1 Nichtlineare Gleichungen in einer Unbekannten

1.1 Ein kurzer Rundgang im Garten der Gleichungen

Als Einstieg in die Numerische Mathematik behandeln wir numerische Lösungsverfahren für Gleichungen in einer Unbekannten. *Linear* sind solche Gleichungen, wenn sie sich in der Form

$$kx = d, \quad k, d \text{ gegeben}, \quad x \text{ gesucht}$$

schreiben lassen. Offensichtlich gibt es, falls $k \neq 0$, eine eindeutige Lösung. Dieses Thema ist also vorläufig abgehakt, wir kümmern uns nun um *nichtlineare* Gleichungen. (Die linearen Gleichungen werden uns erst dann intensiver beschäftigen, wenn sie in Massen, als Systemen mit *mehreren* Unbekannten auftreten.)

Analytische oder numerische Lösung Wenn sich durch algebraische Umformungen die Lösung einer Gleichung explizit, also in der Form $x = \ldots$, anschreiben lässt (im obigen Beispiel: $x = d/k$, allgemein ein Term, in dem nur die üblichen Standard-Rechenoperationen und -Funktionen auftreten), spricht man von einer *analytischen* Lösung

Analytisch lösbar sind beispielsweise *quadratische* Gleichungen, also solche, die sich als

$$x^2 + px + q = 0 \quad p, q \text{ gegeben}, \quad x \text{ gesucht}$$

schreiben lassen. Sie kennen sicherlich die Lösungsformel

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

> Es reicht aber nicht, eine Lösungsformel hinschreiben zu können, sie muss auch verlässlich genaue Ergebnisse liefern. Die scheinbar triviale Lösung einer quadratischen Gleichung nach obiger Formel kann recht ungenau werden. Lassen Sie Ihren Taschenrechner damit die kleinere Lösung der quadratischen Gleichung
>
> $$x^2 - 12345678x + 9 = 0$$
>
> berechnen. Der (sechzehnstellig) genaue Wert ist $x_1 = 7{,}290\,000\,597\,780\,479 \times 10^{-7}$. Obwohl übliche Rechner zehn- bis vierzehnstellig genau rechnen, liefern sie nur die ersten paar Stellen richtig. Die numerisch genauere Methode berechnet zuerst die *betragsmäßig größere* Lösung x_1 mit der klassischen Formel und findet dann die *betragsmäßig kleinere* Lösung x_2 mit der alternativen Lösungsformel
>
> $$x_2 = \frac{q}{x_1}.$$

Algebraische und transzendente Gleichungen Lineare, quadratische und kubische Gleichungen sind die einfachsten Beispiele *polynomialer* Gleichungen. Ein Polynom in einer Variablen x ist eine Summe von x-Potenzen, multipliziert mit Koeffizienten, also ein Ausdruck der Form

$$a_n x^n + \cdots + a_2 x^2 + a_1 x + a_0.$$

Die höchste auftretende Potenz heißt die *Ordnung* oder der *Grad* des Polynoms oder der Gleichung.

Kubische Gleichungen und Gleichungen vierter Ordnung sind im Prinzip analytisch lösbar, aber die Formeln (Cardanische Formeln, N. Tartaglia[1], G. Cardano[2], L. Ferrari[3], um 1540) sind so unhandlich, dass sie praktisch kaum verwendet werden. Numerische Verfahren sind in diesen Fällen meist sinnvoller. Sie liefern Näherungen, die schrittweise, mit immer besserer Genauigkeit, die Lösungen anstreben. Ab Polynomgrad fünf gibt es ohnehin keine allgemeinen Lösungsformeln mehr.

> Der junge norwegische Mathematiker Niels Henrik Abel führt 1826 den „Beweis der Unmöglichkeit, algebraische Gleichungen von höheren Graden als dem vierten allgemein aufzulösen“. Ab dem fünften Grad lassen sich Gleichungen also (im Allgemeinen) nicht durch eine *endliche Zahl elementarer Rechenoperationen* (Addition, Subtraktion, Multiplikation, Division, Wurzelziehen) lösen.

Um die Vorstellung der verschiedenen Gleichungstypen zum Abschluss zu bringen: Gleichungen, in denen auch noch Bruchterme, Wurzeln oder rationale Exponenten vorkommen, lassen sich (möglicher Weise nur mit hohem Aufwand und komplizierten Umformungen) auf Systeme polynomialer (man sagt auch: algebraischer) Gleichungen zurückführen. Terme oder Funktionen, die sich nicht mittels endlich vieler elementarer Rechenoperationen formulieren lassen, sind etwas, das die Kräfte der Algebra übersteigt (*„quod vires algebrae transcendit“*, sagte Leibnitz) und heißen deswegen *transzendent*.

Beispielsweise sind die trigonometrischen Funktionen, die Exponentialfunktion und die entsprechenden Umkehrfunktionen transzendente Funktionen. Treten solche Funktionen in Gleichungen auf, ist normaler Weise nur numerische Lösung möglich.

> Explizite Lösungsformeln gibt es nur für polynomiale Gleichungen niedrigen Grades und die allereinfachsten transzendenten Glaichungen. In allen anderen Fällen können nur numerische Methoden eine Lösung finden.

1.2 Begriffe, Probleme, Lösungen

Hier behandelte Aufgabentypen:

$$\begin{aligned} g(x) &= h(x), && \text{Finden einer } \textit{Lösung} \text{ einer Gleichung} \\ f(x) &= 0, && \text{Finden einer } \textit{Nullstelle} \text{ der Funktion } f \\ x &= \phi(x), && \text{Finden eines } \textit{Fixpunktes} \text{ der Funktion } \phi \end{aligned}$$

> Eine Lösung der Gleichung $f(x) = 0$ heißt *Nullstelle* der Funktion f.
> Eine Lösung der Gleichung $x = \phi(x)$ heißt *Fixpunkt* der Funktion ϕ.

> Eine Fixpunkt-Gleichung $x = \phi(x)$ lässt sich natürlich sofort umformen auf $\phi(x) - x = 0$. Jeder Fixpunkt von ϕ ist also zugleich Nullstelle von $f(x) = \phi(x) - x$.
> Selbstverständlich muss der Funktionsterm in einer Nullstellen-Aufgabe nicht automatisch f heißen, ebensowenig wie in Fixpunkt-Gleichungen die Funktion mit ϕ bezeichnet sein muss. Die Vorlesungsunterlagen schreiben aber in der Regel $x = \phi(x)$, wenn diese Gleichung durch Umformen aus $f(x) = 0$ entstanden ist.

[1] Niccolò Fontana Tartaglia verriet Cardano die Lösung unter dem Siegel der Verschwiegenheit; war stinksauer, als der sie trotzdem veröffentlichte.
[2] auch bekannt durch Kardanwelle und kardanische Aufhängung, die er ebenfalls nicht erfunden hat
[3] Das Rennen um die Lösung für Gleichungen vierten Grades, sozusagen die Formel Vier, wurde damals von Ferrari gewonnen.

Nullstellen von Polynomen nennt man auch *Wurzeln*.[4]

Eine *analytische Lösung* ist ein expliziter Ausdruck, in dem nur bekannte Größen und Funktionen vorkommen.

> Welche Funktionen dabei als „bekannt“ vorausgesetzt werden, ist nicht exakt festgelegt. Letztlich lassen sich auch von so geläufigen Funktionen wie Sinus oder Cosinus Werte nur durch numerische Verfahren berechnen – auch wenn Ihnen der Taschenrechner diese Arbeit abnimmt.

Demgegenüber steht die *numerische Lösung*, eine Rechenvorschrift, die eine schon irgendwie bekannte Näherung schrittweise verbessert.

Mehrfache Nullstellen: Eine Funktion f hat an der Stelle x eine genau n-fache Nullstelle, wenn zugleich $f(x) = 0, f'(x) = 0, f''(x) = 0, \ldots, f^{(n-1)}(x) = 0$ und $f^{(n)}(x) \neq 0$. (Dabei setzen wir die Existenz stetiger Ableitungen mindestens bis zur n-ten Ordnung voraus.)

Die auftretenden Funktionen $f, g, \ldots$ und Variablen $x, y, \ldots$ bezeichnen in dieser Vorlesung in der Regel *reelle* Größen. Die *komplexen* Zahlen sind an sich der natürliche Lebensraumfür Polynome und Funktionen (unter anderem deswegen, weil Polynome n-ten Grades dort immer genau n Nullstellen haben–wobei manche mehrfach gezählt werden, damit das so stimmt; Fundamentalsatz der Algebra). Die meisten Definitionen und Verfahren lassen sich leicht für komplexe Variable und komplexwertige Funktionen verallgemeinern. Trotzdem beschränken wir uns (abgesehen von gelegentlichen Hinweisen) auf Rechenverfahren in den reellen Zahlen.

Checkliste zum Lösen nichtlinearer Gleichungen

Gleichzeitig Inhaltsangabe und Stoffübersicht der folgenden Abschnitte.

- Vorarbeiten
  - Überblicken Sie den Verlauf der Funktionen (Wertetabelle, graphische Darstellung).
  - Definitionsbereich? Wo können die Lösung liegen? Wie viele Lösungen gibt es?
  - Lassen sich günstige Umformungen finden?
- Trivialmethoden für Computer oder Taschenrechner
  - Systematisches Einsetzen in Wertetabelle
  - Hineinzoomen im Funktionsgraph
- Klassische Lösungsverfahren
  - Intervallhalbierung
  - Sekantenmethode und Regula Falsi
  - Newton-Verfahren (heißt auch Newton-Raphson-Verfahren)
  - Fixpunkt-Iteration

1.3 Beispiele zum Aufwärmen

In den Übungen und in der Vorlesung diskutieren wir Beispiele der folgenden Art. Auch die folgenden Abschnitte 1.5 und 1.6 bringen weitere Erklärungen.

[4] Allerdings klingt „Wurzel“ statt „Lösung“ oder „Nullstelle“ im heutigen Fachdeutsch eher veraltet; im Englischen ist *root of a polynomial* der gängige Fachausdruck, und auch *root of a function or an equation* ist neben *zero of a function or solution of an equation* durchaus üblich.

Aus der Finanzmathematik

Ein Kredit von 100.000 € soll in 180 Monatsraten zu je 900 € zurückgezahlt werden. Was ist der Zinssatz bei diesen Konditionen?

Die Rentenformel für nachschüssige Zahlung liefert für den (monatlichen) Aufzinsungsfaktor q die Gleichung

$$900 = 100\,000\frac{q-1}{1-q^{-180}}. \tag{1}$$

Zustandsgleichung eines realen Gases

Wie groß ist das Molvolumen von Stickstoff bei 20 C und $1\,\mathrm{bar} = 1 \times 10^5\,\mathrm{Pa}$ nach der Van der Waals-Gleichung?

Die Zustandsgleichung

$$\left(p + \frac{a}{V_{mol}^2}\right)(V_{mol} - b) = RT$$

beschreibt den Zusammenhang zwischen Druck p, Molvolumen V_{mol} und Temperatur T. Die Konstanten a und b haben für Stickstoff die Werte

$$a = 0{,}129\,\mathrm{Pa\,m^6/mol^2}, \quad b = 38{,}6 \times 10^{-6}\,\mathrm{m^3/mol}.$$

Die molare Gaskonstante ist $R = 8{,}3145\,\mathrm{J/molK}$. Nach Einsetzen der Zahlenwerte verbleibt als Gleichung für V_{mol}:

$$\left(100\,000 + \frac{0{,}129}{V_{mol}^2}\right)(V_{mol} - 0{,}000\,038\,6) = 2437{,}4 \tag{2}$$

Widerstände in Rohrleitungen

Die Rohrreibungszahl λ hängt von der Reynoldszahl Re ab. Bei laminarer Strömung gilt einfach $\lambda = 64/\mathrm{Re}$. Im turbulenten Bereich, ab etwa $\mathrm{Re} > 2000$, listen technische Handbücher verschiedene, teilweise empirische Formeln für λ. Auf theoretischem Weg hat Prandtl für ein glattes Rohr die Beziehung

$$\lambda = \frac{1}{(2\log_{10}(\mathrm{Re}\sqrt{\lambda}) - 0{,}8)^2} \tag{3}$$

abgeleitet, die bis $\mathrm{Re} = 3{,}4 \times 10^6$ mit Versuchen übereinstimmt. Wie groß ist λ bei $\mathrm{Re} = 1 \times 10^6$?

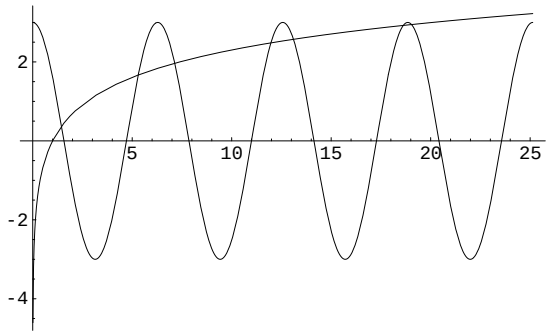


Abbildung 1: Schaubild zur Gleichung $3\cos x = \log x$. Den x-Werten der Schnittpunkte der Funktionsgraphen entsprechen die Lösungen der Gleichung.

Ohne tiefere Bedeutung

Es ist gut, wenn die bisherigen Beispiele den Eindruck einer gewissen Praxisnähe vermittelt haben. Der technischen Hintergrund und damit verbundene Verständnisschwierigkeiten verstellen aber den Blick auf die mathematischen Inhalte. Sie lernen hier nicht Physik, sondern numerische Verfahren, und die lassen sich leichter an einfachen Musterbeispielen illustrieren. Deswegen:

Finden Sie die Lösungen der Gleichung

$$3\cos x = \log x \tag{4}$$

Wichtiger Hinweis: hier meint log natürlich den natürlichen Logarithmus[5]. Argumente in Winkelfunktionen sind immer im Bogenmaß einzusetzen!

1.4 Graphische Lösung: Ein Bild sagt mehr als tausend Formeln

Entsprechend der Checkliste aus Kapitel 1.2 verschaffen wir uns am Beispiel von Gleichung 4 einen ersten Überblick. Diese Gleichung läßt nicht unmittelbar erkennen, ob, wo und wieviele Lösungen sie hat. Da sowohl Cosinus als auch Logarithmus geläufige Funktionen sind, bietet sich eine graphische Darstellung an. (Abbildung 1). Aus dem Schaubild lässt sich die Anzahl und ungefähre Lage der Lösungen erkennen. Rechenprogramme, die Wertetabellen berechnen oder in einen Funktionsgraphen hineinzoomen können, liefern rasch brauchbare Werte (die Checkliste nennt diese Vorgangsweisen „Trivialmethoden“).

1.5 Passende Umformungen: Nullstellen und Fixpunkte

Die Lösungen der Gleichung $3\cos x = \log x$ sind genau die Nullstellen der Funktion $f(x) = 3\cos x - \log x$. Ein Vergleich von Abbildung 1 mit Abbildung 2 stellt diesen Sachverhalt klar

[5] Für den dekadischen Logarithmus sprechen außer der evolutionsbedingten Zufälligkeit, dass Menschen zehn Finger haben, keine Argumente. Für Leute, die nicht bis drei zählen können, ist die Basis $e = 2{,}718\,281\,8\ldots$ ohnedies natürlicher.

