennt, dass die schnelleren Verfahren am Anfang etwas ziellos umherirren, dort hilft die Bisektion weiter. Später aber konvergieren sie sehr schnell.

Lösung von Aufgabe 8

a. Kräftebilanz:

An der Position (x, y) beträgt die Auslenkung der 1. Feder

$$\Delta L = \sqrt{(x+L)^2+y^2} - L_1$$

die Rückstellkraft ist also in horizontaler Richtung

$$\begin{aligned} F_{1,h} &= -c\Delta L \cos\alpha_1 \\ &= -c\left(\sqrt{(x+L)^2+y^2} - L_1\right)\frac{x+L}{\sqrt{(x+L)^2+y^2}} \\ &= -c\left(1 - \frac{L_1}{\sqrt{(x+L)^2+y^2}}\right)(x+L) \end{aligned}$$

in vertikaler Richtung

$$\begin{aligned} F_{1,v} &= -c\Delta L \sin\alpha_1 \\ &= -c\left(1 - \frac{L_1}{\sqrt{(x+L)^2+y^2}}\right)y \end{aligned}$$

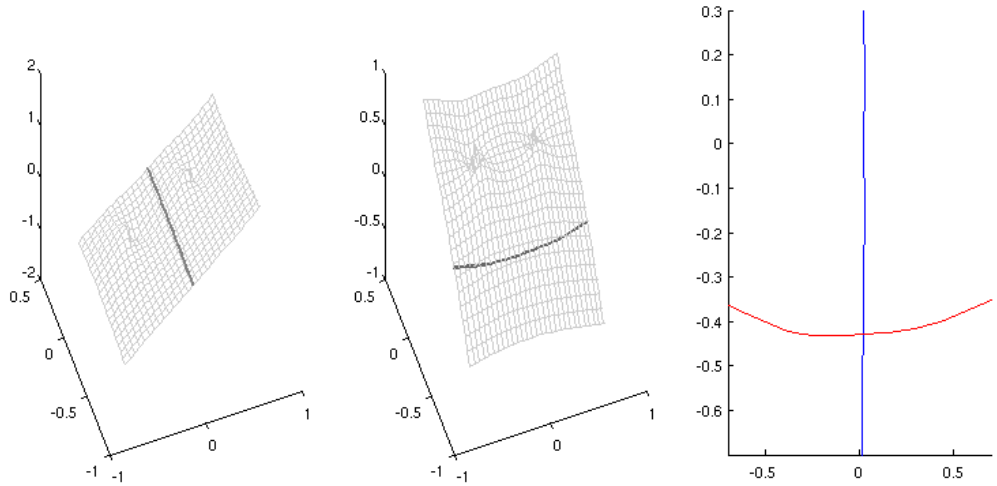
Berücksichtigt man analoge Beziehungen für die 2. Feder sowie die Gewichtskraft der Masse, erhält man sofort das angegebene System für die Kräftebilanz in horizontaler und vertikaler Richtung.

b. Bestimmen der Lösung mit dem Newton-Verfahren:

Zunächst muss die Jacobi-Matrix berechnet werden. Nach sorgfältiger Rechnung (und leichten Zusammenfassungen) erhält man

$$\begin{aligned} \frac{\partial F_1}{\partial x} &= 2 - \frac{L_1 y^2}{((x+L)^2+y^2)^{3/2}} - \frac{L_2 y^2}{((x-L)^2+y^2)^{3/2}} \\ \frac{\partial F_1}{\partial y} &= \frac{L_1(x+L)y}{((x+L)^2+y^2)^{3/2}} + \frac{L_2(x-L)y}{((x-L)^2+y^2)^{3/2}} \\ \frac{\partial F_2}{\partial x} &= \frac{\partial F_1}{\partial y} \\ \frac{\partial F_2}{\partial y} &= 2 - \frac{L_1(x+L)^2}{((x+L)^2+y^2)^{3/2}} - \frac{L_2(x-L)^2}{((x-L)^2+y^2)^{3/2}} \end{aligned}$$

Mit plotZeros verschafft man sich zunächst einen Überblick



und liest als Startwert ab:

```
x0 = [0; -0.4]
```

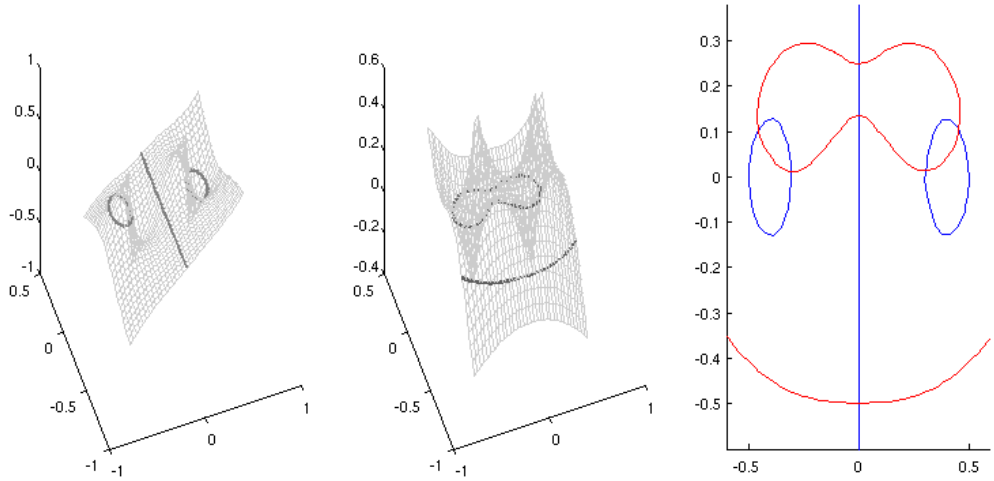
Mit Hilfe der Funktion solveNewton aus der Vorlesung erhält man sofort die Lösung

```
x = [0.0201; -0.4292]
```

Alle Berechnungen können mit dem Matlab-Skript ex08b.m ausgeführt werden.

c. Lösungen bei hoher Vorspannung:

Mit den neuen Werten erhält man als Plot



Man erkennt 3 Schnittpunkte auf der Symmetrieachse und je zwei Punkte auf der linken und rechten Seite, aufgrund der Symmetrie ($L_1 = L_2$) spiegelbildlich angeordnet.

Ablesen der Startwerte (unter Zuhilfenahme der Lupe des Matlab-Plots) liefert auf der y-Achse die Ergebnisse

```
y1 = -0.4985, y2 = 0.1346, y3 = 0.2501
```

Mit dem Startwert

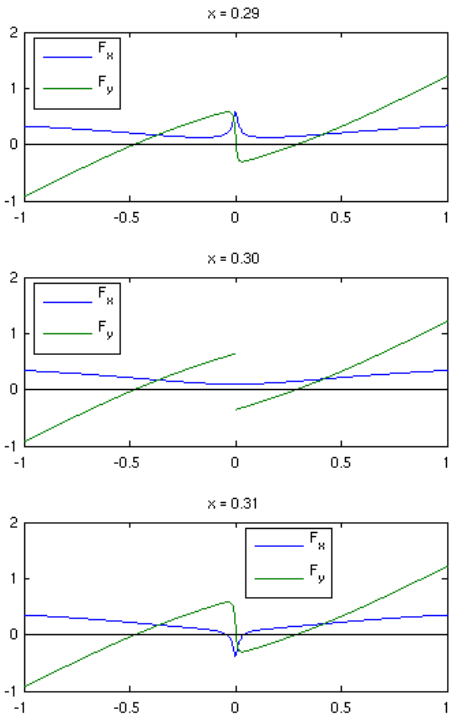
```
x04 = [0.45; 0.10]
```

bekommt man auch den oberen der seitlichen Punkte bei

```
x = [0.4543; 0.1050]
```

Der untere Punkt (etwa bei `[0.3; 0.02]`) wiedersetzt sich hartnäckig: Das Newton-Verfahren konvergiert regelmäßig gegen einen der anderen Werte, wie dicht man auch versucht, an den Wert zu kommen (etwa durch höher aufgelöste Plots). Geht man zurück in die Formel für die Kräfte, wird die Ursache klar: Die Position liegt genau auf dem Fußpunkt der 2. Feder, sie hat dann die Länge 0. Dadurch wird der Richtungsvektor undefiniert, die Funktion liefert in Matlab den Wert `NaN`.