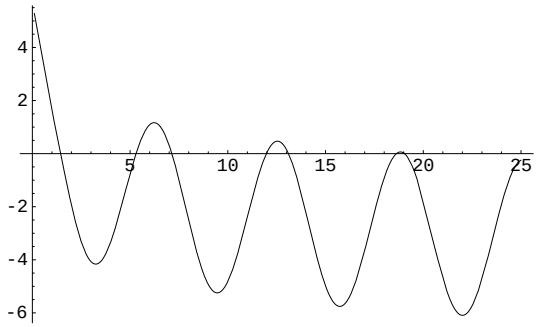


Abbildung 2: Schaubild zur Funktion $f(x) = 3\cos x - \log x$. Die Nullstellen von f entsprechen den x-Werten der Schnittpunkte in der Abbildung 1

und zeigt zum Beispiel: In der Nähe von $x = 5$, jedenfalls im Bereich $4 < x < 6$, muss eine der Nullstellen von f liegen.

Welche Form der graphischen Darstellung man günstigerweise wählt, hängt von der gegebenen Gleichung ab. In diesem Beispiel lassen sich cos und log als bekannte Funktionen leicht skizzieren, deswegen ist die Darstellung der Lösung durch die (x-Werte der) Schnittpunkte zweier Kurven übersichtlich. Andererseits lässt die Darstellung von $f(x) = 3\cos x - \log x$ die Nullstellen unmittelbar erkennen. Die klassischen Methoden zum Finden von Nullstellen ab Kapitel 1.7 erfordern ohnedies eine solche Umformung der Gleichung.

Die Gleichung $3\cos x = \log x$ lässt sich aber auch beispielsweise umformen zu

$$x = \arccos \frac{\log x}{3} \quad . \tag{5}$$

In dieser Form liegt eine Fixpunkt-Aufgabe $x = \phi(x)$ vor, mit $\phi(x) = \arccos((\log x)/3)$.

Fixpunkt-Iteration

Was passiert, wenn man auf der rechten Seite von Gleichung 5 einen Wert für x einsetzt, den Ausdruck ausrechnet und das Ergebnis wieder in der rechten Seite einsetzt? Beginnend etwa mit $x = 1$ liefert dieses Verfahren die Folge

$$1; \quad 1{,}5708; \quad 1{,}419\,69; \quad 1{,}453\,72; \quad 1{,}445\,76; \quad 1{,}447\,61; \quad 1{,}447\,18\ldots$$

Die Folge konvergiert gegen $\xi = 1{,}447\,258\,6$, das ist die kleinste Lösung der gegebenen Gleichung und gleichzeitig der einzige Fixpunkt der Funktion

$$\phi(x) = \arccos \frac{\log x}{3}.$$

Sie sehen hier ein Beispiel einer *Fixpunkt-Iteration*.

Fixpunkt-Iteration
Gegeben eine Gleichung $x = \phi(x)$.

> Beginne mit einem Startwert
> Setze Wert auf rechter Seite der Formel ein und werte aus
> Setze das Ergebnis wieder und wieder rechts in die Formel ein, bis sich die Resultate nicht mehr ändern

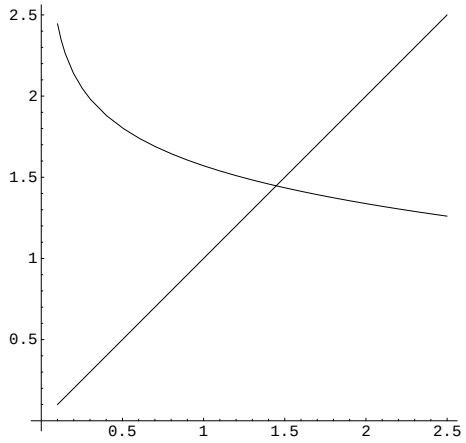


Abbildung 3: Schaubild zur Fixpunktaufgabe mit der Funktion $\phi(x) = \arccos((\log x)/3)$. Der Fixpunkt von ϕ entspricht der Nullstelle von f in der Nähe von 1,4. Weitere Fixpunkte von ϕ gibt es nicht. Durch die Umformulierung sind alle anderen Lösungen der ursprünglichen Gleichung verlorengegangen!

Weitere Beispiele von Fixpunkt-Iterationen:

- Geben Sie eine Zahl in den Taschenrechner ein und drücken Sie wiederholt auf die Wurzeltaste. Die Ergebnisse konvergieren gegen 1 (Fixpunkt von $f(x) = \sqrt{x}$).
- Geben Sie eine Zahl < 20 in den Taschenrechner ein und drücken Sie abwechselnd wiederholt auf die Tasten exp und $1/x$. Die Ergebnisse (nach dem $1/x$-Schritt) konvergieren gegen 0,567 14 (Fixpunkt von $f(x) = 1/\exp x$).
- Berechnen von Quadratwurzeln war schon in der griechischen Antike ein wichtiges Problem und (für rationale Zahlen) gelöst. Die Wurzel aus a ist definiert als Lösung von $x^2 = a$; eine für $x \neq 0$ äquivalente Umformung dieser Gleichung ist

  $$x = \frac{1}{2}\left(x + \frac{a}{x}\right) \ .$$

  Schon den Babyloniern soll die oft als Heron-Verfahren bezeichnete Iteration

  $$x^{(0)} = a; \quad x^{(k+1)} = \frac{1}{2}\left(x^{(k)} + \frac{a}{x^{(k)}}\right) \text{ für } k = 0, 1, 2, \ldots$$

  bekannt gewesen sein.
- Gleichung 3 ist eine Fixpunkt-Gleichung. Mit dem Startwert 0,05 liefern wenige Fixpunkt-Iterationen eine genaue Lösung.

Aber es funktioniert nicht immer: Eine andere mögliche Fixpunkt-Form von Gleichung 4 lautet

$$x = \exp(3\cos x) \ .$$

Wenn Sie hier $x = 1$ rechts einsetzen und das für die Ergebnisse jeweils wiederholen, erhalten Sie die Folge

$$1; \quad 5{,}057\,68; \quad 2{,}760\,46; \quad 0{,}061\,745\,5; \quad 19{,}971; \quad 3{,}6805\ldots$$

Ihre Werte wechseln unregelmäßig und konvergieren nicht.

Zusammenfassung

Nicht jede Fixpunkt-Iteration konvergiert. Passende Umformungen sind nicht immer leicht zu finden. Anderseits sind viele numerische Verfahren vom Typ einer Fixpunkt-Iteration. Das rechtfertigt eine ausführliche theoretische Untersuchung solcher Verfahren im Kapitel 1.12.

1.6 Diskussion der Beispiele: Wichtige und unwichtige Terme

Hier werden die in Kapitel 1.1 vorgestellten Beispiele ausführlich besprochen.

1.6.1 Eine fast lineare Gleichung

Die am Anfang von Kapitel 1.1 erwähnte Gleichung

$$x^2 - 12345678x + 9 = 0$$

ist, wenn es um die betragskleinere der beiden Lösungen geht, eigentlich keine quadratische Gleichung! Begründung: Die gesuchte Lösung ist von der Größenordnung 10^{-6} bis 10^{-7}; der Term x^2 in der Gleichung ist also gegenüber dem linearen Term $12345678x$ um mehr als zehn Größenordnungen kleiner. Für alle praktischen Zwecke ist eine solche Gleichung linear mit einem kleinen quadratischen Korrekturterm. Lösen Sie daher nach dem linearen Term auf:

$$x = \frac{1}{12345678}(x^2 + 9)\ .$$

Der Startwert $x^{(0)} = 0$ liefert selbst auf den billigsten Taschenrechnern ohne Wurzeltaste bereits ein bessere Näherung $x^{(1)} = 7{,}290\,000\,597\,78 \times 10^{-7}$ als die meisten Rechner durch Anwendung der Standard-Lösungsformel erreichen können.

> Locker formuliert: Viele Gleichungen enthalten Terme, in denen die Unbekannte zwar auftritt, aber im Vergleich zu anderen Termen wenig Einfluss hat. Wenn eine Gleichung dadurch leichter lösbar wird, lassen sich solche Terme in erster Näherung vernachlässigen. In weiteren Schritten korrigiert man das Ergebnis, indem man Näherungswerte in den anfangs vernachlässigten Termen einsetzt.

1.6.2 Van der Waals-Gleichung

Die Gleichung (2) lässt sich zu einer kubischen Gleichung umformen,

$$-4.9794 \cdot 10^{-6} + 0.129 V_{mol} - 2441.3 V_{mol}^2 + 100000 V_{mol}^3 = 0 \quad , \tag{6}$$

und wäre damit im Prinzip analytisch lösbar. Tun Sie's nicht! Ein wenig Einsicht in den physikalischen Hintergrund dieser Gleichung legt eine andere Vorgangsweise nahe: Bei Zimmertemperatur ist Stickstoff nahezu ein ideales Gas, das der Gleichung

$$pV_{mol} = RT$$

gehorcht. In der Van der Waals-Gleichung

$$\left(p + \frac{a}{V_{mol}^2}\right)(V_{mol} - b) = RT \tag{7}$$

ist der Term a/V_{mol}^2 eine Korrektur der idealen Gasgleichung und für die im Beispiel gegebenen Parameter gegenüber p vernachlässigbar klein. Dem umgeformten Polynom (6) sieht man es

nicht an, aber die ursprüngliche Gleichung (7) entspricht – im Bereich der gegebenen Daten – nicht einer „richtigen“ kubischen Gleichung, sondern vielmehr einer lineare Gleichung in V_{mol} plus einem kleinen Korrekturterm a/V_{mol}^2.

Daher lässt sich diese Gleichung auflösen, wenn man „unwichtige“ Terme der Unbekannten auf der rechten Seite stehen lässt. Hier formen wir um zu

$$V_{mol} = \frac{RT}{p + a/V_{mol}^2} + b = \frac{2437{,}4}{100000 + 0{,}129/V_{mol}^2} + 0{,}000\,038\,6$$

und ignorieren wir erst einmal den Korrekturterm a/V_{mol}^2. Das liefert eine nullte Näherung für das Molvolumen,

$$V_0 = \frac{2437{,}4}{100000} + 0{,}000\,038\,6 = 0{,}024\,413\ .$$

Der Trick ist nun, diese Näherung für V_{mol} in der rechten Seite der Gleichung einzusetzen und daraus eine verbesserte Näherung

$$V_1 = \frac{2437{,}4}{100000 + 0{,}129/0{,}024\,413^2} + 0{,}000\,038\,6 = 0{,}024\,360$$

zu berechnen. Wiederholtes Einsetzen liefert keine weitere Verbesserung:

$$V_2 = \frac{2437{,}4}{100000 + 0{,}129/0{,}024\,360^2} + 0{,}000\,038\,6 = 0{,}024\,360\ .$$

Damit haben wir (jedenfalls auf fünf Dezimalstellen genau) den Wert $V_{mol} = 0{,}024\,360\,\mathrm{m}^3$ bestimmt.

> Bußübung für die Fastenzeit: Schlagen Sie in Wikipedia die Cardanischen Formeln nach und lösen Sie die Aufgabe damit. Vergleichen Sie den Zeitaufwand mit der obigen Methode.

1.6.3 Finanzmathematik

In Gleichung 1 erwarten wir für den Aufzinsungsfaktor q einen Wert knapp über 1. Den Term q^{-180} im Nenner wird vermutlich $\ll 1$ und nicht so wichtig sein. Das motiviert, die Gleichung nach dem q im Zähler aufzulösen.

$$q = 1 + \frac{900}{100000}(1 - q^{-180})$$

Ignoriert man q^{-180} auf der rechten Seite, dann folgt als nullte Näherung

$$q_0 = 1 + \frac{900}{100000} = 1{,}009$$

Auch hier funktioniert der Trick, q_0 in der rechten Seite einzusetzen und daraus eine verbesserte Näherung

$$q_1 = 1 + \frac{900}{100000}(1 - 1{,}009^{-180}) = 1{,}007\,206$$

zu berechnen. Wiederholtes Einsetzen liefert

$$q_2 = 1{,}006\,529 \quad q_3 = 1{,}006\,210 \quad q_4 = 1{,}006\,047\ldots$$

Es braucht aber hier insgesamt 14 Iterationen, bis sich die Werte bei $q = 1{,}005\,851$ stabilisieren.

Bemerkungen zum Abschluss

Ist eine Gleichung in der Form $f(x) = g(x)$ gegeben (Beispiel: Gleichung 4), lässt sich nicht unmittelbar erkennen, welche Terme „wichtig“ oder „unwichtig“ sind. Regel: man löse nach jener Seite der Gleichung auf, welcher den *steileren* Funktionsgraph im Schnittpunkt hat.

Passende Umformungen für Fixpunkt-Iterationen erfordern oft ein tieferes Verständnis der einzelnen Terme in einer Gleichung. Es gibt zum Glück Lösungsverfahren, die mehr nach „Schema F“ ablaufen. Eines davon stellt das nächste Kapitel vor.

1.7 Intervallhalbierung

Kennen Sie die Geschichte von den zwei Möglichkeiten? Sie beginnt mit dem Zwischenwertsatz.

Zwischenwertsatz

Eine Funktion f, die auf einem abgeschlossenen Intervall $[a, b]$ stetig ist, nimmt in diesem Intervall auch jeden Wert zwischen $f(a)$ und $f(b)$ an.

Ist f insbesondere für $x = a$ negativ und für $x = b$ positiv (oder umgekehrt), dann garantiert der Zwischenwertsatz: f hat mindestens eine Nullstelle in diesem Intervall.

Es gibt immer zwei Möglichkeiten...

Angenommen, wir suchen eine Nullstelle einer im Bereich $a \leq x \leq b$ stetigen Funktion. Es lässt sich rechnerisch sofort prüfen, ob $f(a)$ und $f(b)$ unterschiedliches Vorzeichen haben. Wenn ja, dann garantiert der Zwischenwertsatz die Existenz eine Nullstelle im Bereich $a \leq x \leq b$, aber wir wissen nicht, wo sie liegt. Nun gibt es zwei Möglichkeiten: Entweder ist $b - a$ klein, dann ist es gut: Wir können sowohl a als auch b als Näherung für eine Nullstelle von f auffassen. Andernfalls berechnen wir den Mittelpunkt c des Intervalls, $c = (a + b)/2$. Nun gibt es wieder zwei Möglichkeiten. Ist $f(c) = 0$, so ist es gut: es liegt dort eine Nullstelle vor. Anderenfalls hat f an den Enden eines der Teilintervalle $a \leq x \leq c$ oder $c \leq x \leq b$ verschiedene Vorzeichen (klar? Das ist der springende Punkt!). In einem der beiden Intervalle muss also eine Nullstelle liegen. Betrachten wir dieses Intervall und nennen wir der Einfachheit die neuen Intervallgrenzen wieder a und b.

Nun gibt es zwei Möglichkeiten: Entweder ist $b-a$ klein, dann ist es gut: Wir können sowohl a als auch b als Näherung für eine Nullstelle von f auffassen. Andernfalls bilden wir $c = (a+b)/2$. Nun gibt es wieder zwei Möglichkeiten...