Um zu verstehen, wieso im Plot falsche Nullstellen auftreten, untersuchen wir die kritische Stelle (0.3, 0.0) genauer und plotten drei Kurven für F_x und F_y, jeweils für x = 0.29, 0.30 und 0.31, und variables y:



Die y-Komponente der Kraft ist hier unstetig!

Physikalisch ist klar, dass hier keine Gleichgewichtsposition zu finden ist: Die Federn liegen horizontal, sie können die vertikale Gewichtskraft nicht ausgleichen. Abgesehen davon: Mit der Länge 0 hat man sicher den Linearitätsbereich jeder realen Feder verlassen!

Alle Rechnungen können mit ex08c.m nachvollzogen werden.

Lösung von Aufgabe 9

a. Interpolationspolynom:

- Für N = 5 liefert die allgemeine Formel

$$P(x) = \sum_{k=1}^{N} \left(\prod_{j \neq k} \frac{x - x_j}{x_k - x_j} \right) y_k$$

die konkrete Version

$$\begin{aligned} P(x) \quad = \quad & \frac{(x-x_2)(x-x_3)(x-x_4)(x-x_5)}{(x_1-x_2)(x_1-x_3)(x_1-x_4)(x_1-x_5)} y_1 + \frac{(x-x_1)(x-x_3)(x-x_4)(x-x_5)}{(x_2-x_1)(x_2-x_3)(x_2-x_4)(x_2-x_5)} y_2 \\ & + \frac{(x-x_1)(x-x_2)(x-x_4)(x-x_5)}{(x_3-x_1)(x_3-x_2)(x_3-x_4)(x_3-x_5)} y_3 + \frac{(x-x_1)(x-x_2)(x-x_3)(x-x_5)}{(x_4-x_1)(x_4-x_2)(x_4-x_3)(x_4-x_5)} y_4 \\ & + \frac{(x-x_1)(x-x_2)(x-x_3)(x-x_4)}{(x_5-x_1)(x_5-x_2)(x_5-x_3)(x_5-x_4)} y_5 \end{aligned}$$

Einsetzen der Werte ergibt dann (nach endlichem Rechnen)

$$P(x) = -\frac{1}{24}x^4 + \frac{11}{12}x^3 - \frac{131}{24}x^2 + \frac{139}{12}x - 6$$

- Wem die Rechnung zu lang ist (oder wer keinen symbolischen Rechner hat), der kann stattdessen auch das Gleichungssystem mit der Vandermonde-Matrix lösen:

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 & 16 \\ 1 & 3 & 9 & 27 & 81 \\ 1 & 4 & 16 & 64 & 256 \\ 1 & 5 & 25 & 125 & 625 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 1 \\ 1 \\ 4 \end{pmatrix}$$

liefert das gleiche Ergebnis (numerisch natürlich).

b. Spline-Interpolationsfunktionen:

- Gesucht sind die Polynome P_1, P_2, P_3, P_4 auf den entsprechenden vier Intervallen.
- Als erstes werden die Intervallbreiten h_i bestimmt, sie sind hier

  $h_i = 1$, i = 1 .. 4

- Die Formel für die Steigungen

$$\begin{aligned} & h_{k+1}M_k + 2(h_k + h_{k+1})M_{k+1} + h_k M_{k+2} \quad = \\ & \quad 3\left(-\frac{h_{k+1}}{h_k} y_k + \left(\frac{h_{k+1}}{h_k} - \frac{h_k}{h_{k+1}} \right) y_{k+1} \quad + \quad \frac{h_k}{h_{k+1}} y_{k+2} \right) \qquad k = 1 \ldots N-2 \end{aligned}$$

vereinfacht sich dann zu

$$M_k + 4M_{k+1} + M_{k+2} = -3y_k + 3y_{k+2} \qquad k = 1 \ldots 3$$

also konkret

$$\begin{aligned} M_1 + 4M_2 + M_3 &= 0 \\ M_2 + 4M_3 + M_4 &= -3 \\ M_3 + 4M_4 + M_5 &= 9 \end{aligned}$$

- Für natürliche Splines kommen noch die Gleichungen

$$\begin{aligned} 2M_1 + M_2 &= \frac{3}{h_1}(y_2 - y_1) \\ 2M_N + M_{N-1} &= \frac{3}{h_{N-1}}(y_N - y_{N-1}) \end{aligned}$$

dazu, konkret also

$$\begin{aligned} 2M_1 + M_2 &= 3 \\ M_4 + 2M_5 &= 9 \end{aligned}$$

- Diese 5 Gleichungen lassen sich mit Matlab schnell lösen, man erhält als Steigungen

  M_i = [1.5536, -0.1071, -1.1250, 1.6071, 3.6964]

- Dies in die Formel

$$\begin{aligned} P_k(x) &= \frac{1}{h_k^3}\left((3h_k - 2s_k)s_k^2 y_{k+1} + (h_k + 2s_k)(h_k - s_k)^2 y_k \right. \\ &\quad \left. + s_k^2(s_k - h_k)h_k M_{k+1} + s_k(s_k - h_k)^2 h_k M_k\right) \end{aligned}$$

  eingesetzt liefert

$$\begin{aligned} P_1(x) &= -0.5536x^3 + 1.6607x^2 - 0.1071x \\ P_2(x) &= 0.7679x^3 - 6.2679x^2 + 15.7500x - 10.5714 \\ P_3(x) &= 0.4821x^3 - 3.6964x^2 + 8.0357x - 2.8571 \\ P_4(x) &= -0.6964x^3 + 10.4464x^2 - 48.5357x + 72.5714 \end{aligned}$$

- Analog hat man für not-a-knot-Splines die zusätzlichen Gleichungen

$$\begin{aligned} h_2^2 M_1 + (h_2^2 - h_1^2)M_2 - h_1^2 M_3 &= 2\frac{h_1^2}{h_2}(y_2 - y_3) - 2\frac{h_2^2}{h_1}(y_1 - y_2) \\ h_{N-1}^2 M_{N-2} + (h_{N-1}^2 - h_{N-2}^2)M_{N-1} - h_{N-2}^2 M_N &= 2\frac{h_{N-2}^2}{h_{N-1}}(y_{N-1} - y_N) - 2\frac{h_{N-1}^2}{h_{N-2}}(y_{N-2} - y_{N-1}) \end{aligned}$$

  konkret

$$\begin{aligned} M_1 - M_3 &= 4 \\ M_3 - M_5 &= -6 \end{aligned}$$

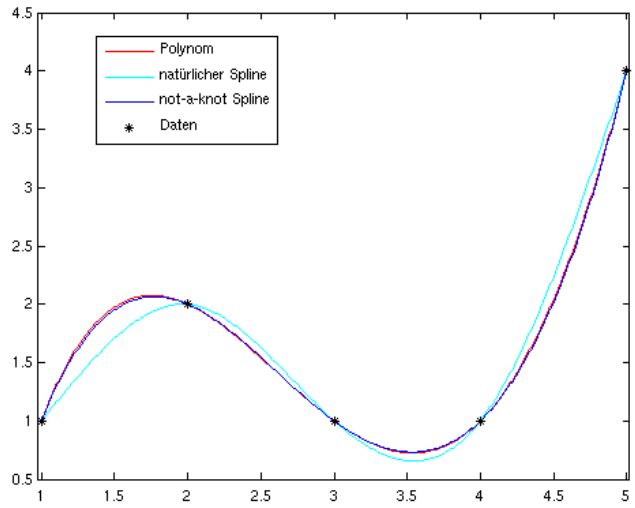
  Damit erhält man die Steigungen

  M_i = [3.0833 -0.5417 -0.9167 1.2083 5.0833]

  und die Polynome

$$\begin{aligned} P_1(x) &= 0.5417x^3 - 4.2500x^2 + 9.9583x - 5.2500 \\ P_2(x) &= P_1(x) \\ P_3(x) &= 0.2917x^3 - 2.0000x^2 + 3.2083x + 1.5000 \\ P_4(x) &= P_3(x) \end{aligned}$$