Sie können nun die Geschichte selber fortsetzen. Beachten Sie aber, dass die Intervalllänge in jedem Erzählschritt halbiert wird. Für jede beliebig klein vorgegebene Genauigkeitsschranke $\epsilon > 0$ erreichen Sie nach einer endlichen Anzahl von Schritten ein Intervall mit Länge $b-a < \epsilon$. Damit endet die Geschichte wie im wirklichen Leben: Es gibt immer zwei Möglichkeiten, aber jede Entscheidung schränkt den Freiraum für weitere Aktionen ein. Irgendwann sind die Alternativen dann doch ausgeschöpft.

Formalisiert angeschrieben, lautet dieses Verfahren

Intervallhalbierung (Bisektionsverfahren)
Gegeben eine Funktion f, zwei Werte a und b mit $f(a) \cdot f(b) < 0$, eine Fehlerschranke $\epsilon > 0$. Ist f im Intervall $a \leq x \leq b$ stetig, dann findet dieser Algorithmus die Näherung c an eine Nullstelle ξ von f mit Fehler $|c - \xi| < \epsilon$.

Wiederhole
  setze $c \leftarrow (a + b)/2$
  falls $f(a) \cdot f(c) < 0$
    setze $b \leftarrow c$
  sonst
    setze $a \leftarrow c$
bis $|b - a| < \epsilon$ oder $f(c) = 0$

Lineare Konvergenz

Die beste Schätzung für den Wert der Nullstelle ist der Mittelpunkt des Intervalls. Der maximale Fehlerbetrag ist dann durch $\epsilon_0 \leq |b-a|/2$ beschränkt; größer als die halbe Intervallbreite kann er nicht sein. Intervallhalbierung reduziert diese Fehlerschranke pro Schritt um den Faktor $1/2$ oder, da

$$\left(\frac{1}{2}\right)^{3{,}3} \approx \frac{1}{10} \quad ,$$

um einen Faktor $1/10$ pro (durchschnittlich) 3,3 Schritten. Man kann sagen: Intervallhalbierung produziert eine korrekte Dezimalstelle pro 3,3 Iterationen. Der maximale Fehler nach dem i-ten Schritt, ϵ_i, ist höchstens halb so groß wie der vorherige maximale Fehler ϵ_{i-1}. Es gilt also

$$\epsilon_i \leq C\epsilon_{i-1} \quad \text{mit } C = \frac{1}{2} \; .$$

Allgemein: Wenn bei einem Verfahren für die Fehlerschranken aufeinanderfolgender Iterationsschritte gilt

$$\epsilon_i \leq C\epsilon_{i-1} \quad \text{mit } C < 1 \; .$$

spricht man von *linearer* Konvergenz.

Vor- und Nachteile

Vorteile der Intervallhalbierung: einfach zu verstehen, leicht zu programmieren. Wenn die Voraussetzungen erfüllt sind, konvergiert es mit Sicherheit. Es ist ein *Einschlussverfahren*, das heißt, es liefert nicht nur einen Näherungswert, sondern grenzt die Lösung von beiden Seiten her ein.

Nachteile: Man braucht Startwerte–aber das ist ein Problem jedes numerischen Verfahrens. Intervallhalbierung ist langsam; nur lineare Konvergenz – die dafür aber sicher.

1.8 Regula Falsi (lineares Eingabeln)

Funktionen, die in der Umgebung der Nullstelle glatt verlaufen, lassen sich dort durch eine Gerade annähern. Statt, wie bei der Intervallhalbierung, den Wert c genau in der Mitte zwischen a und b anzunehmen, wählen wir c als Nullstelle der Gerade durch $(a, f(a))$ und $(b, f(b))$, siehe Abbildung 4.

$$c = a - f(a)\frac{a-b}{f(a)-f(b)} = \frac{af(b) - bf(a)}{f(b) - f(a)}$$

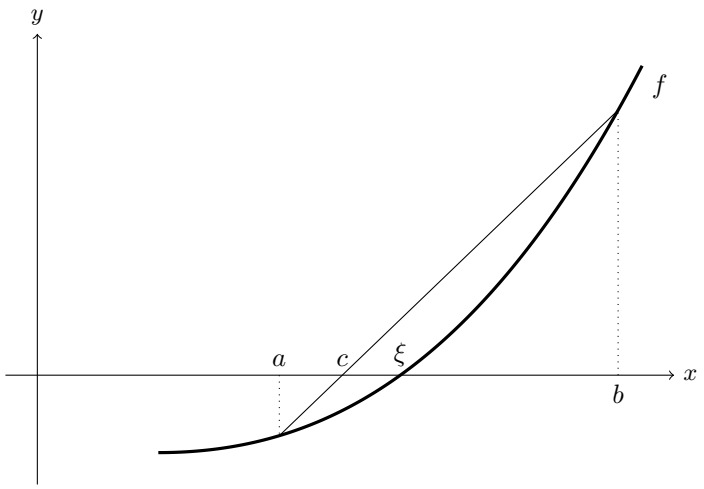


Abbildung 4: Die Regula Falsi berechnet c, die Nullstelle der Verbindungsgeraden, als Näherungswert an die Nullstelle ξ der Funktion f.

Regula Falsi (lineares Eingabeln)
Gegeben eine Funktion f, zwei Werte a und b mit $f(a) \cdot f(b) < 0$ und eine Genauigkeitsschranke $\epsilon > 0$. Ist $f(x)$ im Intervall $a \leq x \leq b$ stetig, dann findet dieser Algorithmus[a] eine Näherung c an eine Nullstelle ξ von f mit Genauigkeit $|c - \xi| < \epsilon$.

Wiederhole
  setze $c \leftarrow a - f(a)\dfrac{a-b}{f(a)-f(b)}$
  falls $f(b) \cdot f(c) < 0$
    setze $a \leftarrow b$
  sonst
    (klassische Version) nix
    (Illinois-Variante) reduziere $f(a)$ auf $\frac{1}{2}f(a)$
    (Pegasus-Variante) reduziere $f(a)$ auf $\dfrac{f(a)f(b)}{f(b)+f(c)}$
  setze $b \leftarrow c$
bis $|b-a| < \epsilon$ oder $f(c) = 0$

[a] mit dem hier gegebenen Abbruchkriterium allerdings nur die beiden Varianten

Allerdings bringt die Regula Falsi in der Standard-Version im Vergleich zur Intervallhalbierung kein wesentlich besseres Konvergenzverhalten. Typischer Weise bleibt nach einigen Iterationen die Intervallgrenze a fix, die andere Grenze b konvergiert zwar zur Nullstelle, aber die Abbruchbedingung $|b - a| < \epsilon$ wird nicht erreicht. Sorgfältige Programmierer würden im obigen Algorithmus jedenfalls noch eine Notbremse einbauen: zähle die Anzahl der Iterationen mit und brich ab, wenn eine Maximalzahl überschritten wird.

Die Illinois- oder die Pegasus-Variante verbessern das Konvergenzverhalten im Vergleich zur Intervallhalbierung deutlich; mutige Programmierer würden in diesem Fall auf die Abfrage

nach einer maximalen Iterationszahl verzichten.

Intervallhalbierung und die verschiedenen Regula-Falsi-Versionen haben gemeinsam, daß sie die Nullstelle von beiden Seiten her „eingabeln“ — sie sind Einschlussverfahren, das ist gut. Nachteilig ist, dass man zu Beginn des Verfahrens zwei Näherungswerte braucht, und zwar je einen auf jeder Seite der Nullstelle. Das kann sehr schwer zu erreichen sein, wenn man zwei nahe beisammen liegende Nullstellen hat, da dann eine der ursprünglichen Näherungen dazwischen liegen muß. Mehrfache Nullstellen gerader Ordnung können diese Verfahren überhaupt nicht finden.

> Was ist „falsch“ an der Regula Falsi? Natürlich nicht die Regel selbst, sondern die angenommenen Startwerte a und b. Aus diesen beiden „falschen Lösungen“ berechnet die Regel eine bessere Näherungslösung.
> Die Methode ist uralt, die Grundidee war schon Jahrhunderte vor Chr. weltweit bekannt: Babyloniern, Ägypter, Inder und Chinesen lösten damit lineare Gleichungen. Aus arabischen Quellen nach Europa bringt sie um 1200 Leonardo von Pisa, genannt Fibonacci. Er beschreibt mehrere Varianten, darunter die *regula duarum falsarum positionum*, die „Methode vom doppelten falschen Ansatz“. So sollte sie auch richtiger Weise heißen, aber es hat sich schlampig verkürzt „Regula Falsi“ durchgesetzt.
> Fibonacci löste damit nur lineare Probleme; da berechnet die Regel aus zwei falschen Startwerten sofort die richtige Lösung. Die Anwendung als iteratives Verfahren für Nullstellen nichtlinearer Funktionen ist dann doch nicht so alt. Mitte des vorigen Jahrhunderts fand man kleine, aber nicht unwesentliche Verbesserungen der Rechenregel (Pegasus-, Illinois-Variante). Sogar noch kürzlich, 2020, veröffentlichten Oliveira und Takahashi eine weitere Verbesserung (`https://en.wikipedia.org/wiki/ITP_method`).

1.9 Sekantenmethode

Die Sekantenmethode berechnet gleich wie die Regula Falsi eine neue Näherung durch lineare Interpolation, verlangt aber nicht, dass die Werte a und b die Nullstelle einschließen, siehe Abbildung 5.

Die formale Beschreibung des Verfahrens bezeichnet hier die Startwerte a und b mit $x^{(0)}$ und $x^{(1)}$ und die weiteren iterativ berechneten Näherungswerte mit $x^{(k)}, x^{(k+1)}, \ldots$

Sekantenmethode
Gegeben eine Funktion f, zwei Werte $x^{(0)}$ und $x^{(1)}$, eine Genauigkeitsschranke $\epsilon > 0$ und eine maximale Iterationsanzahl k_{max}. Für hinreichend gute Startwerte $x^{(0)}$ und $x^{(1)}$ findet dieser Algorithmus die Näherung $x^{(k)}$ an eine Nullstelle ξ von f mit Genauigkeit $|x^{(k)} - \xi| \approx \epsilon$ oder bricht nach einer Maximalzahl von k_{max} Schritten ab.

setze $k = 1$
Wiederhole
  setze $x^{(k+1)} = x^{(k)} - f(x^{(k)}) \dfrac{x^{(k)} - x^{(k-1)}}{f(x^{(k)}) - f(x^{(k-1)})}$
  erhöhe $k = k + 1$
bis $|x^{(k+1)} - x^{(k)}| < \epsilon$ oder $k \geq k_{max}$

Superlineare Konvergenz

Die Sekantenmethode zeigt *superlineare* Konvergenz. (Notwendige technische Details: f zweimal stetig differenzierbar, keine mehrfache Nullstelle.) Das heißt, für die Fehlerschranken $|x^{(k+1)} - \xi|$ und $|x^{(k)} - \xi|$ aufeinanderfolgender Schritte gilt, sofern $|x^{(k)} - \xi|$ schon hinreichend klein ist:

$$|x^{(k+1)} - \xi| \leq C|x^{(k)} - \xi|^p \quad \text{mit } p > 1 \ .$$

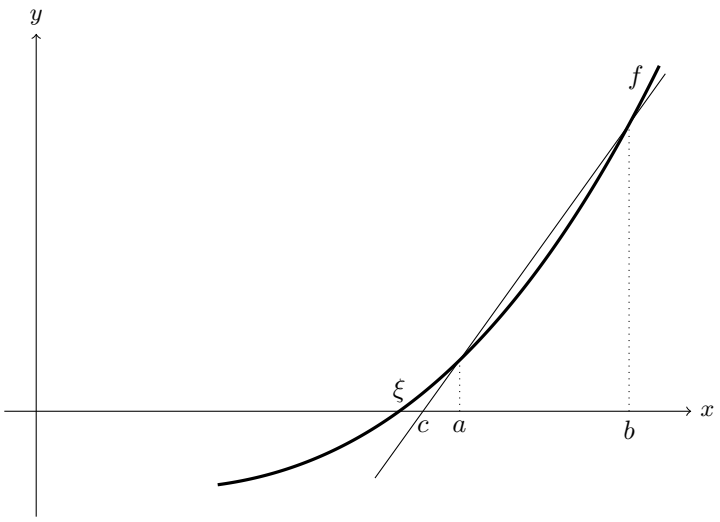


Abbildung 5: Die Sekantenmethode berechnet den nächsten Näherungswert c mittels einer Schnittgeraden (Sekante) durch zwei Punkte des Funktionsgraphen. Die beiden Werte a und b schließen die Nullstelle ξ jedoch nicht unbedingt ein.

Der Fehler reduziert sich also nicht bloß um einen Faktor C, sondern zusätzlich noch mit der Potenz p. Für die Sekantenmethode lässt sich zeigen

$$p = \frac{1+\sqrt{5}}{2} \approx 1{,}618 \ .$$

Angenommen, es ist $|x^{(k)} - \xi| = 0{,}01$. Überlegen Sie sich, was den Fehler stärker verringert: Multiplikation mit einem Faktor $C = 1/2$, oder Potenzieren mit $p = 1{,}6$!

1.10 Newton-Verfahren

Heißt auch Newton-Raphson-Verfahren, aber erst einige Jahrzehnte nach Isaac Newton und Joseph Raphson formuliert Thomas Simpson das Verfahren so, wie wir es heute kennen.

Gesucht sei eine Nullstelle der Funktion f. Gegeben sei ein Startwert $x^{(0)}$ in der Nähe der Nullstelle. Das Newton-Verfahren versucht, ähnlich der Sekantenmethode, die Funktion f durch eine lineare Funktion anzunähern und verwendet dazu die Tangente an f im Punkt $(x^{(0)}, f(x^{(0)}))$. Der Schnittpunkt der Tangente mit der x-Achse ist der nächste Näherungswert, siehe Abbildung 6.

Herleitung aus der Taylorentwicklung von f um den Punkt $x^{(0)}$. Ist f genügend oft differenzierbar, dann gilt:

$$f(x) = f(x^{(0)}) + (x - x^{(0)})f'(x^{(0)}) + \frac{(x-x^{(0)})^2}{2!}f''(x^{(0)}) + \dots$$

Es soll gelten $f(x) = 0$. Vernachlässigen von Gliedern höherer Ordnung liefert die Gleichung

$$0 = f(x^{(0)}) + (x - x^{(0)})f'(x^{(0)}) \ ,$$

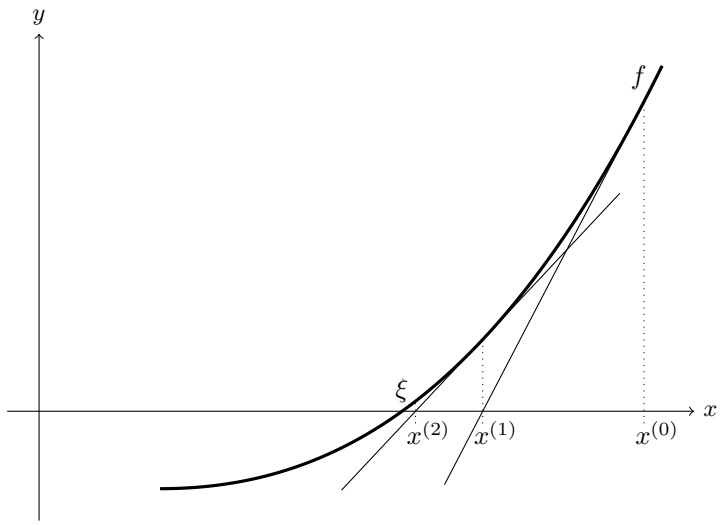


Abbildung 6: Graphische Deutung des Newton-Verfahrens: Die Tangente an f im Punkt $(x^{(0)}, f(x^{(0)}))$ schneidet die x-Achse in $x^{(1)}$. Der Wert $x^{(2)}$ im nächsten Schritt liegt schon nahe an der Nullstelle ξ.

aus der sich x ausdrücken lässt:

$$x = x^{(0)} - \frac{f(x^{(0)})}{f'(x^{(0)})} \ .$$

Newton-Verfahren
Gegeben eine differenzierbare Funktion f und ein Startwert $x^{(0)}$.
Gesucht eine Nullstelle von f.