- graphische Darstellung der Interpolationsfunktionen



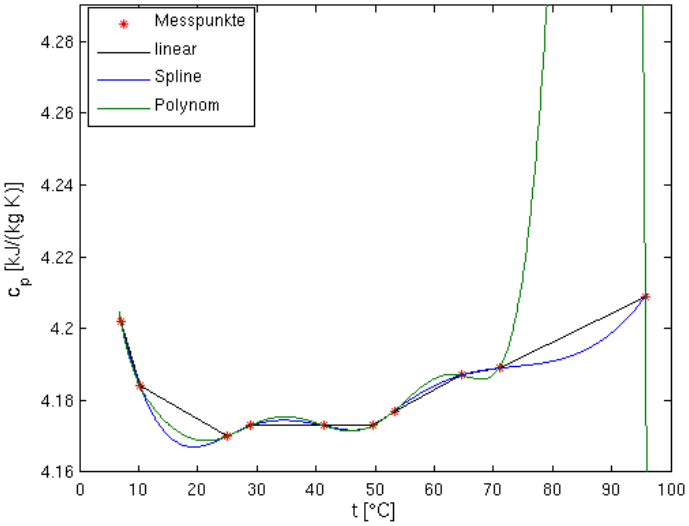
der natürliche Spline läuft an den Rändern geradlinig aus

Polynom (4. Ordnung) und not-a-knot-Spline (2x 3. Ordnung) sind kaum zu unterscheiden

Reproduktion aller Ergebnisse mit dem Matlab-Skript ex09.m

Lösung von Aufgabe 10

- Alle Berechnungen können mit dem Matlab-Skript ex10.m ausgeführt werden.
- Ergebnisse:



c_p-Wert bei 15 °C (in kJ/(kg K)):

linear	Polynom	Spline
4.1795	4.1735	4.1703

- Für den Plot braucht die lineare Interpolation nicht berechnet zu werden, da Matlabs Plotfunktion die Punkte selbst durch Strecken verbindet.
- Die Koeffizienten des Interpolationspolynoms sind sehr unterschiedlich groß, insbesondere sind die Koeffizienten der höchsten Potenzen sehr klein gegen die anderen. Matlab gibt hier eine Warnung aus, dass das Polynom schlecht konditioniert ist.

Lösung von Aufgabe 11

a. Aufstellen des Gleichungssystems:

Die Systemmatrix ist die Vandermonde-Matrix

$$A = \begin{pmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ 1 & x_3 & x_3^2 \\ 1 & x_4 & x_4^2 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 3 & 9 \\ 1 & 4 & 16 \end{pmatrix}$$

die rechte Seite ist einfach

$$b = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix} = \begin{pmatrix} 3 \\ 2 \\ 3 \\ 7 \end{pmatrix}$$

b. QR-Zerlegung:

1. Schritt

$$s_1 = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

$$v_1 = s_1 - |s_1|e_1 = \begin{pmatrix} -1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

$$Q_1 = 1 - \frac{2}{v_1^T v_1}(v_1 v_1^T) = \frac{1}{2}\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix}$$

$$A_1 = Q_1 A = \begin{pmatrix} 2 & 5 & 15 \\ 0 & -2 & -10 \\ 0 & -1 & -5 \\ 0 & 0 & 2 \end{pmatrix}$$

2. Schritt

$$s_2 = \begin{pmatrix} 0 \\ -2 \\ -1 \\ 0 \end{pmatrix}$$

$$v_2 = s_2 - |s_2|e_2 = \begin{pmatrix} 0 \\ -4.2361 \\ -1 \\ 0 \end{pmatrix}$$

$$Q_2 = 1 - \frac{2}{v_2^T v_2}(v_2 v_2^T) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -0.8944 & -0.4472 & 0 \\ 0 & -0.4472 & 0.8944 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$A_2 = Q_2 A_1 = \begin{pmatrix} 2 & 5 & 15 \\ 0 & 2.2361 & 11.1803 \\ 0 & 0 & 0 \\ 0 & 0 & 2 \end{pmatrix}$$

3. Schritt

$$
\begin{aligned}
s_3 &= \begin{pmatrix} 0 \\ 0 \\ 0 \\ 2 \end{pmatrix} \\
v_3 &= s_3 - |s_3| e_3 = \begin{pmatrix} 0 \\ 0 \\ -2 \\ 2 \end{pmatrix} \\
Q_3 &= 1 - \frac{2}{v_3^T v_3}(v_3 v_3^T) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \\
A_3 &= Q_3 A_2 = \begin{pmatrix} 2 & 5 & 15 \\ 0 & 2.2361 & 11.1803 \\ 0 & 0 & 2 \\ 0 & 0 & 0 \end{pmatrix} = R \\
Q &= Q_1^T Q_2^T Q_3^T = \begin{pmatrix} 0.5 & -0.6708 & 0.5 & 0.2236 \\ 0.5 & -0.2236 & -0.5 & -0.6708 \\ 0.5 & 0.2236 & -0.5 & 0.6708 \\ 0.5 & 0.6708 & 0.5 & -0.2236 \end{pmatrix}
\end{aligned}
$$

c. Lösen der Normalengleichung

rechte Seite

$$
Q^T b = \begin{pmatrix} 7.5 \\ 2.9069 \\ 2.5 \\ -0.2236 \end{pmatrix}
$$

zu lösendes System also

$$
R_1 x = \begin{pmatrix} 2 & 5 & 15 \\ 0 & 2.2361 & 11.1803 \\ 0 & 0 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 7.5 \\ 2.9069 \\ 2.5 \end{pmatrix}
$$

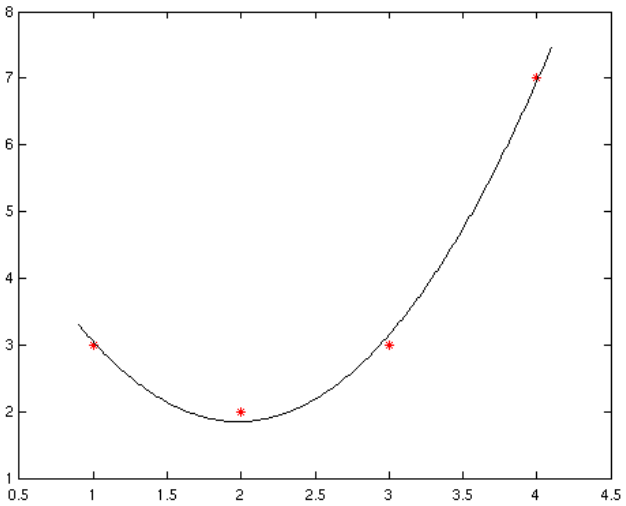
Rückwärtssubstitution liefert

$$
x = \begin{pmatrix} 6.75 \\ -4.95 \\ 1.25 \end{pmatrix}
$$

Das Ausgleichspolynom ist daher

$$
P(x) = 1.25x^2 - 4.95x + 6.75
$$

- graphische Darstellung der Ausgleichskurve



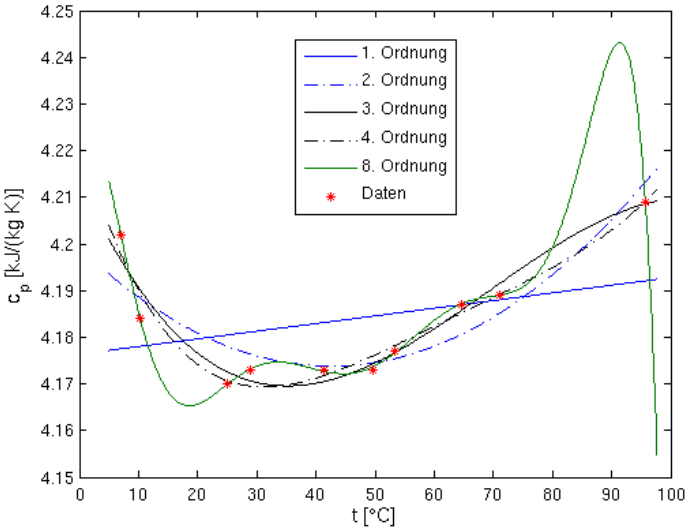
Reproduktion aller Ergebnisse mit dem Matlab-Skript ex11.m

Lösung von Aufgabe 12

Alle numerischen Berechnungen können mit dem Matlab-Skript ex12.m ausgeführt werden.

a. Ausgleichskurven:

Ergebnis



Bewertung der Kurven

- Die 8. Ordnung schwingt zu stark.
- 1. und 2. Ordnung geben die generelle Tendenz nicht wieder
- Tendenz der 3. Ordnung bei t > 70 °C sieht falsch aus
- 4. Ordnung passt gut und schwingt nicht - optimal.

b. Ausgleichspolynom für gewichtete Daten:

Da Matlabs `polyfit`-Funktion nicht mit gewichteten Daten umgehen kann, muss das Ausgleichsverfahren durchgeführt werden.