Iterationsvorschrift

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})} \qquad \text{für } k = 0, 1, 2 \ldots$$

Quadratische Konvergenz

Das Newton-Verfahren zeigt *quadratische* Konvergenz. Das heißt, für die Fehlerschranken $\epsilon_{k+1} = |x^{(k+1)} - x|$ und $\epsilon_k = |x^{(k)} - x|$ aufeinanderfolgender Schritte gilt, sofern ϵ_k schon hinreichend klein ist:

$$\epsilon_{k+1} \leq C\epsilon_k^2$$

Der neue Fehler ist also um einen Faktor C kleiner als das *Quadrat* des alten Fehlers. Der genaue Wert von C ist dabei nicht so wichtig.

Angenommen, es ist $\epsilon_k = 1 \times 10^{-4}$. Das heißt, der Fehler beträgt eine Einheit in der vierten Nachkommastelle. Dann gilt bei quadratischer Konvergenz $\epsilon_{k+1} = C \cdot 1 \times 10^{-8}$. Der Fehler beträgt also C Einheiten in der achten Nachkommastelle. Wenn C größenordnungsmäßig im Bereich 1 ist, hat sich die Anzahl der korrekten Stellen ungefähr verdoppelt.

Quadratische Konvergenz: Neuer Fehler $\sim$ Quadrat des alten Fehlers.

Faustregel: Sofern schon einige signifikante Stellen exakt sind, sind im nächsten Näherungswert etwa doppelt so viele signifikante Stellen korrekt.

1.11 Abbruchbedingungen

Rechner haben nur eine fixe Zahl von Binärstellen zur Verfügung, um Gleitkommazahlen zu speichern. Möglicherweise erreicht $f(x)$ für kein Gleitkomma-Argument x exakt den Wert Null. Wenn die Nullstelle ξ in der Gegend von 1 liegt, können Sie leicht eine Näherung x mit absolutem Fehler $|x - \xi| < 10^{-6}$ finden. Liegt die Nullstelle um $\xi \approx 10^{22}$, werden Sie einen absoluten Fehler dieser Güte nicht erreichen können. Eine übliche Wahl der Abbruchschranke ϵ ist $\varepsilon_m(|a| + |b|)/2$, wenn ε_m die Maschinengenauigkeit und a, b die ursprünglichen Intervallgrenzen sind. Wenn a, b und die Nullstelle selber nahe bei Null liegen, ist Vorsicht bei dieser Formel geboten. Die Abbruchschranke darf jedenfalls nicht kleiner als die kleinste positive Maschinenzahl sein (typischerweise um 10^{-38} für 4-Byte-Datentypen, 10^{-308} für 8-Byte-Datentypen).

Maschinengenauigkeit

Die Maschinengenauigkeit ε_m ist die kleinste positive Gleitkommazahl, die, zur Gleitkommazahl 1,0 addiert, eine von 1,0 verschiedene Summe ergibt (typischerweise um 10^{-7} für 4-Byte-Datentypen, 10^{-16} für 8-Byte-Datentypen).

1.12 Fixpunkt-Iteration

Im Abschnitt 1.5 haben wir bereits Fixpunkte von Funktionen durch wiederholtes Einsetzen bestimmt. Viele numerische Verfahren lassen sich als Spezialfälle einer Fixpunkt-Iteration betrachten. Aussagen über die Konvergenz von Fixpunkt-Iterationen sind deswegen von allgemeiner Bedeutung.

Fixpunkt-Iteration
Gegeben eine Funktion ϕ und ein Startwert $x^{(0)}$.
Gesucht ein Fixpunkt ξ von ϕ.

Iterationsvorschrift
$x^{(k+1)} = \phi(x^{(k)})$ für $k = 0, 1, 2 \ldots$

Fixpunkt-Iteration konvergiert für kontrahierende Abbildungen
Die Funktion ϕ besitze einen Fixpunkt ξ Sei ferner I ein offenes Intervall der Form $(\xi - r, \xi + r)$ um den Fixpunkt ξ, in dem ϕ als *kontrahierende Abbildung* wirkt, d. h.

$$|\phi(x) - \phi(y)| \leq C|x - y| \text{ gilt mit } C < 1 \text{ für alle } x, y \in I \ .$$

Dann konvergiert für alle $x^{(0)} \in I$ die Fixpunkt-Iteration $x^{(k+1)} = \phi(x^{(k)})$ mindestens linear gegen ξ.

Beweis: Zuerst zeigt man durch Induktion: $x^{(k)} \in I$ für alle $k = 0, 1, 2 \ldots$ Die Aussage ist laut Voraussetzung richtig für $k = 0$. Angenommen, es liegt bereits $x^{(k)} \in I$, also weniger als r von ξ entfernt: $|x^{(k)} - \xi| < r$. Dann können wir die Kontraktionsbedingung und Fixpunkt-Eigenschaft für $x^{(k)}$ und ξ anwenden und erhalten

$$|x^{(k+1)} - \xi| = |\phi(x^{(k)}) - \phi(\xi)| \leq C|x^{(k)} - \xi| < Cr \; .$$

Da $C < 1$, ist also auch

$$|x^{(k+1)} - \xi| < r \quad \text{und somit} \quad x^{(k+1)} \in I$$

Aus diesen Überlegungen folgt auch unmittelbar für die Fehler $\epsilon^{(k)} = |x^{(k)} - \xi|$ und $\epsilon^{(k+1)} = |x^{(k+1)} - \xi|$:

$$\epsilon^{(k+1)} \leq C\epsilon^{(k)} \leq C^k \epsilon_0, \text{ somit } \epsilon^{(k+1)} \to 0 \text{ für } k \to \infty \; .$$

So wie der Satz hier formuliert ist, setzt er die Existenz eines Fixpunktes voraus. Dadurch wird der Konvergenz-Beweis kurz und schmerzlos. Eine etwas allgemeinere Formulierung und ein technisch aufwändigerer Beweis zeigen, dass aus der Kontraktions-Eigenschaft auch schon die Existenz und Eindeutigkeit eines Fixpunktes folgen. Das ist der berühmte Fixpunktsatz von Banach.

Zusammenhang kontrahierende Abbildung–Steigung der Funktion

Die Eigenschaft $|\phi(x) - \phi(y)| \leq C|x - y|$ bedeutet für $C < 1$ anschaulich: Funktionswerte unterscheiden sich weniger als die Eingabewerte. Wie stark sich Funktionswerte im Verhältnis zu Eingabewerten ändern, ist (im Grenzwert für kleine Änderungen) durch die Steigung der Funktion bestimmt.

Ist ϕ in einer Umgebung von ξ stetig differenzierbar und $|\phi'(\xi)| < 1$, so ist in einer Umgebung von ξ die Kontraktionseigenschaft erfüllt: Wegen der Stetigkeit von ϕ' gibt es ein offenes Intervall I um ξ, in dem $|\phi'| \leq C < 1$ gilt. Für $x, y \in I$ gilt nach dem Mittelwertsatz der Differentialrechnung

$$\phi(x) - \phi(y) = (x - y)\phi'(\eta) \quad \text{für ein } \eta \in I \; .$$

Damit ist auch

$$|\phi(x) - \phi(y)| \leq C|x - y| \; , \quad C < 1$$

Eine Kurzfassung dieser Aussage:

Abbildung 7 illustriert das Konvergenzverhalten der Fixpunkt-Iteration für verschiedene ϕ.

1.13 Konvergenzordnung

Wir haben lineare, superlineare und quadratische Konvergenz bereits erwähnt. Hier fassen wir den Begriff der Konvergenzordnung genauer.

Konvergenzordnung
Sei ξ Fixpunkt von ϕ, und es gelte für alle Startwerte aus einem Intervall um ξ und die zugehörige Folge $\{x^{(k)}\}$ aus der Vorschrift $x^{(k+1)} = \phi(x^{(k)}), k = 0, 1, 2 \ldots$

$$|x^{(k+1)} - \xi| \leq C|x^{(k)} - \xi|^p$$

mit $p \geq 1$ und $C < 1$, falls $p = 1$.
Das Iterationsverfahren heißt dann ein Verfahren von mindestens p-ter Ordnung

Für das lokale Konvergenzverhalten einer Fixpunkt-Iteration ist der Wert der ersten Ableitung am Fixpunkt maßgeblich. Für $|\phi'(\xi)| < 1$ ist lineare Konvergenz gesichert; je kleiner der Betrag der Ableitung, desto schneller konvergiert das Verfahren, wobei $C \approx |\phi'(\xi)|$. Ganz besonders rasche, nämlich superlineare Konvergenz tritt auf, wenn $|\phi'(\xi)| = 0$.

Mit Hilfe der Taylorentwicklung lässt sich zeigen: Ist $\phi(x)$ in einer Umgebung von ξ genügend oft differenzierbar und

$$\phi'(\xi) = 0,\ \phi''(\xi) = 0, \ldots, \phi^{(p-1)}(\xi) = 0, \text{ und } \phi^{(p)}(\xi) \neq 0\ ,$$

dann liegt für $p = 2, 3, \ldots$ ein Verfahren p-ter Ordnung vor. Ein Verfahren erster Ordnung liegt vor, wenn zu $p = 1$ gilt: $|\phi'(\xi)| < 1$.

1.14 Konvergenz des Newton-Verfahrens

Das Newtonverfahren, angewandt auf die Funktion f, entspricht einem Fixpunkt-Verfahren für die Funktion ϕ,

$$\phi(x) = x - \frac{f(x)}{f'(x)}$$

Nun ist

$$\phi'(x) = \frac{f''(x)f(x)}{(f'(x))^2}\ ,$$

und da an einer einfachen Nullstelle $f(x) = 0, f'(x) \neq 0$ gilt, verschwindet $\phi'(x)$ dort. Man überzeugt sich leicht, dass $\phi''(x) \neq 0$ gilt, sofern $f''(x) \neq 0$. Daraus folgt die quadratische Konvergenz des Newtonverfahrens bei einfachen Nullstellen. Bei mehrfachen Nullstellen lässt sich lineare Konvergenz nachweisen.

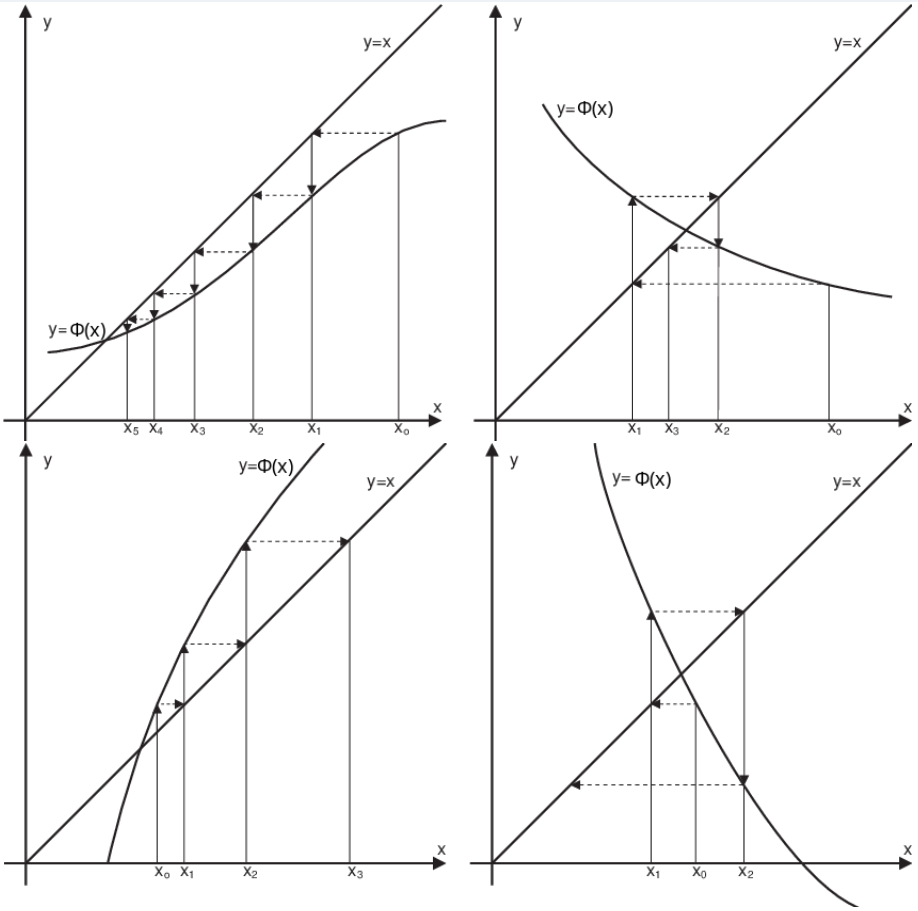


Abbildung 7: Fixpunkt-Iteration in graphischer Darstellung für verschiedene Funktionen ϕ. Mögliche Fälle: Einseitige Annäherung an den Fixpunkt, falls in einer Umgebung des Fixpunktes $0 < \phi' < 1$; alternierende Konvergenz, falls $-1 < \phi' < 0$, Divergenz falls $\phi' > 1$ oder $\phi' < -1$.

2 Systeme nichtlinearer Gleichungen

Abschnitt 1.2 definiert die Begriffe *Lösung, Nullstelle, Fixpunkt* für skalare Funktionen $\mathbb{R} \to \mathbb{R}$. Diese Begriffe lassen sich problemlos auf vektorwertige Funktionen $\mathbb{R}^n \to \mathbb{R}^n$ übertragen. Auch hier lassen sich Gleichungen auf verschiedene Weise formulieren.

Schreibweise für Vektoren und vektorwertige Funktionen: Fettdruck

Reellwertige Funktionen, Skalare: $f : \mathbb{R} \to \mathbb{R}$, $y = f(x)$
Vektorwertige Funktionen, Vektoren: $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$, $\mathbf{y} = \mathbf{f}(\mathbf{x})$

Komponenten eines Vektors $\mathbf{x} \in \mathbb{R}^n$:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad \text{oder } \mathbf{x}^T = [x_1, x_2, \ldots, x_n]$$

Normalerweise ist mit $\mathbf{x}$ ein Spalten-, mit $\mathbf{x}^T$ ein Zeilenvektor gemeint.

Iterationsindizes werden hier (um sie von Vektorkomponenten zu unterscheiden) hochgestellt und in Klammern gesetzt: $\mathbf{x}^{(k)}, k = 0, 1, 2 \ldots$

2.1 Lösung, Nullstelle und Fixpunkt: mehrdimensionaler Fall

Aufgabentypen im $\mathbb{R}^n$

Es seien $\mathbf{f}, \mathbf{g}, \mathbf{h}, \mathbf{\Phi}$ Funktionen $\mathbb{R}^n \to \mathbb{R}^n$ und $\mathbf{x} \in \mathbb{R}^n$

Problemstellung: gesucht ist ein $\mathbf{x}$, für das gilt...