Matrix und rechte Seite des linearen Gleichungssystems erhält man aus

$$A_{ki} = \frac{x_k^i}{\sigma_k} \qquad k = 1 \ldots m, i = 0 \ldots n$$

$$b_k = \frac{y_k}{\sigma_k} \qquad k = 1 \ldots m$$

Hat man in Matlab die Werte in Spaltenvektoren t, cp und sigma gespeichert, kann man die Systemgrößen berechnen durch

```
sigma4Mat = sigma*ones(1, 5);
A4 = [ones(length(t),1), t, t.^2, t.^3, t.^4]./sigma4Mat;
b = cp./sigma;
```

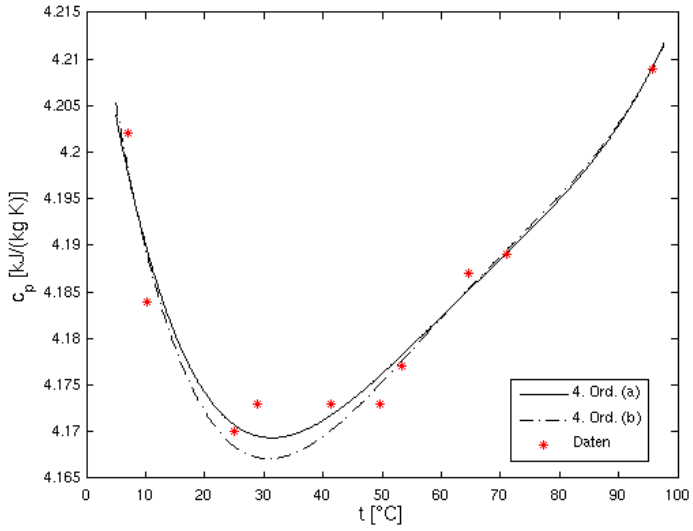
Die Lösung des Ausgleichssystems erhält man mit

```
a4 = A4 \ b;
```

Für ein Matlab-Polynom muss man die Reihenfolge umdrehen (erster Koeffizient in Matlab = höchste Potenz)

```
poly4b = a4(5:-1:1);
```

Mit `polyval` erhält man somit die Werte für den Plot



Vergleich a/b

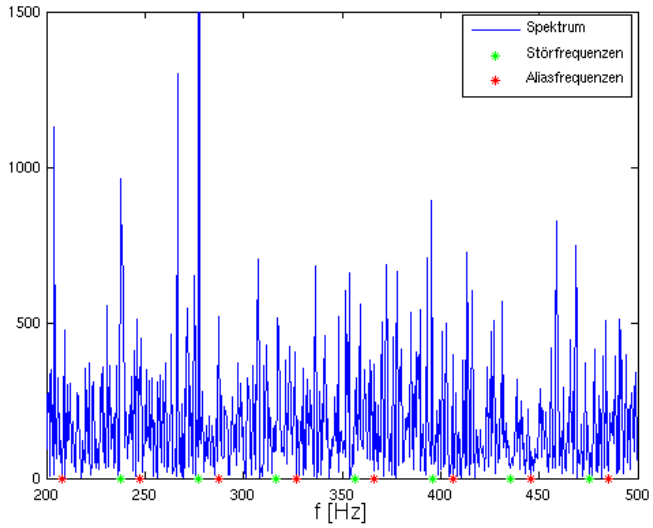
Die Werte bei 30 °C und 40 °C haben einen deutlich größeren Messfehler, sie werden als "Ausreißer" weniger stark berücksichtigt. Die Ausgleichskurve verläuft daher tiefer, dichter an den anderen Punkten.

Lösung von Aufgabe 13

- Alle Berechnungen können mit dem Matlab-Skript ex13.m ausgeführt werden.
- Die Messungen erfolgen im Zeitabstand Δt = 1 ms, die Nyquist-Frequenz ist also

$$f_N = \frac{1}{2\Delta t} = 500\,\mathrm{Hz}$$

- Relevante Frequenzanteile außer Rauschen scheinen ab etwa 300 Hz nicht mehr aufzutreten. Zur genaueren Kontrolle wird der obere Frequenzbereich zusammen mit den Störfrequenzen und den Kandidaten für Aliasfrequenzen vergrößert dargestellt



  Höchstens bei der möglichen Aliasfrequenz 287 Hz ist eine Spitze zu erkennen, die aber genauso gut auch vom Rauschen herrühren kann.

- Wie sich das Aliasing der höheren Rauschfrequenzen auswirkt, ist nicht unmittelbar klar, aber vermutlich sind die "gespiegelten" Anteile in den allgemeinen "Rauschteppich" eingegangen.

Lösung von Aufgabe 14

a. Berechnung der Ortskurve:

- Der Cosinussatz für das Dreieck OPQ liefert

$$l_2^2 = l_1^2 + x^2 - 2 l_1 x \cos\varphi$$

Nach x auflösen →

$$\begin{aligned} x &= l_1 \cos\varphi \pm \sqrt{l_1^2 \cos^2\varphi - l_1^2 + l_2^2} \\ &= l_1 \cos\varphi \pm \sqrt{l_2^2 - l_1^2 \sin^2\varphi} \end{aligned}$$

Negatives Vorzeichen entfällt wegen $l_2 > l_1$, $x > 0$. Einführen von λ ergibt

$$x = l_2(\lambda \cos\varphi + \sqrt{1 - \lambda^2 \sin^2\varphi})$$

Gleichmäßige Umdrehung heisst

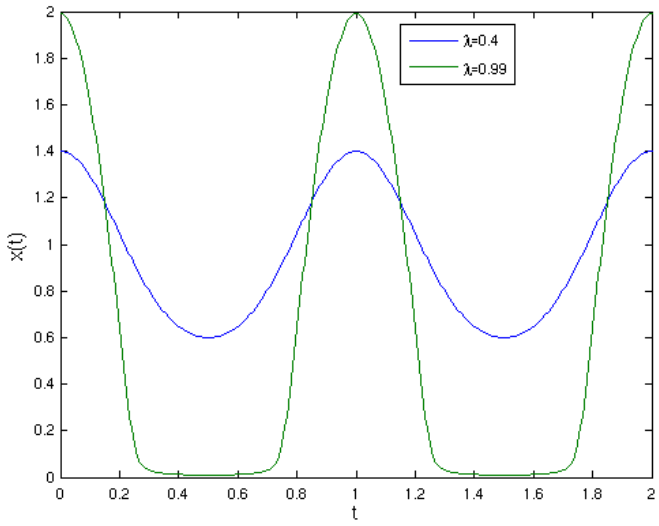
$\varphi = \omega\, t$

also

$$x(t) = l_2 \left(\lambda \cos(\omega t) + \sqrt{1 - \lambda^2 \sin^2(\omega t)}\right)$$

- Die Parameter l_2 und ω spielen für die Form von Ortskurve und Spektrum keine entscheidende Rolle:
  - l_2 skaliert (bei festem Schubstangenverhältnis λ) lediglich x(t), ist also für alle Fourierkoeffizienten ein gemeinsamer Faktor;
  - ω tritt nur als Faktor von t auf, gibt also die Grundfrequenz vor und streckt das Spektrum, ohne seine Form zu beeinflussen.

  Im Weiteren werden sie fest gewählt zu
  - $l_2 = 1$
  - $\omega = 2\pi$ (also T = 1)
- Plot

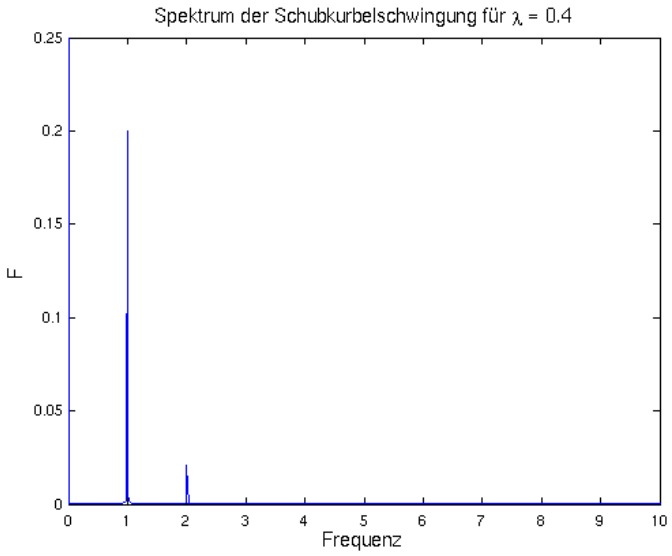


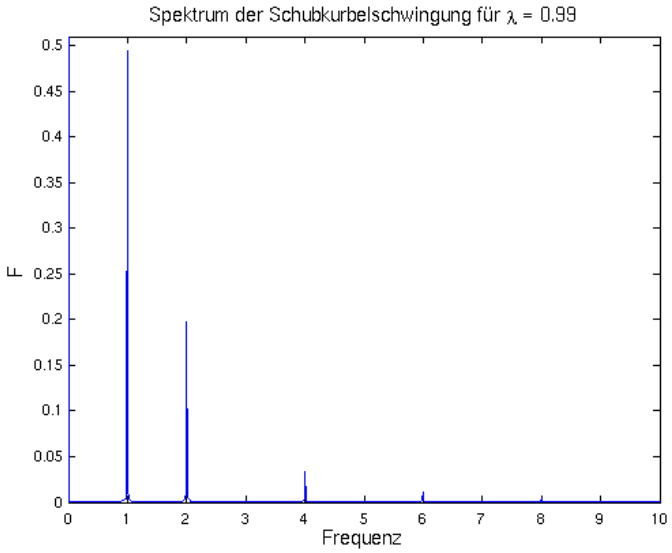
b. Bestimmung des Spektrums:

- Zunächst werde mit großen Werten für T und N gerechnet, um eine möglichst genaue Darstellung zu erhalten:
  - T = 100;
  - N = 4096;

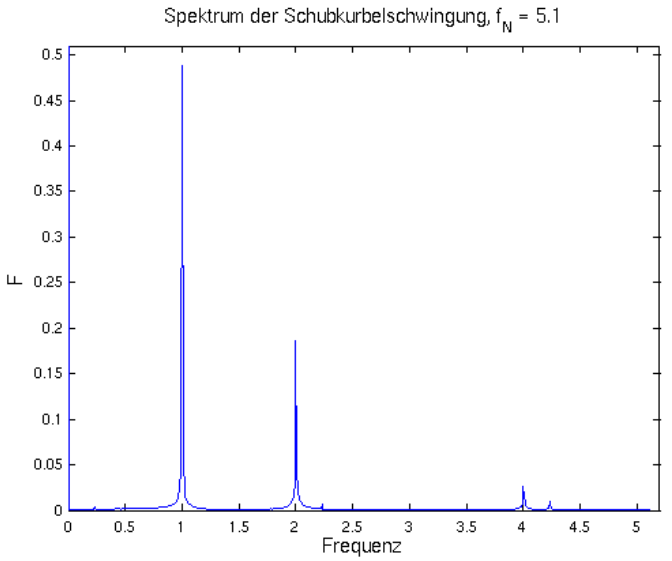
Die zugehörige Nyquistfrequenz ist $f_N = 20.475$

Um den interessanten Teil des Plots hervorzuheben, wird nur bis zur Frequenz f = 10 geplottet und die Skalierung von F so gewählt, dass der höchste Peak nach dem (alles überragenden) Mittelwert F(0) gut zu sehen ist. Man erhält

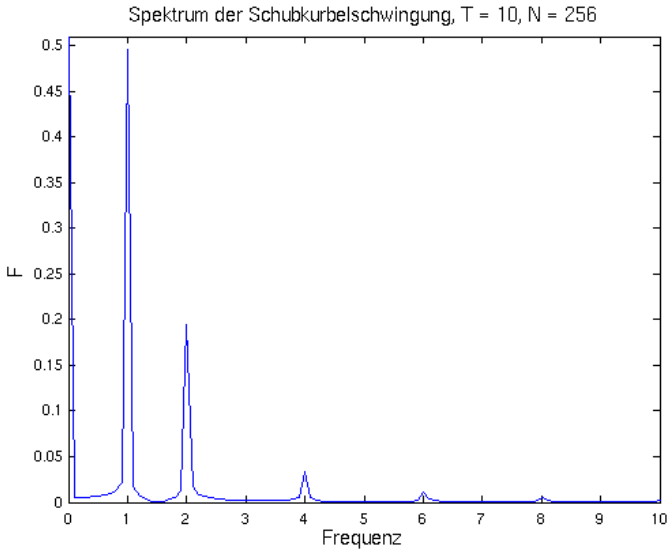




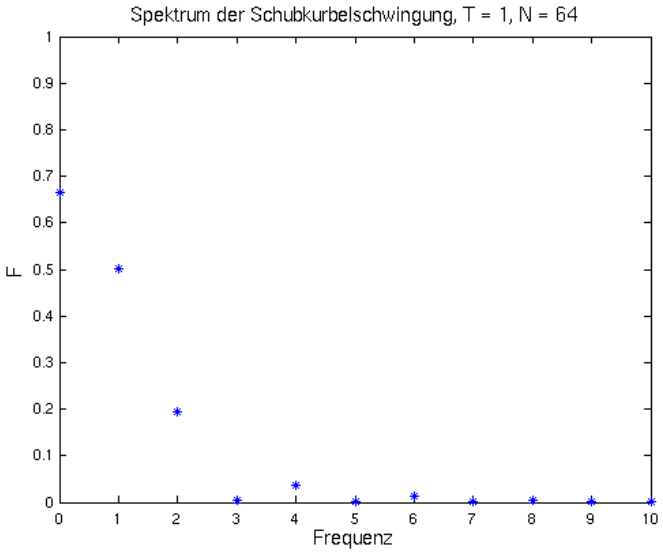
- Verkleinert man N auf 1024, ist die Nyquist-Frequenz nur noch $f_N = 5.115$, die Oberwelle bei f = 6 ist jetzt als Peak bei 2 f_N - f = 4.230 zu sehen (Aliasing).



- Verkleinert man T, verringert man damit die Frequenzauflösung, so dass sich die Peaks verbreitern. Ein großes N nützt hier nichts, sondern bewirkt aufgrund des großen f_N-Werts nur, dass auch (hier nicht vorkommende) sehr hohe Frequenzanteile noch aufgelöst werden.



Der kleinstmögliche Wert für T ist natürlich T = 1, eine Schwingung.



- Reproduktion aller Ergebnisse mit dem Matlab-Skript ex14.m

Lösung von Aufgabe 15

- Alle Berechnungen können mit dem Matlab-Skript ex15.m ausgeführt werden.
- Die Hilfsroutine `diagonalfehler` berechnet den Fehler δ; sie ist hier knapp über eine anonyme Funktion definiert.
- Das QR-Verfahren braucht 11 Schritte, mit vorheriger Transformation in Hessenbergform nur 6 Schritte. Das Verfahren mit Shift benötigt 21 Schritte! Verfolgt man hier die Zwischenergebnisse für D, stellt man fest, dass die untere Zeile und rechte Spalte sehr schnell (nach drei Schritten) diagonal werden. Anschließend geht es aber nur langsam voran. Dieses Verhalten legt nahe, dass man das Verfahren verbessern kann, indem man einen anderen Diagonalwert für σ wählt, sobald eine Zeile/Spalte nahezu diagonal wird.
- Die Reihenfolge der Diagonalelemente ist bei jedem Verfahren verschieden, demzufolge sind auch die Spalten von U (Eigenvektoren) entsprechend vertauscht. Außerdem können einzelne Eigenvektoren noch einen zusätzlichen Faktor -1 haben.

Lösung von Aufgabe 16

- Alle Berechnungen und Zeichnungen können mit dem Matlab-Skript ex16.m ausgeführt werden.
- Die Massen- und die Steifigkeitsmatrix definieren das verallgemeinerte Eigenwertproblem, das in Matlab mit

  ```
  [U, D] = eig(C,M);
  ```

  sofort gelöst werden kann.
- Die Eigenwerte d_i auf der Diagonalen von D sind die Werte ω_i^2, die Eigenfrequenzen f_i erhält man dann als

  $$f_i = \frac{\sqrt{d_i}}{2\pi}$$

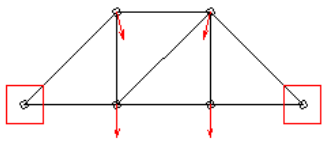
  Für gewöhnlich sortiert man die Eigenwerte der Größe nach, bei Schwingungsproblemen beginnend mit dem kleinsten. Die ersten (niederfrequenten) Schwingungen lassen sich in der Regel leichter anregen und sind daher in der Praxis am bedeutendsten.
- Die Eigenvektoren sind jeweils die Spalten von U, der 1. Eigenvektor (zur Eigenfrequenz $f_1 = 0.0670$ Hz) ist also

  $$\hat{x} = (0, -0.5410, 0, -0.5410, -0.0958, -0.4452, 0.0958, -0.4452)^T$$

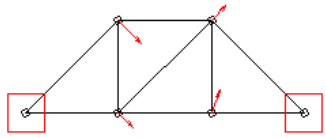
  Gemäß der Definition der Komponenten von $\hat{x}$ bedeutet dies folgende Verschiebungsvektoren für die vier Knoten:

  $$x_1 = \begin{pmatrix} 0 \\ -0.5410 \end{pmatrix} \quad x_2 = \begin{pmatrix} 0 \\ -0.5410 \end{pmatrix} \quad x_3 = \begin{pmatrix} -0.0958 \\ -0.4452 \end{pmatrix} \quad x_4 = \begin{pmatrix} 0.0958 \\ -0.4452 \end{pmatrix}$$

- Zeichnet man diese Vektoren direkt an die Knoten, erhält man eine gute Vorstellung von der Form der 1. Eigenschwingung

  bzw. analog von der 2. Eigenschwingung zur Frequenz $f_2 = 0.0984$ Hz

- Statt selbst zu zeichnen, kann man das auch Matlab erledigen lassen. Dies erledigt hier die Routine `plotMode(xe)` in folgender Weise
  - `x0` enthält die x- und y-Koordinaten der 6 Punkte (1 - 4 beweglich, 5 und 6 fest).
  - `A` ist eine symmetrische 6x6-Matrix mit Einträgen 0 und 1. Dabei bedeutet $a_{ij} = 1$, dass die Punkte i und j durch eine Feder verbunden sind.
  - Die Massen werden einfach als kleine Kreise geplottet.
  - In einer Schleife über die Elemente von A werden die Federn als Verbindungsstrecken der Punkte gezeichnet.
  - Die Vektoren werden mit der Funktion `quiver` erzeugt. Diese enthält für jeden Vektor x- und y-

Koordinate des Angriffspunktes (aus `x0`) und x- und y-Koordinate des Vektors selbst (in `xe`).

Lösung von Aufgabe 17

- Die Aufgabe lässt sich mit etwas Handarbeit erledigen, indem man den Integranden und die Berechnung der Simpson-Näherungen I_1 und I_2 jeweils als Matlab-Funktionen definiert und per Hand iteriert.

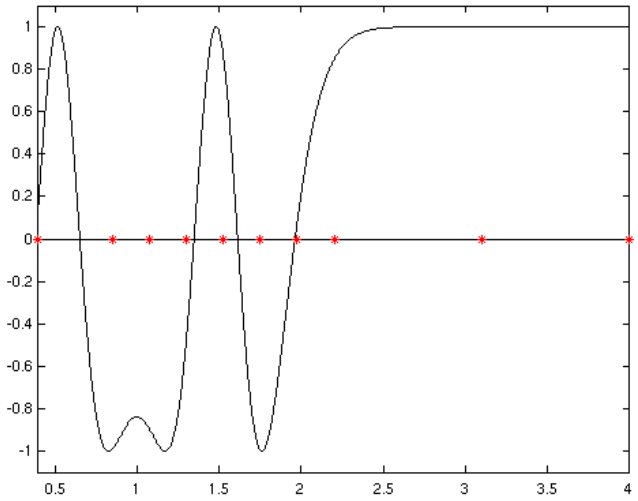
  Man erhält folgende Schritte:

```
    a       b        I1       I2
=============================================
 0.400   4.000    2.7244   1.4803   halbieren
 0.400   2.200   -0.2736  -1.1846   halbieren
 0.400   1.300   -0.6431  -0.3510   halbieren
 0.400   0.850    0.0275   0.0273   ok
 0.850   1.300   -0.3786  -0.3995   halbieren
 0.850   1.075   -0.1988  -0.1996   ok
 1.075   1.300   -0.2007  -0.2001   ok
 1.300   2.200   -0.5415   0.1617   halbieren
 1.300   1.750    0.1498   0.0699   halbieren
 1.300   1.525    0.1156   0.1137   ok
 1.525   1.750   -0.0457  -0.0435   ok
 1.750   2.200    0.0120  -0.0064   halbieren
 1.750   1.975   -0.1296  -0.1302   ok
 1.975   2.200    0.1232   0.1230   ok
 2.200   4.000    1.7540   1.7764   halbieren
 2.200   3.100    0.8764   0.8839   ok
 3.100   4.000    0.9000   0.9000   ok
```

  benutzte x-Werte sind also

  0.400 0.850 1.075 1.300 1.525 1.750 1.975 2.200 3.100 4.000

  im Bild



  Für das Integral ergibt sich damit

  $I \approx 1.4750$ (genauer Wert: 1.4738)

- Mit etwas mehr Matlab-Programmierung lässt sich das Problem sehr schnell und elegant durch Rekursion lösen:
  - Funktion `simpson_adapt` berechnet die beiden Integrale (mit einem Schritt bzw. zwei halben Schritten).
  - Ist die Genauigkeit erreicht, gibt es das (extrapolierte) Ergebnis zurück.
  - Sonst teilt es das Intervall in zwei Hälften, ruft sich selbst zur Berechnung der Teilstücke auf und

addiert die beiden Teilergebnisse.

Als Matlab-Code:

```
function I = simpson_adapt(f, a, b, tol)
% berechnet das Integral mit adaptivem Simpsonverfahren

I1 = (b-a)/6 * (f(a) + 4*f((a+b)/2) + f(b));
I2 = (b-a)/12 * (f(a) + 4*f((3*a+b)/4) + 2*f((a+b)/2) ...
                  + 4*f((a+3*b)/4) + f(b));

if  abs(I1 - I2) < tol
  I = (16*I2 - I1)/15;
else
  c = (a+b)/2;
  Il = simpson_adapt(f, a, c, tol);
  Ir = simpson_adapt(f, c, b, tol);
  I = Il + Ir;
end
```

Achtung: Bei numerischen Problemen mit dem Integral (etwa einem Pol) kann die Funktion in eine Endlosschleife geraten!

Mit einem solchen rekursiven Verfahren lassen sich auch die Endpunkte bequem sammeln sowie alle Zwischenergebnisse ausgeben.

- Das Matlab-Skript ex17.m zeigt die komplette Lösung.

Lösung von Aufgabe 18

- Alle Berechnungen können mit dem Matlab-Skript ex18.m ausgeführt werden.

a. Die gesuchten Werte erhält man sofort, wenn man die Flächen unter den Kurven 1 → 2 und 3 → 4 kennt. Aus der Adiabatengleichung erhält man die zugehörige Funktion

$$P(V) = \frac{\text{const.}}{V^\kappa}$$

Die Konstante erhält man, indem man einen bekannten Wert einsetzt, der auf der Kurve liegt, also etwa (V_1, p_1) oder (V_2, p_2) für die Kurve 1 → 2. Beide Werte müssen die gleiche Konstante liefern! Damit kann man die beiden Kurven leicht als Matlab-Funktionen definieren und mit `quad` integrieren.

b. Die Kurve 3 → 4 wird durch die angegebene Funktion ersetzt. Dabei ist zu beachten, dass man in p(V) die Hilfsfunktion y(x) als Funktion von V schreiben muss, also y(x(V)). Mit der notwendigen Sorgfalt beim Einsatz von punktweisen Operationen ist p(V) damit schnell hingeschrieben.