$\mathbf{g}(\mathbf{x}) = \mathbf{h}(\mathbf{x})$, (Finden einer *Lösung* des Gleichungssystems)
$\mathbf{f}(\mathbf{x}) = 0$, (Finden einer *Nullstelle* der Funktion $\mathbf{f}$)
$\mathbf{x} = \mathbf{\Phi}(\mathbf{x})$, (Finden eines *Fixpunktes* der Funktion $\mathbf{\Phi}$)

Im Vergleich zu den Definitionen von Abschnitt 1.2 hat fast nichts geändert außer der Schreibweise.

Beispiel: ein *nichtlineares Gleichungssystem* mit zwei Unbekannten

$$\begin{aligned} 4x_1 - x_2 + x_1x_2 &= 1 \\ -x_1 + 6x_2 &= 2 - \log(x_1x_2) \end{aligned}$$

hat die Form $\mathbf{g}(\mathbf{x}) = \mathbf{h}(\mathbf{x})$ mit

$$\mathbf{g}(\mathbf{x}) = \begin{bmatrix} g_1(x_1, x_2) \\ g_2(x_1, x_2) \end{bmatrix} = \begin{bmatrix} 4x_1 - x_2 + x_1x_2 \\ -x_1 + 6x_2 \end{bmatrix}, \qquad \mathbf{h}(\mathbf{x}) = \begin{bmatrix} h_1(x_1, x_2) \\ h_2(x_1, x_2) \end{bmatrix} = \begin{bmatrix} 1 \\ 2 - \log(x_1x_2) \end{bmatrix}$$

Das Gleichungssystem lässt sich umformulieren:

$$\begin{aligned} 4x_1 - x_2 + x_1x_2 - 1 &= 0 \\ -x_1 + 6x_2 + \log(x_1x_2) - 2 &= 0 \end{aligned}$$

In dieser Form lautet die Aufgabe: gesucht sind *Nullstellen der vektorwertigen Funktion* $\mathbf{f} : \mathbb{R}^2 \to \mathbb{R}^2$, also Lösungen von $\mathbf{f}(\mathbf{x}) = 0$ mit

$$\mathbf{f}(\mathbf{x}) = \begin{bmatrix} f_1(x_1, x_2) \\ f_2(x_1, x_2) \end{bmatrix} = \begin{bmatrix} 4x_1 - x_2 + x_1x_2 - 1 \\ -x_1 + 6x_2 + \log(x_1x_2) - 2 \end{bmatrix}$$

Eine andere, äquivalente Umformung liefert

$$\begin{aligned} x_1 &= \frac{1}{4}(x_2 - x_1x_2 + 1) \\ x_2 &= \frac{1}{6}(x_1 - \log(x_1x_2) + 2) \end{aligned}$$

Hier sind *Fixpunkte der vektorwertigen Funktion* $\mathbf{\Phi} : \mathbb{R}^2 \to \mathbb{R}^2$ gesucht, also Lösungen von $\mathbf{x} = \mathbf{\Phi}(\mathbf{x})$ mit

$$\mathbf{\Phi}(\mathbf{x}) = \begin{bmatrix} \phi_1(x_1, x_2) \\ \phi_2(x_1, x_2) \end{bmatrix} = \begin{bmatrix} \frac{1}{4}(x_2 - x_1x_2 + 1) \\ \frac{1}{6}(x_1 - \log(x_1x_2) + 2) \end{bmatrix}$$

Noch ein Hinweis zur Schreibweise: Wenn wir einen speziellen Fixpunkt gefunden haben, dann bezeichnen wir den im Folgenden mit $\boldsymbol{\xi}$, um ihn von anderen, allgemeinen Werten $\mathbf{x}$ zu unterscheiden.

2.2 Mehrdimensionale Fixpunkt-Iteration

Fixpunkt-Iterationen sind auch im mehrdimensionalen Fall möglich. Ein Fixpunkt einer Abbildung $\mathbf{\Phi} : \mathbb{R}^n \to \mathbb{R}^n$ ist – völlig analog zur eindimensionalen Definition – ein Wert $\boldsymbol{\xi} \in \mathbb{R}^n$, für den gilt:

$$\boldsymbol{\xi} = \mathbf{\Phi}(\boldsymbol{\xi}) \, .$$

Genauso wie im eindimensionalen Fall findet Fixpunkt-Iteration (falls sie konvergiert) einen Fixpunkt. Noch einmal: Wir setzen hier Vektoren aus dem $\mathbb{R}^n$ und vektorwertige Funktionen in fetter Schrift ($\mathbf{\Phi}, \boldsymbol{\xi}, \mathbf{x} \ldots$), zum Unterschied von Variablen und reellwertigen Funktionen ($\phi, \xi, x, \ldots$). Sonst ändert sich nichts am Schema der Fixpunkt-Iteration.

Fixpunkt-Iteration, mehrdimensional
Gegeben sei eine Abbildung $\mathbf{\Phi} : \mathbb{R}^n \to \mathbb{R}^n, \quad \mathbf{x} \to \mathbf{\Phi}(\mathbf{x})$.
Gesucht ist ein Fixpunkt $\boldsymbol{\xi}$ von $\mathbf{\Phi}$.

$\mathbf{x}^{(0)}$ als Startwert gegeben.
Iterationsvorschrift
$\mathbf{x}^{(k+1)} = \mathbf{\Phi}(\mathbf{x}^{(k)})$ für $k = 0, 1, 2 \ldots$

Beachte die Konvergenzbedingungen (Abschnitt 2.4)!

Beispiel: Fixpunkt-Iteration für ein System zweier nichtlinearer Gleichungen

Gegeben sei das nichtlineare Gleichungssystem (log ist natürlich der natürliche Logarithmus)

$$\begin{aligned} 4x - y + xy - 1 &= 0 \\ -x + 6y + \log(xy) - 2 &= 0 \end{aligned}$$

Ausgehend von der Näherungslösung $x_0 = 1$ und $y_0 = 1$ bestimme man durch geeignete Fixpunkt-Iteration verbesserten Näherungen.

In der Nähe des Startwertes hängt die erste Gleichung am stärksten vom Term $4x$ ab; die zweite Gleichung von $6y$. Vorgangsweise: löse die beiden Gleichungen jeweils nach diesen Termen auf.

$$\begin{aligned} x &= \frac{1}{4}(y - xy + 1) \\ y &= \frac{1}{6}(x - \log(xy) + 2) \end{aligned}$$

Die Funktion $\mathbf{\Phi}$ ist hier ein Vektor aus zwei reellwertigen Funktionen ϕ und ψ, der Vektor $\mathbf{x}$ hat zwei Komponenten x und y.

$$\mathbf{\Phi}(\mathbf{x}) = \begin{bmatrix} \phi(x,y) \\ \psi(x,y) \end{bmatrix} = \begin{bmatrix} \frac{1}{4}(y - xy + 1) \\ \frac{1}{6}(x - \log(xy) + 2) \end{bmatrix}$$

Iteration liefert die Folge $(1;1)$, $(1/4;1/2)$, $(0{,}343\,75; 0{,}721\,574)$, $(0{,}368\,383; 0{,}622\,985)$, …, die gegen den Fixpunkt $(0{,}353\,443\,88; 0{,}639\,968\,47)$ konvergiert.

2.3 Normen

Exakte Lösung, Näherungslösung und Fehler sind bei Gleichungssystemen jeweils Vektoren im $\mathbb{R}^n$. Wir brauchen ein Maß für die Größe oder Länge des Fehlervektors, oder für den Abstand der Näherung von der exakten Lösung. Im eindimensionalen Fall messen wir die „Größe“ von x mit dem Absolutbetrag $|x|$, und den Abstand zweier Werte x und y auf der reellen Achse durch $|y - x|$.

Während es aber in $\mathbb{R}$ nur eine sinnvolle Definition für den Absolutbetrag gibt, stehen im $\mathbb{R}^n$ mehrere Möglichkeiten offen. Da ist zunächst einmal die „übliche“ Definition für die Länge eines Vektors, auch *euklidische* Länge oder *2-Norm* genannt. Oft lässt sich aber mit anderen Normen einfacher arbeiten. Wir verwenden noch die *1-Norm* und die *∞-Norm*.

Normen im $\mathbb{R}^n$ für einen Vektor $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$

$$\|\mathbf{x}\|_1 = \sum_{i=1}^{n} |x_i| \quad , \quad \text{Einsnorm, Summennorm}$$

$$\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^{n} (x_i)^2} \quad , \quad \text{Zweinorm, euklidische Norm}$$

$$\|\mathbf{x}\|_\infty = \max_i |x_i| \quad , \quad \text{Unendlich-Norm, Maximums-Norm}$$

Erinnern Sie sich an die Definition einer Norm aus Mathematik 2?

Eine Norm im $\mathbb{R}^n$ ist eine Funktion, die jedem Vektor $\mathbf{x} \in \mathbb{R}^n$ eine nichtnegative reelle Zahl $\|\mathbf{x}\| \in \mathbb{R}_0^+$ zuordnet, wobei $\forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, $\forall \alpha \in \mathbb{R}$ drei Bedingungen gelten müssen:

- Nur der Nullvektor hat Norm 0

$$\|\mathbf{x}\| = 0 \quad \Rightarrow \quad \mathbf{x} = 0$$

- Skalar α läst sich als Betrag herausheben

$$\|\alpha \cdot \mathbf{x}\| = |\alpha| \cdot \|\mathbf{x}\|$$

- Die Dreiecksungleichung gilt

$$\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|$$

Norm und Distanz

Eine Norm kann auch die *Distanz* zwischen zwei Punkten $\mathbf{x}$ und $\mathbf{y}$ messen:

$$\text{dist}(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|$$

- Taxis in Manhattan messen Strecken in der 1-Norm.

  Deswegen heißt die 1-Norm auch Taxi- oder Manhattan-Norm (engl. *taxicab norm*)
- Abstand in der Luftlinie entspricht der 2-Norm.
- Größter Unterschied in den Komponenten: ∞-Norm.

Matrixnormen

- Der Hauptberuf von Matrizen ist, Vektoren zu multiplizieren.
- Das Ergebnis einer Matrix-Vektor-Multiplikation ist wieder ein Vektor; der ist gewöhnlich länger oder kürzer und verdreht gegenüber Ausgangsvektor.
- Eine *Matrixnorm* misst als „Größe“ einer Matrix, wie stark sie die Länge von Vektoren ändern kann.
- Eine gegebene Matrix kann Vektoren nicht beliebig stark verlängern. Es gibt für jede Matrix einen „Maximal-Verlängerungs-Faktor“

Der „Maximal-Verlängerungs-Faktor“ ist eine Matrixnorm

Verschiedene Matrixnormen

Die 1-, 2- und ∞-Normen lassen sich von den entsprechenden Vektornormen ableiten: Sie geben für die Rechenoperation $\mathbf{y} = A \cdot \mathbf{x}$ an, um wieviel $\mathbf{y}$ gegenüber $\mathbf{x}$ *maximal vergrößert* wird. Eins- und ∞-Norm lassen sich aus den Matrixelementen einfach berechnen:

$\|A\|_1$	*Einsnorm*: maximale Spaltenbetragssumme
$\|A\|_\infty$	*Unendlich-Norm*: maximale Zeilenbetragssumme

Für die Matrix-Zweinorm lässt sich keine so einfache Rechenvorschrift angeben, obwohl gerade sie häufig verwendet wird.

MATLAB kann alle Normen leicht berechnen: $\|A\|_1$ = `norm(A,1)`, $\|A\|_2$ = `norm(A)`, $\|A\|_\infty$ = `norm(A,Inf)`.

Matrixnorm, allgemeine Definition

Matrizen lassen sich addieren und mit Skalaren multiplizieren. In diesem Sinn verhalten sie sich genauso wie Vektoren des $\mathbb{R}^n$. Alles, was sich wie ein Vektor verhält, können wir als „Vektor“ interpretieren: Die $m \times n$-Matrizen bilden einen *Vektorraum*. Der Begriff „Norm“ wird genau so definiert wie die Norm von Vektoren des $\mathbb{R}^n$. Vergleichen Sie die Definition einer Norm im $\mathbb{R}^n$ auf Seite 23 – sie wird hier nahezu wörtlich übernommen.

Eine *Norm* im $\mathbb{R}^m \times \mathbb{R}^n$ ist eine Funktion, die jeder $m \times n$-Matrix A eine nichtnegative reelle Zahl $\|A\| \in \mathbb{R}_0^+$ zuordnet, wobei $\forall A, B \in \mathbb{R}^m \times \mathbb{R}^n$, $\forall \alpha \in \mathbb{R}$ drei Bedingungen gelten müssen:

- Nur die Nullmatrix hat Norm 0:

$$\|A\| = 0 \quad \Rightarrow \quad A = 0$$

- Skalar α läst sich als Betrag herausheben:

$$\|\alpha \cdot A\| = |\alpha| \cdot \|A\|$$

- Die Dreiecksungleichung gilt:

$$\|A + B\| \leq \|A\| + \|B\|$$

Diese drei Grundregeln muss jede Norm erfüllen. Aber es gibt für die 1-, 2- oder ∞-Norm noch Bonus-Features. Für diese Matrix-Normen gelten nämlich noch folgende Rechenregeln:

$$\|A \cdot B\| \leq \|A\| \cdot \|B\| \tag{8}$$

$$\|A \cdot \mathbf{x}\| \leq \|A\| \cdot \|\mathbf{x}\| \tag{9}$$

Vergleiche Absolutbetrag: $|a \cdot b| = |a| \cdot |b|$

Frobeniusnorm:

Noch eine weitere Norm; Die Frobenius-Norm $\|A\|_F$ wird so ähnlich berechnet wie die Vektor-Zweinorm: *Quadrieren, summieren, Wurzel ziehen*

Frobenius-Norm: $\qquad \|A\|_F = \sqrt{\sum a_{ij}^2}$

Die Frobeniusnorm lässt sich leichter berechnen als die Matrix-Zweinorm und dient zu deren Abschätzung:

$$\|A\|_2 \leq \|A\|_F$$

Auch für $\|A\|_F$ gelten neben den Norm-Axiome noch die Rechenregeln

$$\|A \cdot B\|_F \leq \|A\|_F \cdot \|B\|_F \quad , \quad \|A \cdot \mathbf{x}\|_2 \leq \|A\|_F \|\mathbf{x}\|_2$$

MATLAB: $\|A\|_F =$ `norm(A,'fro')`.

Matrixnormen – das Kleingedruckte[6]

[6] Was hier dasteht, ist nicht wichtig,
wenn 's nicht dastünd', wär's nicht richtig.

Die lockere Erklärung *„Matrixnorm ist maximaler Verlängerungsfaktor“* ist mathematisch korrekt für 1-, 2- und ∞-Norm, wenn Vektorlängen in den jeweiligen Normen gemessen werden. Die Frobeniusnorm überschätzt aber gewöhnlich den maximal auftretenden Verlängerungsfaktor, wenn Vektorlängen in der 2-Norm gemessen werden. Immerhin liefert sie eine obere Schranke für den Verlängerungsfaktor.
Auch die Vorschrift $\|A\| = \max_{i,j} |a_{ij}|$ erfüllt die drei Bedingungen einer Norm, ist aber nicht immer eine obere Schranke für den Verlängerungsfaktor.

2.4 Konvergenz

Die Konvergenz der mehrdimensionalen Fixpunkt-Iteration hängt wie im eindimensionalen Fall mit dem Begriff der kontrahierenden Abbildung zusammen.

Konvergenz der Fixpunkt-Iteration im $\mathbb{R}^n$
Die Funktion $\mathbf{\Phi}(x)$ besitze einen Fixpunkt $\boldsymbol{\xi}$: $\mathbf{\Phi}(\boldsymbol{\xi}) = \boldsymbol{\xi}$. Sei ferner B eine offene Umgebung um den Fixpunkt in der Form $B = \{\mathbf{x} : \|\boldsymbol{\xi} - \mathbf{x}\| < r\}, \quad r > 0$. Wenn $\mathbf{\Phi}$ in B eine *kontrahierende Abbildung* in (irgend-) einer Norm $\|\cdot\|$ ist, d. h.,

$$\|\mathbf{\Phi}(\mathbf{x}) - \mathbf{\Phi}(\mathbf{y})\| \leq C\|\mathbf{x} - \mathbf{y}\| \,, \quad C < 1 \text{ für alle } \mathbf{x}, \mathbf{y} \in B \,,$$

dann konvergiert die Fixpunkt-Iteration $\mathbf{x}^{(k+1)} = \mathbf{\Phi}(\mathbf{x}^{(k)})$ mindestens linear gegen $\boldsymbol{\xi}$ für alle $\mathbf{x}^{(0)} \in B$.

Der Beweis erfolgt analog zu der eindimensionalen Form des Konvergenzsatzes. Auch der Begriff der Konvergenzordnung läßt sich unter Verwendung von Normen geradewegs auf den mehrdimensionalen Fall übertragen.

Kontraktion und Jacobi-Matrix

Das Konvergenzkriterium $|\phi'(\xi)| < 1$ im eindimensionalen Fall (vergleiche Seite 17) lässt sich auf den mehrdimensionalen Fall übertragen. Dazu fasst man die partiellen Ableitungen von $\mathbf{\Phi}$ in einer Matrix D_ϕ, genannt die *Jacobi -Matrix*, zusammen.

Jacobi-Matrix D_ϕ einer Funktion $\mathbf{\Phi} : \mathbb{R}^n \to \mathbb{R}^n$

$$D_\phi = \begin{bmatrix} \frac{\partial \phi_1}{\partial x_1} & \frac{\partial \phi_1}{\partial x_2} & \cdots & \frac{\partial \phi_1}{\partial x_n} \\ \frac{\partial \phi_2}{\partial x_1} & \frac{\partial \phi_2}{\partial x_2} & \cdots & \frac{\partial \phi_2}{\partial x_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial \phi_n}{\partial x_1} & \frac{\partial \phi_n}{\partial x_2} & \cdots & \frac{\partial \phi_n}{\partial x_n} \end{bmatrix}$$

Dann lässt sich ganz ähnlich wie im eindimensionalen Fall aussagen:

Das Fixpunktverfahren konvergiert lokal,
falls in der 1-,2-, Frobenius- oder ∞-Matrixnorm gilt

$$\|D_\phi\| < 1$$

2.5 Newton-Verfahren für Systeme

Gegeben sei eine vektorwertige Funktion $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$. Gesucht sei eine Nullstelle von $\mathbf{f}$. Das ist ein Vektor $\mathbf{x} \in \mathbb{R}^n$ als Lösung von

$$\mathbf{f}(\mathbf{x}) = \mathbf{0}$$

Dies ist die allgemeine Formulierung eines Systems von n linearen oder nichtlinearen Gleichungen in n Unbekannten. Und noch einmal sei darauf hingewiesen: wir setzen Vektoren aus dem $\mathbb{R}^n$ und vektorwertige Funktionen in fetter Schrift $(\mathbf{x}, \mathbf{f}(\mathbf{x}), \ldots)$, zum Unterschied von Variablen und reellwertigen Funktionen $(x, f(x), \ldots)$.

Komponentenweise ausgeschrieben mit

$$\mathbf{f} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{bmatrix} \quad \text{und} \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad \text{lautet das System} \quad \begin{array}{l} f_1(x_1, x_2, \ \ldots \ , x_n) = 0 \\ f_2(x_1, x_2, \ \ldots \ , x_n) = 0 \\ \qquad \ldots \\ f_n(x_1, x_2, \ \ldots \ , x_n) = 0 \end{array} .$$

Das Newton-Verfahren für Systeme führt die Lösung eines nichtlinearen Systems auf die Lösung einer Folge von linearen Gleichungssystemen zurück. Die Lösung von Systemen linearer Gleichungen ist vergleichsweise einfach gegenüber nichtlinearen Gleichungssystemen. Wir behandeln lineare Gleichungssysteme später noch ausführlich, aber einstweilen nehmen wir an, dass Sie aus der Mittelschule damit hinreichend vertraut sind.

Sofern die entsprechenden partiellen Ableitungen existieren, definieren wir die *Jacobi-Matrix* D_f von $\mathbf{f}$ durch

$$D_f = \begin{bmatrix} \dfrac{\partial f_1}{\partial x_1} & \dfrac{\partial f_1}{\partial x_2} & \cdots & \dfrac{\partial f_1}{\partial x_n} \\ \dfrac{\partial f_2}{\partial x_1} & \dfrac{\partial f_2}{\partial x_2} & \cdots & \dfrac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & & \vdots \\ \dfrac{\partial f_n}{\partial x_1} & \dfrac{\partial f_n}{\partial x_2} & \cdots & \dfrac{\partial f_n}{\partial x_n} \end{bmatrix}$$

Angenommen, ein Punkt $\mathbf{x}^{(0)}$ ist als Startwert in der Nähe einer Nullstelle gegeben. Dann lässt sich $\mathbf{f}$ in der Umgebung von $\mathbf{x}^{(0)}$ in linearisierter Näherung schreiben als (Taylorscher Lehrsatz für Funktionen mehrerer Veränderlicher)

$$\mathbf{f}(\mathbf{x}^{(0)} + \Delta\mathbf{x}) = \mathbf{f}(\mathbf{x}^{(0)}) + D_f(\mathbf{x}^{(0)}) \cdot \Delta\mathbf{x} + \mathbf{R}$$

mit einem Restglied $\mathbf{R}$, das im Limes $\Delta\mathbf{x} \to 0$ mit höherer Ordnung verschwindet. Wir vernachlässigen das Restglied und fordern $\mathbf{f}(\mathbf{x}^{(0)} + \Delta\mathbf{x}) = 0$. Aus der daraus entstandenen Gleichung

$$0 = \mathbf{f}(\mathbf{x}^{(0)}) + D_f(\mathbf{x}^{(0)}) \cdot \Delta\mathbf{x}$$

lässt sich der Korrekturvektor $\Delta\mathbf{x}$ bestimmen und damit eine verbesserte Näherung $\mathbf{x}^{(1)} = \mathbf{x}^{(0)} + \Delta\mathbf{x}$.

Das Newton-Verfahrens für Systeme lässt sich so formulieren:

Newton-Verfahren für Systeme
Gegeben eine differenzierbare vektorwertige Funktion $\mathbf{f}$ und ein Startwert $\mathbf{x}^{(0)}$. Gesucht eine Nullstelle von $\mathbf{f}$.

Iterationsvorschrift

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \Delta\mathbf{x}^{(k)}$$

mit $\Delta\mathbf{x}^{(k)}$ als Lösung von $D_f(\mathbf{x}^{(k)})\Delta\mathbf{x}^{(k)} = -\mathbf{f}(\mathbf{x}^{(k)})$

Auch dieses Verfahren ist ein Fixpunktverfahren, und zwar für die Funktion

$$\mathbf{\Phi}(\mathbf{x}) = \mathbf{x} - D_f^{-1}(\mathbf{x})\mathbf{f}(\mathbf{x}).$$

Notwendig für die Durchführbarkeit ist, dass D_f^{-1} existiert.

Man kann zeigen: Sofern D_f^{-1} an der Nullstelle existiert, konvergiert das Verfahren für genügend genaue Startwerte quadratisch.

Da es oft sehr mühsam ist, immer alle Elemente von D_f an jedem Punkt $\mathbf{x}^{(k)}$ zu berechnen, geht man manchmal so vor, daß man D_f an einem einzigen Punkt $\mathbf{x}^{(0)}$ berechnet und für den weiteren Verlauf des Verfahrens fix lässt. Dieses Verfahren heißt vereinfachtes Newton-Verfahren. Dafür muss $x^{(0)}$ bereits eine brauchbare Näherung sein. Das vereinfachte Newton-Verfahren konvergiert allerdings nur linear.

Das Newton-Verfahren für Systeme erfordert also in jedem Schritt die Lösung eines linearen Gleichungssystems. Das nächste Kapitel bringt die systematische Behandlung linearer Gleichungssysteme.

Beispiel: nichtlineares Gleichungssystem aus Abschnitt 2.2

Die Funktion $\mathbf{f}$ und ihre Jacobi-Matrix D_f sind hier

$$\mathbf{f}(\mathbf{x}) = \begin{bmatrix} 4x - y + xy - 1 \\ -x + 6y + \log(xy) - 2 \end{bmatrix}, \qquad D_f = \begin{bmatrix} 4+y & -1+x \\ -1+\frac{1}{x} & 6+\frac{1}{y} \end{bmatrix}.$$

Startwert $(1;1)$ eingesetzt liefert

$$\mathbf{f}(\mathbf{x}) = \begin{bmatrix} 3 \\ 3 \end{bmatrix}, \qquad D_f = \begin{bmatrix} 5 & 0 \\ 0 & 7 \end{bmatrix}.$$

Zu lösen ist also das Gleichungssystem

$$\begin{bmatrix} 5 & 0 \\ 0 & 7 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = - \begin{bmatrix} 3 \\ 3 \end{bmatrix}$$

Es liefert den Korrekturvektor und die verbesserte Lösung

$$\Delta\mathbf{x}^{(0)} = \begin{bmatrix} -0,6 \\ -0,428571 \end{bmatrix}, \quad \mathbf{x}^{(1)} = \mathbf{x}^{(0)} + \Delta\mathbf{x}^{(0)} = \begin{bmatrix} 0,4 \\ 0,571429 \end{bmatrix}.$$

Der nächste Schritt wertet zuerst $\mathbf{f}$ und D_f für die neuen Werte von $\mathbf{x}$, löst das Gleichungssystem für den Korrekturterm $\Delta\mathbf{x}^{(1)}$ und errechnet daraus die verbesserte Näherung $\mathbf{x}^{(2)} = \mathbf{x}^{(1)} + \Delta\mathbf{x}^{(1)}$. Die Matrix D_f hat aber hier nicht mehr so „schöne“ Einträge; das Gleichungssystem ist deswegen nicht so unmittelbar lösbar wie im ersten Schritt. Das vereinfachte Newtonverfahren würde zwar $\mathbf{f}$ neu auswerten, die Matrix D_f des ersten Schrittes beibehalten. Einfacherer Rechengang, aber langsamere (nur lineare statt quadratischer) Konvergenz!

3 Lineare Gleichungssysteme, direkte Verfahren

Wir verwenden die Standard-Notation für ein System linearer Gleichungen:

$$A\mathbf{x} = \mathbf{b} \,,$$

worin A die Koeffizientenmatrix, $\mathbf{b}$ die rechte Seite und $\mathbf{x}$ den Lösungsvektor bezeichnet. Wenn das System aus n Gleichungen und Unbekannten besteht, dann ist A eine $n \times n$-Matrix.

Lösungsverfahren für lineare Gleichungssysteme lassen sich in zwei Hauptgruppen unterteilen: direkte und iterative Verfahren.

- Direkte Verfahren berechnen (rundungsfehlerfreie Rechnung vorausgesetzt) eine exakte Lösung. Eliminations- und Substitutionsverfahren sowie die Cramersche Regel fallen in diese Kategorie. Wenn Sie mit Papier und Stift Systeme mit zwei oder drei Unbekannten lösen wollen, sind solche direkte Verfahren die Methoden der Wahl. Computer können heutzutage problemlos für mehrere zehntausend Gleichungen und Unbekannten direkte Lösungsmethoden verwenden.
- Iterative Verfahren berechnen schrittweise immer bessere Näherungslösungen. Diese Methoden eigenen sich aber nur für Gleichungssysteme mit spezieller Matrix-Struktur. Computer lösen damit riesig große Gleichungssysteme (mehrere Millionen Unbekannte), wie sie zum Beispiel bei numerischer Strömungssimulation oder Festigkeitsberechnungen auftreten.

Dieses Kapitel behandelt direkte Verfahren und wiederholt (was aus Mathematik 1 bekannt sein sollte) theoretische Aussagen zur Existenz und Eindeutigkeit der Lösung; iterative Methoden behandelt Kapitel 4.

Software in anerkannt hoher Qualität ist in der Programmbibliothek LAPACK (`http://www.netlib.org/lapack/`) frei verfügbar. Sie werden auch in kommerziell angebotenen Paketen nichts Besseres finden. Auch MATLAB enthält die LAPACK-Algorithmen. (Übrigens ist MATLAB ursprünglich als einfache Benutzerschnittstelle zu diesem Programmpaket entstanden).

3.1 Dreiecksmatrizen

Wenn A eine untere oder obere *Dreiecksmatrix* ist, kann man das Gleichungssystem $A\mathbf{x} = \mathbf{b}$ einfach durch schrittweises Einsetzen lösen (Vorwärts- oder Rückwärtssubstitution).

Andernfalls transformiert man das Gleichungssystem auf Dreiecksgestalt, wie es Abschnitt 3.2 beschreibt. Andere Möglichkeit: man *faktorisiert* A zuerst in ein Produkt von Dreiecksmatrizen. Das entsprechende Verfahren beschreibt Abschnitt 3.5.

Wir bezeichnen Dreiecksmatrizen mit L und R. In der üblichen Notation sind in L nur Einträge im linken unteren Dreieck von Null verschieden und alle Einträge der Hauptdiagonale gleich eins. In R sind nur Einträge im rechten oberen Dreieck einschließlich der Hauptdiagonale ungleich Null. Beispiel für $n = 4$:

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ \ell_{21} & 1 & 0 & 0 \\ \ell_{31} & \ell_{32} & 1 & 0 \\ \ell_{41} & \ell_{42} & \ell_{43} & 1 \end{bmatrix}, \qquad R = \begin{bmatrix} r_{11} & r_{12} & r_{13} & r_{14} \\ 0 & r_{22} & r_{23} & r_{24} \\ 0 & 0 & r_{33} & r_{34} \\ 0 & 0 & 0 & r_{44} \end{bmatrix}$$

Wenn die Elemente einer Dreiecksmatrix L auf einem Feld `a[i][j]` und die rechte Seite $\mathbf{b}$ auf einem Vektor `b[i]` gespeichert sind, löst das folgende Java-Programmsegment das Gleichungssystem $L\mathbf{x} = \mathbf{b}$ schrittweise durch *Vorwärts-Substitution*.