Lösung von Aufgabe 19

- Zunächst muss die Differentialgleichung wie üblich in die Grundform gebracht werden, um die Funktion f zu erhalten:

$$\begin{aligned}\vec{y} &= \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} := \begin{pmatrix} x \\ \dot{x} \end{pmatrix} \\ \dot{\vec{y}} &= \begin{pmatrix} \dot{x} \\ \ddot{x} \end{pmatrix} = \begin{pmatrix} y_2 \\ -g - \frac{b}{m} y_2^2 \operatorname{sign}(y_2) \end{pmatrix} =: \vec{f}(t, \vec{y}(t))\end{aligned}$$

- Als Formeln ausgeschrieben liefert das Butcher-Diagramm:

$$\begin{aligned} k_1 &= f(t, y) \\ k_2 &= f(t + \frac{1}{3}h, y + \frac{1}{3}hk_1) \\ k_3 &= f(t + \frac{2}{3}h, y + (-\frac{1}{3}k_1 + k_2)h) \\ k_4 &= f(t + h, y + (k_1 - k_2 + k_3)h) \\ y(t+h) &= y + \frac{1}{8}h(k_1 + 3k_2 + 3k_3 + k_4) \end{aligned}$$

- Alle Berechnungen können nun mit dem Matlab-Skript ex19.m ausgeführt werden.
- Der Vergleich mit Matlabs ode45-Ergebnissen liefert:

t	x(Hand)	x(Matlab)
0	5000.00	5000.00
10	4588.74	4589.63
20	3780.78	3796.62
30	2948.64	2945.69
40	2104.34	2088.65
50	1253.51	1230.97
60	399.21	373.21
70	-456.85	-484.54

Lösung von Aufgabe 20

Alle Berechnungen können mit dem Matlab-Skript ex20.m ausgeführt werden.

a. ohne Tilger:

Im Fall ohne Tilger vereinfacht sich die Bewegungsgleichung zu

$$m_1\ddot{x} + b_1\dot{x} + c_1 x = F_1 \cos \Omega t$$

Die entsprechende Systemfunktion incl. aller Parameter ist schnell hingeschrieben:

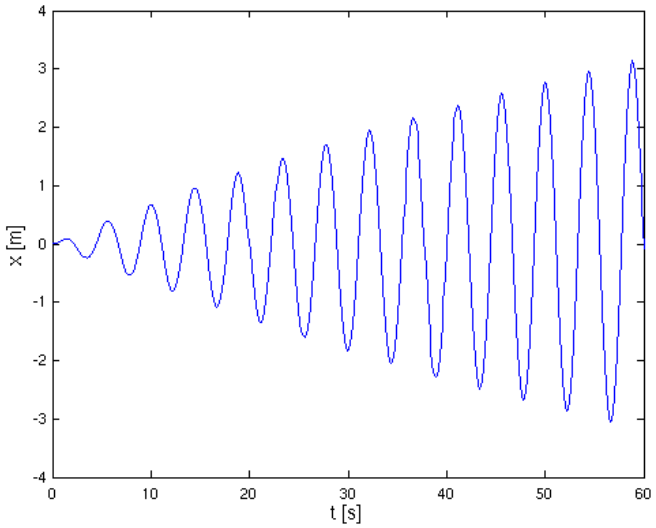
```
function dydt = f1d(t, y, m, b, c, F, Om)
% rechte Seite der DGL bei erzwungener Schwingung ohne Tilger
x = y(1);
v = y(2);
dydt = [v; (F*cos(Om*t) - b*v - c*x)/m];
```

Für den Solver muss eine Hilfsfunktion eingeführt werden, die nur von t und y abhängt:

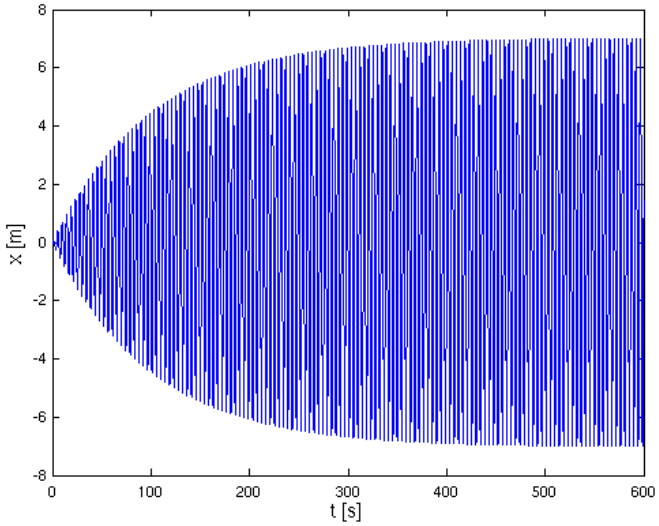
```
f1 = @(t,y) f1d(t,y,m1,b1,c1, F1, Om);
```

Dann kann mit dem Standardsolver `ode45` das Problem sofort gelöst werden:

```
[t, y] = ode45(f1, [0 60], [0 0]);
x = y(:,1); % Auslenkung
v = y(:,2); % Geschwindigkeit
plot(t, x);
xlabel("t [s]","FontSize",12);
ylabel("x [m]","FontSize",12);
```



Der Plot zeigt, dass sich die Schwingung der Masse noch aufschaukelt, man muss für die Dauerschwingung also länger simulieren. Für t_1 = 600 s erhält man



Aus dem Bild kann man die erreichte Amplitude ungefähr ablesen. Genauer geht es, indem man die Daten aus der Simulation direkt verwendet. Dazu werden zunächst die Werte nach t = 500 s herausgefiltert, dann unter diesen das Maximum herausgesucht:

```
index = find(t >= 500);
xSpaet = x(index);
amplitude1d = max(abs(xSpaet))
```

Man erhält eine Amplitude von 7.0085 m. Bei genauerer Betrachtung stellt man übrigens fest, dass die Amplitude immer noch - wenn auch nur leicht - ansteigt!

b. mit Tilger:

Im Fall mit Tilger hat man zwei Gleichungen jeweils zweiter Ordnung, man braucht also einen Vektor mit insgesamt 4 Zustandsgrößen. Am einfachsten definiert man

$$\vec{y} = \begin{pmatrix} x_1 \\ x_2 \\ \dot{x_1} \\ \dot{x_2} \end{pmatrix}$$

Schreibt man die Matrix-Vektor-Multiplikationen komponentenweise aus, kann man die Systemfunktion leicht hinschreiben:

$$\vec{f}(t,\vec{y}) = \begin{pmatrix} y_3 \\ y_4 \\ \left(F_1 \cos \Omega t - (b_1+b_2)y_3 + b_2 y_4 - (c_1+c_2)y_1 + c_2 y_2\right)/m_1 \\ \left(b_2 y_3 - b_2 y_4 + c_2 y_1 - c_2 y_2\right)/m_2 \end{pmatrix}$$

Eleganter und - spätestens bei größeren Systemen - übersichtlicher wird es, wenn man y in Zweiervektoren zerlegt und die Matrizen stehen lässt:

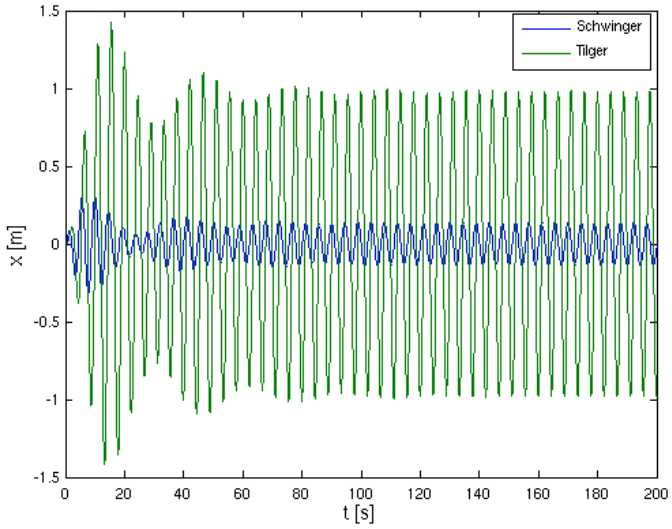
$$\begin{aligned} \vec{x} &= \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \\ \vec{v} &= \begin{pmatrix} \dot{x_1} \\ \dot{x_2} \end{pmatrix} = \begin{pmatrix} y_3 \\ y_4 \end{pmatrix} \\ \vec{f}(t,\vec{y}) &= \begin{pmatrix} \vec{v} \\ \vec{M}^{-1}(\vec{F}(t) - \vec{B}\vec{v} - \vec{C}\vec{x}) \end{pmatrix} \end{aligned}$$

In Matlab lautet die entsprechende Funktion dann einfach

```
function dydt = f2d(t, y, M, B, C, Fhat, Om)
% rechte Seite der DGL bei erzwungener Schwingung mit Tilger
x = y(1:2);
v = y(3:4);
```

```
dx = v;
dv = inv(M)*(Fhat*cos(Om*t) - B*v - C*x);
dydt = [dx; dv];
```

Das kann wieder leicht mit `ode45` integriert werden, man erhält



Spätestens ab t = 100 s ist die Dauerschwingung erreicht. Wie oben erhält man eine Amplitude von 0.1413 m für den Schwinger und von 0.9942 m für den Tilger. Das Maximum über *alle* Tilgerauslenkungen zeigt, dass er in der Einschwingphase bis auf 1.4287 m ausgelenkt wird.