Achtung, Java zählt Feldindizes von 0 bis $n-1$; konventionelle Mathematik-Schreibweise zählt Zeilen und Spalten von 1 bis n.)

```
for (int i=0; i<n; i++) {
   x[i] = b[i];
   for (int j=0; j<i; j++) {
      x[i] -= a[i][j] * x[j];
   }
}
```

Ähnlich kompakt lässt sich die Lösung eines Gleichungssystems mit rechter oberer Dreiecksmatrixformulieren. Unterschiede zu vorhin: Die *Rückwärts-Substitution* beginnt in der letzten Zeile und schreitet von unten nach oben fort; durch das Hauptdiagonalelement wird dividiert; weiters ist hier die rechte Seite anfangs bereits auf `x[ ]` gespeichert, der Algorithmus überschreibt `x[ ]` mit dem Lösungsvektor.

```
for (int i=n-1; i>-1; i--) {
   for (int j=i+1; j<n; j++) {
      x[i] -= a[i][j] * x[j];
   }
   x[i] /= a[i][i];
}
```

Der Rechenaufwand, gemessen an der Zahl der Punktoperationen (Multiplikationen und Divisionen) beträgt für die Vorwärts-Substitution $n^2/2 - n/2$. Die Rückwärts-Substitution braucht $n^2/2 + n/2$ Punktoperationen. Für große n ist allein der quadratische Term ausschlaggebend. Wir sagen daher

Rechenaufwand bei Vorwärts- oder Rückwärts-Substitution
Lösen eines $n \times n$-Dreieckssystem erfordert $O(n^2)$ Punktoperationen.

Der Rechenaufwand wächst also quadratisch mit der Anzahl der Gleichungen.

Frage: Was tut man, wenn ein Gleichungssystem nicht in Dreiecksform vorliegt? Antwort: man formt es in ein solches um (natürlich so, dass Originalsystem und umgeformtes System äquivalent sind, also genau die gleichen Lösungen haben).

Darum geht es im nächsten Abschnitt.

3.2 Gauß-Elimination

Die einfache Gauß-Elimination läßt sich so formulieren:

Einfache Gauß-Elimination
Gegeben eine $n \times n$-Matrix A und rechte Seite $\mathbf{b}$. Sofern keines der a_{kk} zu einer Division durch Null führt, transformiert dieses Verfahren das System $A\mathbf{x} = \mathbf{b}$ auf ein äquivalentes System in oberer Dreiecksform $R\mathbf{x} = \mathbf{c}$.

Für alle Spalten $k = 1, \ldots n-1$
in Spalte k eliminiere alle Einträge unterhalb des Diagonalelements

Das Eliminieren in Spalte k läuft dabei folgendermaßen ab:

Für alle Zeilen $i = k+1, \ldots, n$ unterhalb der Diagonale
setze $p = a_{ik}/a_{kk}$
subtrahiere das p-fache der k-ten Zeile von Zeile i

Die Subtraktion wird durch folgende Schleife bewerkstelligt:

Für alle Spalten $j = k, \ldots n$
$a_{ij} = a_{ij} - pa_{kj}$
Für rechte Seite: $b_i = b_i - pb_k$

Diese Rechenvorschrift *überschreibt* Einträge in A und $\mathbf{b}$ mit den jeweiligen Zwischenresultaten und letztlich mit den Einträgen von R und $\mathbf{c}$. Sie verzichtet darauf, Einträge unterhalb der Diagonale (die eigentlich gleich Null sein sollen) zu löschen. Es wird sich später, in Abschnitt 3.5, herausstellen, dass diese Einträge eine wichtige Bedeutung haben.

Der Rechenaufwand beträgt $n^3/3 - n/3$ Punktoperationen für die Transformation der Matrix und $n^2/2 - n/2$ Punktoperationen für die Transformation der rechten Seite.

Als JAVA Code sieht Gauß-Elimination verblüffend einfach aus. Zu Beginn muss in `x[ ]` die rechte Seite gespeichert sein. In drei geschachtelte Schleifen wird schließlich die Lösung auf `x[ ]` geschrieben.

```
for (int k=0; k<n; k++) {
   for (int i=k+1; i<n; i++) {
      double p = a[i][k] / a[k][k];
      for (int j=k+1; j<n; j++) {
         a[i][j] -= p * a[k][j];
      }
      x[i] -= p * x[k];
   }
}
```

Das Programm spart es sich, im k-ten Schritt die Elemente unterhalb der Hauptdiagonale in der k-ten Spalte zu berechnen, weil ohnehin 0 herauskommen muss. Es verzichtet auch darauf, diese Nullen explizit in die Matrix hineinzuschreiben, sondern lässt dort die Zwischenresultate einfach stehen.

Das schrittweise Rückwärts-Einsetzen läßt sich mit $n^2/2 + O(n)$ Punktoperationen genäß dem Programmsegment aus dem vorigen Abschnitt erledigen. (Diese Doppelschleife verwendet nur das obere Dreieck von A; die Zwischenresultate $\neq 0$ unterhalb der Diagonale von vorhin stören daher nicht.)

Kombiniert liefern diese beiden Codesegmente einen einfachen Gleichungslöser.

Rechenaufwand bei einfacher Gauss-Elimination wächst kubisch mit der Anzahl der Gleichungen.
Gauß-Elimination löst ein $n \times n$-System $A\mathbf{x} = \mathbf{b}$ mit $O(n^3)$ Rechenoperationen.

(Genau nachgezählt sind es $n^3/3 + n^2 - n/3 = n^3/3 + O(n^2)$ Punktoperationen.)

3.3 Pivotsuche

Die einfache Gauß-Elimination hat einen Haken: Der Rechenschritt $p = a_{ik}/a_{kk}$ kann zu einer Division durch Null führen. Für eine Matrix zufällig gewählter reeller Zahlen ist das extrem unwahrscheinlich, aber Murphy's Gesetz besagt: *If anything can go wrong, it will.* Und tatsächlich versagt das Verfahren bei so simplen Systemen wie

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix},$$

weil gleich als erste Rechenoperation durch Null dividiert wird. Das ist definitiv nicht Schuld des Gleichungssystems, es hat nämlich die eindeutige Lösung $x_1 = 1, x_2 = 1$. Vertauscht man andererseits erste und zweite Gleichung, dann läuft das Verfahren problemlos.

Auch aus Gründen der Rechengenauigkeit kann es günstig sein, Gleichungen oder Unbekannte systematisch zu vertauschen. Man nennt diese Vorgehensweise *Pivotierung*.

Gauß-Elimination mit vollständiger Pivotsuche
Gegeben eine $n \times n$-Matrix A und rechte Seite $\mathbf{b}$. Dieses Verfahren transformiert das System $A\mathbf{x} = \mathbf{b}$ auf obere Dreiecksform $R\mathbf{x} = \mathbf{c}$. Die Matrix A wird dabei durch R und der Vektor $\mathbf{b}$ durch $\mathbf{c}$ überschrieben.

```
Für alle Spalten k = 1,...,n-1
    suche das betragsgrößte Element in der
    quadratischen Untermatrix aus den Zeilen
    und Spalten k bis n.

    Bringe dieses Element durch geeignetes Vertauschen
    von Gleichungen und Unbekannten an die Stelle a_kk.

    Wenn a_kk = 0
        Stopp.
    sonst
        in Spalte k eliminiere alle
        Einträge unterhalb des Diagonalelementes
        wie in der simplen Gauß-Elimination.
```

Pivot bedeutet „Drehachse, Angelpunkt". Bei der Elimination dreht sich alles darum, welche Gleichung jeweils benützt werden soll, um die entsprechende Unbekannte in den verbleibenden Gleichungen zu eliminieren. Angelpunkt ist das Element a_{kk}, mit dessen Hilfe der Faktor $p = a_{ik}/a_{kk}$ berechnet wird. Es heißt deswegen das *Pivotelement*. Ein günstiges Pivotelement durch geeignete Vertauschungen zu finden heißt *Pivotsuche* oder *-wahl*.

Die notwendigen Zeilen- und Spaltenvertauschungen komplizieren ein Rechenprogramm in Vergleich zur Dreifachschleife der einfachen Gauß-Elimination erheblich. Der Gewinn an zusätzlicher Einsicht in das Verfahren steht dazu in keinem annehmbaren Verhältnis. Deswegen ist hier kein Programm zur vollständigen Pivotsuche abgedruckt.

Üblicherweise führen Gleichungslöser keine vollständige, sondern nur *Zeilen-Pivotsuche* durch. Das heißt, sie beschränken sich der Einfachheit halber auf Vertauschen von Zeilen (=Gleichungen)[7]. Die Empfindlichkeit gegenüber Rundungsfehlern ist bei Zeilen-Pivotsuche etwas höher als bei vollständiger Pivot-Suche.

3.4 Lösbarkeit linearer Gleichungssysteme

Die Faustregel „Bei genausoviel Gleichungen wie Unbekannten gibt es immer eine Lösung“ *ist falsch*. Ich wiederhole (weil ich es bei Prüfungen immer wieder so höre): *ist falsch*!

Bei den drei Gleichungssystemen

$$\begin{aligned} x+y&=2 \\ 2x+2y&=4 \end{aligned} \qquad \begin{aligned} x+y&=2 \\ x+2y&=3 \end{aligned} \qquad \begin{aligned} x+y&=2 \\ 2x+2y&=3 \end{aligned}$$

sehen Sie hoffentlich mit freiem Auge: Beim ersten und beim zweiten ist $x = 1, y = 1$ eine Lösung. Das dritte System ist unlösbar. Beim ersten System gibt es allerdings noch unendlich viele weitere Lösungen. Diese beispiele illustrieren den allgemeinen Fall.

Lösbarkeit linearer Systeme
Für ein lineares Gleichungssystem gilt genau eine von drei Aussagen: Es gibt

- unendlich viele Lösungen;
- eine eindeutige Lösung;
- keine Lösung.

(Sie haben das in der Mathematik-Grundvorlesung gelernt!)

> Dieser Abschnitt behandelt nur Systeme mit genausoviel Gleichungen wie Unbekannten, aber die obige Aussage gilt auch für lineare Systeme mit mehr Gleichungen als Unbekannten. Bei weniger Gleichungen als Unbekannten sind nur die zwei Fälle möglich: keine Lösung oder unendlich viele Lösungen.

Das Gaußsche Eliminationsverfahren kann entscheiden, welcher Fall vorliegt und mögliche Lösungen berechnen.

3.4.1 Eliminationsverfahren

Gauß-Elimination mit vollständiger oder Zeilen-Pivotsuche transformiert die Originalmatrix A und rechte Seite $\mathbf{b}$ auf ein System in *Stufenform*: Von jeder Zeile zur nächsten nimmt (von links her gesehen) die Zahl der führenden Nullen um mindestens eins zu.

Mögliche Fälle nach Abschluss des Eliminationsverfahrens
Das System ist auf Stufenform transformiert.

- Es treten Nullzeilen in A auf und alle entsprechenden Einträge in b sind ebenfals Null: *unendlich viele Lösungen.*
- Es treten Nullzeilen in A auf, aber zumindest ein entsprechender Eintrag in b ist nicht Null: *keine Lösung.*
- Es treten keine Nullzeilen in A auf: *eine eindeutige Lösung.*

[7] Sie finden in Wikipedia unter dem Stichwort *Gaußsches Eliminationsverfahren* Pseudocode in mehreren Varianten.

Der MATLAB-Befehl `rref([A,b])` (rref steht für *reduced row echelon form*, reduzierte Stufenform) führt eine Variante des Eliminationsverfahrens durch (Gauß-Jordan Verfahren), allerdings nur mit Spalten-Pivotsuche. Für die Ergebnismatrix in der rref-Form gelten die gleichen Aussagen wie oben.

Wenn ein Computer die Elimination in Gleitkomma-Arithmetik durchführt, lässt sich nicht so leicht überprüfen, ob Einträge exakt gleich Null sind. Durch Rundungsfehler in den Eingabedaten und während der Rechnung werden Matrixelemente oft nicht exakt Null, sondern nur extrem klein. Es ist überhaupt nicht trivial, eine Schranke anzugeben, ab der Einträge als Null anzusehen sind. Dubiose Grenzfälle, in denen das Eliminationsverfahren gerade noch eine Lösung findet, obwohl Matrixzeilen schon fast null sind, heißen *numerisch singulär*.

3.4.2 Rang der Matrix und der erweiterten Matrix

Der *Rang einer $m \times n$-Matrix* ist die Anzahl ihrer linear unabhängigen Zeilen- oder Spaltenvektoren.

> Auch wenn die Matrix unterschiedlich viele Zeilen und Spalten hat, gilt: es gibt immer genau so viele linear unabhängige Zeilen wie Spalten; kurz: Zeilenrang = Spaltenrang.

Der MATLAB-Befehl `rank(A)` bestimmt den Rang der $n \times n$-Matrix A, und `rank([A,b])` den Rang der *erweiterten Koeffizientenmatrix* (der Matrix des Gleichungssystems mit rechter Seite als letzte Spalte angefügt).

Rang und Lösbarkeit: Fallunterscheidungen

- `rank(A)` = `rank([A,b])` $< n$: *unendlich viele Lösungen*
- `rank(A)` $< n$ und `rank(A)` $\neq$ `rank([A,b])`: *keine Lösung*
- `rank(A)` $= n$: *eindeutige Lösung*

Es gibt verschiedene Methoden, den Rang einer Matrix zu berechnen, zum Beispiel Transformation auf Stufenform durch Gauß-Elimination: Der Rang ist die Anzahl der von Null verschiedenen Zeilen. Es gibt dabei aber kein einfaches Entscheidungskriterium, ob ein Wert bloß infolge Rundungsfehlern oder „echt“ ungleich Null ist. MATLAB verwendet ein sehr rechenaufwendiges Verfahren (Singulärwertzerlegung), das aber die verläßlichsten Aussagen liefert.

Falls ein Gleichungssystem unendlich viele Lösungen hat, lässt sich die allgemeine Lösung entweder aus dem `rref([A,b])`-Ergebnis ablesen oder durch folgende Befehle finden:

`pinv(A)*b` liefert eine spezielle Lösung
`null(A)` liefert den *Nullraum* von A: eine Liste von linear unabhängigen Lösungen des *homogenen Systems* $A\mathbf{x} = 0$.

Die allgemeine Lösung ist die Summe aus der speziellen Lösung und einer beliebigen Linearkombination aus dem Nullraum.

> `null(A,'r')` liefert ebenfalls eine Liste von linear unabhängigen Lösungen des homogenen Systems, aber mit „schöneren“ Zahlen bei einfachen Beispielmatrizen. Allerdings rechnet diese Variante numerisch weniger zuverlässig. (Lassen Sie sich nie von äußerer Schönheit blenden, wenn dahinter Falschheit lauert!)