Lösung von Aufgabe 21

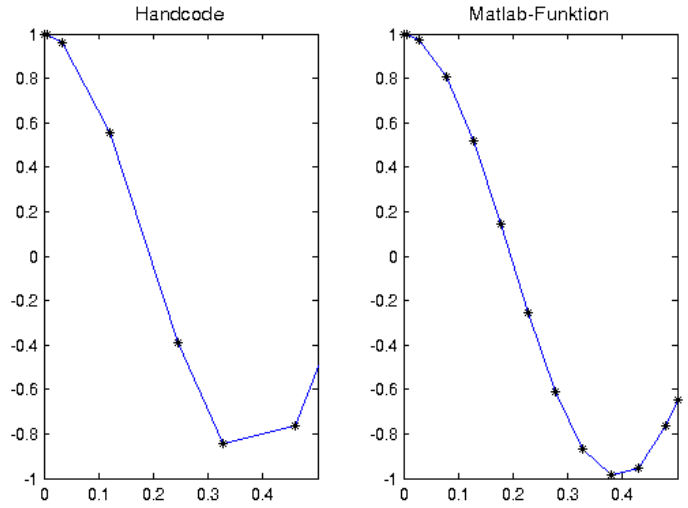
- Alle Berechnungen können mit dem Matlab-Skript ex21.m ausgeführt werden.
- Zunächst muss die Differentialgleichung - wie oben beschrieben - in die Grundform gebracht werden. Die Funktion f kann dann z. B. definiert werden durch

```
omega = 8.0;
f = @(t,y) [y(2); -omega^2*y(1)];
```

- Die einzelnen Formeln des Algorithmus lassen sich direkt nach Matlab übertragen, wenn man noch die Beträge bei Vektoren durch die `norm()`-Funktion ersetzt.
- Alternativ zur Handarbeit können die einzelnen Zeitschritte bis zum Überschreiten der Endzeit bequem in einer `while`-Schleife durchlaufen werden. Dabei werden die anfallenden Ergebnisse (Zeiten und y-Werte) zur späteren Auswertung in Arrays `tall` und `yall` gesammelt.
- Man erhält dann folgende Schritte:

h	t	epslok	
0.0058	0.0058	0.000017	ok
0.0290	0.0348	0.001933	ok
0.0865	0.1213	0.017935	ok
0.1227	0.2440	0.007227	ok
0.2356	0.2440	0.323707	zu groß
0.1274	0.2440	0.187370	zu groß
0.0827	0.3266	0.012660	ok
0.1317	0.4584	0.069428	ok
0.1190	0.5774	0.010382	ok

- Vergleich mit Matlabs `ode23`:



- Am auffälligsten ist die generell kleinere Schrittweite bei Matlab. Ursache ist die Beschränkung von h auf maximal 1/10 des Zeitintervalls. Dadurch ist das Ergebnis insgesamt genauer als der Handcode, aber auch genauer als gefordert. Durch Vergleich mit der bekannten Lösung

  $x(t) = \cos(\omega\, t)$

  erhält man leicht den tatsächlichen maximalen Fehler
  - errMax(Handcode) = 0.1005
  - errMax(Matlab) = 0.0089

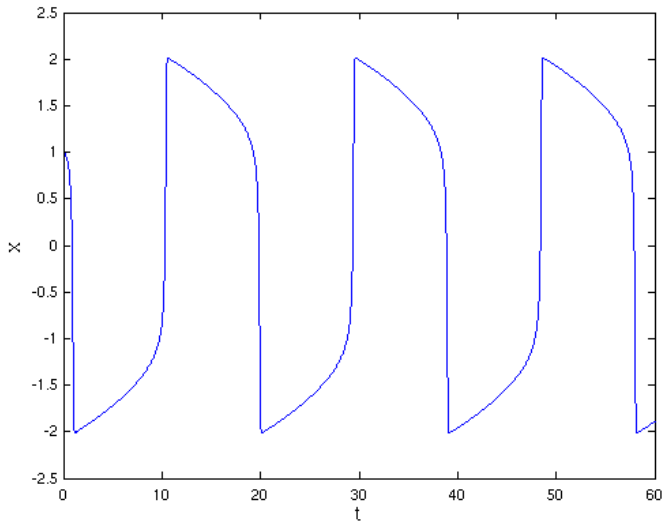
Außerdem wird bei Matlab die Endzeit genau erreicht, während der Handcode darüber hinausschießt. Dies lässt sich durch eine Anpassung der Schrittweite am Ende leicht erreichen.

Lösung von Aufgabe 22

Alle Berechnungen können mit dem Matlab-Skript ex22.m ausgeführt werden.

a. `ode23t` ohne Jacobimatrix

Zur Implementierung der Differentialgleichung wird zunächst die Systemfunktion `f(t, y, mu)` mit dem zusätzlichen Parameter `mu` definiert. In gewohnter Weise wird daraus die Funktion `f1(t,y)` abgeleitet. Um die Ergebnisse der drei Lösungen einfacher vergleichen zu können, wird dem Solver statt des Zeitintervalls `[0 te]` ein ganzer Vektor `0:h:te` mit Ausgabezeitpunkten übergeben. Man erhält als Lösung



b. `ode23t` mit Jacobimatrix

Die Jacobimatrix wird gemäß obiger Formel als Matrixfunktion `J(t, y, mu)` definiert, `J1` hat den festen Parameterwert für `mu`. Mit Hilfe der Optionsstruktur

```
options = odeset("Jacobian", J1);
```

wird sie dem Solver mitgeteilt. Der Unterschied zu den Ergebnisse aus a. ist mit `3.8e-9` deutlich kleiner als die Genauigkeit des Solvers.

c. mit eigenem Solver `odeTrapez`

- Zur Lösung des nichtlinearen Gleichungssystems wird solveNewton.m verwendet.
- Damit ausgerüstet kann das Trapezverfahren programmiert werden. Die nach y(t+h) aufzulösende Gleichung war ja

$$\vec{y}(t+h) - \vec{y}(t) = \frac{h}{2}\left[\vec{f}(t+h, \vec{y}(t+h)) + \vec{f}(t, \vec{y}(t))\right]$$

Nennt man zur Übersicht die gesuchte Größe in x um, kann man die Gleichung schreiben als

$$\vec{F}(\vec{x}) = 0$$

mit der Funktion

$$\vec{F}(\vec{x}) = \vec{x} - \frac{h}{2}\vec{f}(t+h, \vec{x}) - \vec{y}(t) - \frac{h}{2}\vec{f}(t, \vec{y}(t))$$

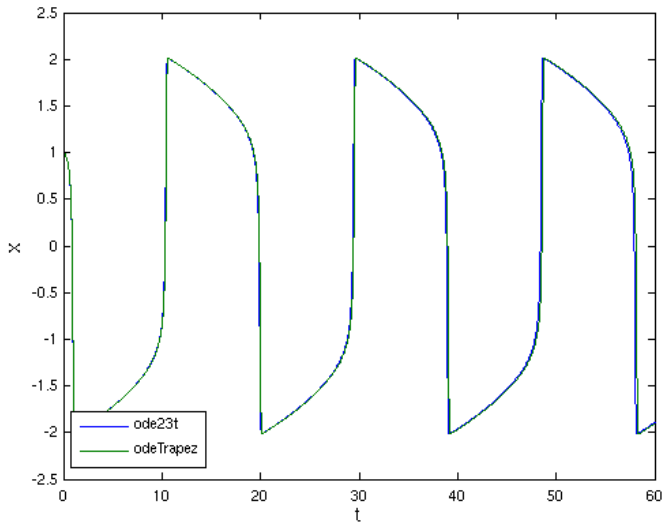
Die zugehörige Funktionalmatrix lässt sich bei bekannter Jacobimatrix J von f leicht berechnen, sie ist

$$\left(\frac{\partial \vec{F}}{\partial \vec{x}}\right)(\vec{x}) = \mathbf{1} - \frac{h}{2}\vec{J}(t+h, \vec{x})$$

Der Solver `odeTrapez(f, tSpan, y0, h, J)` benötigt neben den üblichen Argumenten noch eine feste Schrittweite `h` und die Funktion `J(t,y)` für die Jacobimatrix von `f`. Er definiert in jedem

Zeitschritt die Hilfsfunktionen `F` und `DF` für das Newtonverfahren und bestimmt mit `solveNewton` den nächsten Wert von `y`. Alle Details findet man in odeTrapez.m .

- Plottet man die damit erhaltene Lösung, sieht man, dass die von `odeTrapez` erzeugte Lösung leicht hinterher hinkt



Die Abweichungen an den Sprungstellen werden dadurch zunehmend größer