Am Beispiel des Systems $A\mathbf{x} = \mathbf{b}$ mit

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 4 & 5 & 6 \\ -1 & -2 & -2 & -2 \\ 3 & 6 & 8 & 10 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 2 \end{bmatrix}, \quad \texttt{rref([A,b])} = \begin{bmatrix} 1 & 2 & 0 & -2 & -2 \\ 0 & 0 & 1 & 2 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Aus dem `rref([A,b])`-Ergebnis lässt sich ablesen: $x_2 = \lambda$ und $x_4 = \mu$ sind frei wählbare Parameter, $x_3 = 1 - 2\mu, x_1 = -2 - 2\lambda + 2\mu$. In Vektorform:

$$\mathbf{x} = \begin{bmatrix} -2 \\ 0 \\ 1 \\ 0 \end{bmatrix} + \lambda \begin{bmatrix} -2 \\ 1 \\ 0 \\ 0 \end{bmatrix} + \mu \begin{bmatrix} 2 \\ 0 \\ -2 \\ 1 \end{bmatrix}$$

Bei so einer einfachen Matrix kann MATLAB eine partikuläre Lösung auch in der Form `x=A\b` und den Nullraum mittels `null(A,'r')` berechnen. Dieses Vorgehen ist bei praktischen Problemen mit realen Daten nicht zu empfehlen, aber hier liefert es (abgesehen von einer Warnmeldung) das „schöne“ Ergebnis

$$\mathbf{x} = \begin{bmatrix} 0 \\ -1 \\ 1 \\ 0 \end{bmatrix} + \lambda \begin{bmatrix} -2 \\ 1 \\ 0 \\ 0 \end{bmatrix} + \mu \begin{bmatrix} 2 \\ 0 \\ -2 \\ 1 \end{bmatrix}$$

`pinv(A)*b` und `null(A)`liefern die aus numerischer Sicht vorteilhafteste Darstellung,

$$\mathbf{x} = \begin{bmatrix} -0.2069 \\ -0.41379 \\ 0.034483 \\ 0.48276 \end{bmatrix} + \lambda \begin{bmatrix} -0.77069 \\ 0.10421 \\ 0.56227 \\ -0.28113 \end{bmatrix} + \mu \begin{bmatrix} -0.48335 \\ 0.54725 \\ -0.61115 \\ 0.30558 \end{bmatrix}$$

MATLAB berechnet dieses Resultat mit aufwändigen und zuverlässigen numerischen Verfahren. Aber anders als bei den vorigen Darstellungen der Lösung ergibt probeweises Einsetzen in den Ausdruck $A\mathbf{x} - \mathbf{b}$ nicht exakt Null, sondern aufgund der Rundungsfehler Werte im Bereich 1×10^{-15}. (Schönheit hängt oft doch auch mit Wahrheit zusammen).

3.4.3 Determinante

Die Determinante determiniert, ob ein Gleichungssystem eindeutig lösbar ist.

> Gleichungssysteme $A\mathbf{x} = \mathbf{b}$ mit $\det A \neq 0$ sind eindeutig lösbar.
>
> Allerdings ist diese Regel für das numerische Rechnen unbrauchbar.

Die folgende symmetrische 8×8-Matrix lässt sich in MATLAB durch `A=rosser` erzeugen.

$$A = \begin{bmatrix} 611 & 196 & -192 & 407 & -8 & -52 & -49 & 29 \\ 196 & 899 & 113 & -192 & -71 & -43 & -8 & -44 \\ -192 & 113 & 899 & 196 & 61 & 49 & 8 & 52 \\ 407 & -192 & 196 & 611 & 8 & 44 & 59 & -23 \\ -8 & -71 & 61 & 8 & 411 & -599 & 208 & 208 \\ -52 & -43 & 49 & 44 & -599 & 411 & 208 & 208 \\ -49 & -8 & 8 & 59 & 208 & 208 & 99 & -911 \\ 29 & -44 & 52 & -23 & 208 & 208 & -911 & 99 \end{bmatrix}$$

Für sie berechnet MATLAB derzeit[8] $\det A = -9480{,}580$ also deutlich $\det A \neq 0$. Damit wären Gleichungssysteme mit A eindeutig lösbar. Als Rang berechnet MATLAB aber (korrekt) `rank(A)`=7, und weil $7 < 8$ ist, kann es keine eindeutigen Lösungen geben. MATLABs Wert für die Determinante ist ziemlich falsch.

Für die 6×6-Matrix H, eine sogenannte Hilbert-Matrix,

$$H = \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} & \frac{1}{9} \\ \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} & \frac{1}{9} & \frac{1}{10} \\ \frac{1}{6} & \frac{1}{7} & \frac{1}{8} & \frac{1}{9} & \frac{1}{10} & \frac{1}{11} \end{bmatrix}$$

berechnet MATLAB $\det A = 5{,}3673 \times 10^{-18}$, und das könnte man mit voller Berechtigung gerundet für $\det A = 0$ durchgehen lassen. Als Rang berechnet MATLAB aber (korrekt) `rank(H)`=6. Gleichungssysteme mit H sind also eindeutig lösbar (allerdings extrem anfällig gegenüber Rundungsfehlern).

Diese Beispiele illustrieren:

> Der numerisch berechnete Wert einer Determinante sagt nichts über die Lösbarkeit eines Gleichungssystems aus.

3.5 LR-Zerlegung

Die einfache Gauß-Elimination liefert (wenn sie nicht abbricht) mehr als die Transformation auf Dreiecksgestalt. Sie kann gleichzeitig auch eine Zerlegung

$$A = LR$$

errechnen, wobei L eine untere Dreiecksmatrix mit Einsen in der Hauptdiagonale und R eine obere Dreiecksmatrix ist.

> **LR-Zerlegung**
> Das Gaußsche Eliminationsverfahren ohne Pivotwahl faktorisiert (wenn es nicht abbricht) eine Matrix A in ein Produkt $A = LR$ aus einer linken unteren Dreiecksmatrix L und einer rechten oberen Dreiecksmatrix R.

Nach Durchführen einer Gauß-Elimination mit Pivot-Suche enthalten oberes und unteres Dreieck der Ergebnismatrix die LR-Zerlegung einer Matrix mit - im Vergleich zur Ausgangsmatrix - entsprechend vertauschten Zeilen und Spalten.

Die Elemente von L sind 1 entlang der Hauptdiagonale, und darunter gleich den Multiplikatoren $p = a_{ik}/a_{kk}$ an den entsprechenden Stellen (i, k). Die Elemente von R sind genau jene, die das Eliminationsverfahren in das obere rechte Dreieck schreibt.

Die einzige Änderung im Programm auf Seite 30 zum Eliminationsverfahren ist, sich die Zwischenresultate `p` zu merken. Praktischerweise lässt sich jedes `p` auf dem entsprechenden Feldelement `a[i][k]` speichern; das Verfahren eliminiert nämlich gerade diesen Eintrag, erzeugt also an dieser Stelle eine Null. Die Null muss man sich nicht merken, aber dafür kann man an der dadurch freigewordenen Stelle das Zwischenresultat `p` speichern.

[8] mit Version 2022b. Der Wert mit Version 2021b war $\det A = -10\,611$. Ältere Versionen so um 2018 lieferten $\det A = -9478{,}9$; die 2015-Version bringt $-9448{,}8$; vor 2013 war $\det A = -13\,017$. Es sollte Sie beunruhigen, dass ein Rechenergebnis je nach Programmversion so unterschiedlich ausfällt!

Computerprogramme formulieren das Verfahren in aller Regel so, dass die Originalmatrix A durch R und L überschrieben wird. Das obere Dreieck von A enthält nach erfolgreichem Ablauf die Nichtnull-Einträge von R; die Einsen in der Hauptdiagonale von L verstehen sich von selbst, man braucht sie nicht zu speichern; die restlichen Nichtnull-Elemente von L finden unterhalb der Hauptdiagonale von A Platz. Das Elegante an dieser Speicherung ist, dass sie sich im Verlauf des Verfahrens quasi von selbst ergibt.

Siehe Übungsunterlagen für weitere Informationen!

Für die LR-Zerlegung braucht man keine rechte Seite. Die kommt erst später ins Spiel. Zur Lösung des Systems $A\mathbf{x} = \mathbf{b}$, wenn $A = LR$ bereits gegeben ist, formt man nämlich um:

$$\begin{aligned} A\mathbf{x} &= \mathbf{b} && \\ (LR)\mathbf{x} &= \mathbf{b} && \\ L(R\mathbf{x}) &= \mathbf{b} && \text{setze } \mathbf{y} = R\mathbf{x} \\ L\mathbf{y} &= \mathbf{b} && \text{löse durch Vorwärts-Substitution nach } \mathbf{y} \\ R\mathbf{x} &= \mathbf{y} && \text{löse durch Rückwärts-Substitution nach } \mathbf{x} \end{aligned}$$

Der Rechengang und der Rechenaufwand sind der einfachen Gauß-Elimination völlig äquivalent. Der Vorteil der LR-Zerlegung zeigt sich aber, wenn mehrere Systeme mit der selben Matrix A und unterschiedlichen rechten Seiten $\mathbf{b}_1, \mathbf{b}_2, \ldots$ gelöst werden sollen. Die LR-Zerlegung ist der arbeitsintensive Teil ($\sim n^3/3$ Punktoperationen) und muss nur einmal durchgeführt werden. Die einzelnen Lösungen kosten dann nur mehr $\sim n^2$ Punktoperationen pro rechter Seite.

Für symmetrische Matrizen lassen sich spezielle Varianten der Gauß-Elimination durchführen. Ausnützen der Symmetrie spart Rechenzeit und Speicherplatz. Eine mögliche Zerlegung ist

$$A = LDL^T \,,$$

mit einer Diagonalmatrix D. Die *Cholesky-Zerlegung*

$$A = LL^T$$

ist für symmetrisch positiv definiter Matrizen das Standardverfahren.

3.6 Ein Rechenbeispiel zu Gauß-Elimination und LR-Zerlegung

Das Gaußsche Eliminationsverfahren ist eine Rechenvorschrift zur systematischen Elimination von Unbekannten. Bei Gleichungssystemen mit „einfachen“ Zahlen weicht man oft vom systematischen Weg ab und versucht, den Rechengang abzukürzen (z. B. schon vorhandene Nullen auszunützen). Dabei besteht immer die Gefahr, „im Kreis herum“ zu rechnen, Gleichungen nicht oder doppelt zu verwenden. Es ist daher wichtig, eine allgemein gültige Rechenvorschrift angeben zu können.
Gegeben sei das Gleichungssystem $A \cdot \mathbf{x} = \mathbf{b}$ mit

$$A = \begin{bmatrix} 5 & 6 & 7 \\ 10 & 20 & 23 \\ 15 & 50 & 67 \end{bmatrix} \quad , \quad \mathbf{b} = \begin{bmatrix} 6 \\ 6 \\ 14 \end{bmatrix}$$

Man rechnet man mit der *erweiterten Koeffizientenmatrix*

$$[A\,\mathbf{b}] = \begin{bmatrix} 5 & 6 & 7 & 6 \\ 10 & 20 & 23 & 6 \\ 15 & 50 & 67 & 14 \end{bmatrix}$$

Elimination in der ersten Spalte

(Vergleichen Sie mit dem Algorithmus auf Seite 30.)

$n = 3$; in der Spalte $k = 1$ sollen alle Einträge unterhalb des Diagonalelementes eliminiert werden, das sind die Elemente in Zeilen $i = 2, 3$

$k = 1, i = 2$: Elimination in erster Spalte, zweite Zeile

Elimination von $a_{ik} = a_{21} = 10$ mittels des Diagonalelements $a_{kk} = 5$ von Zeile $k = 1$. Multipliziere erste Zeile mit $a_{ik}/a_{kk} = 2$ und subtrahiere:

$$\begin{array}{cccc|} 10 & 20 & 23 & 6 & \\ 10 & 12 & 14 & 12 & |- \\ \hline 0 & 8 & 9 & -6 & \end{array}$$

$k = 1, i = 3$: Elimination in erster Spalte, dritte Zeile

Elimination von $a_{ik} = a_{31} = 15$ mittels des Diagonalelements $a_{kk} = 5$ von Zeile $k = 1$. Multipliziere erste Zeile mit $a_{ik}/a_{kk} = 3$ und subtrahiere:

$$\begin{array}{cccc|} 15 & 50 & 67 & 14 & \\ 15 & 18 & 21 & 18 & |- \\ \hline 0 & 32 & 46 & -4 & \end{array}$$

Transformierte erweiterte Koeffizientenmatrix

nach Elimination in der ersten Spalte:

$$[A\,\mathbf{b}]^{(1)} = \begin{bmatrix} 5 & 6 & 7 & 6 \\ 0 & 8 & 9 & -6 \\ 0 & 32 & 46 & -4 \end{bmatrix}$$

Elimination in der zweiten Spalte

In der Spalte $k = 2$ sollen alle Einträge unterhalb des Diagonalelementes eliminiert werden, das ist nur mehr das Element in Zeile $i = 3$

$k = 2, i = 3$: Elimination in zweiter Spalte, dritte Zeile

Elimination von $a_{ik} = a_{32} = 32$ mittels des Diagonalelements $a_{kk} = 8$ von Zeile $k = 2$. Multipliziere zweite Zeile mit $a_{ik}/a_{kk} = 4$ und subtrahiere:

$$\begin{array}{cccc|} 0 & 32 & 46 & -4 & \\ 0 & 32 & 36 & -24 & |- \\ \hline 0 & 0 & 10 & 20 & \end{array}$$

Transformierte erweiterte Koeffizientenmatrix

Nach Elimination in der zweiten Spalte ist das Eliminationsverfahren beendet.

$$[A\,\mathbf{b}]^{(2)} = \begin{bmatrix} 5 & 6 & 7 & 6 \\ 0 & 8 & 9 & -6 \\ 0 & 0 & 10 & 20 \end{bmatrix}$$

Rücksubstitution

Aus der dritten Zeile:

$$\begin{aligned} 10x_3 &= 20 \\ x_3 &= 2 \end{aligned}$$

Einsetzen für x_3 in zweiter Zeile

$$\begin{aligned} 8x_2 + 9x_3 &= -6 \\ 8x_2 + 18 &= -6 \\ 8x_2 &= -24 \\ x_2 &= -3 \end{aligned}$$

Einsetzen für x_2 und x_3 in erster Zeile

$$\begin{aligned} 5x_1 + 6x_2 + 7x_3 &= 6 \\ 5x_1 - 18 + 14 &= 6 \\ 5x_1 &= 10 \\ x_1 &= 2 \end{aligned}$$

Der Matlab-Befehl `x = A\b` arbeitet im Prinzip nach diesem Verfahren, allerdings in Form der LR-Zerlegung mit Spaltenpivotsuche.

Mehrere rechte Seiten

Sind zur selben Matrix A Lösungen für mehrere rechte Seiten zu berechnen, fügt man diese der erweiterten Koeffizientenmatrix als weitere Spalten an und rechnet in gleicher Weise.

Pivotwahl

Bei diesem Beispiel war es nicht notwendig, Gleichungen zu tauschen, um Divisionen durch Null zu vermeiden. Bei Spaltenpivotwahl hätte man vor dem ersten Schritt die dritte Gleichung zur ersten gemacht (weil sie den betragsgrößten Eintrag in der ersten Spalte hat).

LR-Zerlegung

Die auf Dreiecksform transformierte Matrix und die entsprechenden Pivot-Faktoren liefern auch die LR-Zerlegung. Eine rechte Seite ist für die LR-Zerlegung nicht notwendig.

$$\begin{bmatrix} 5 & 6 & 7 \\ 10 & 20 & 23 \\ 15 & 50 & 67 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 4 & 1 \end{bmatrix} \cdot \begin{bmatrix} 5 & 6 & 7 \\ 0 & 8 & 9 \\ 0 & 0 & 10 \end{bmatrix}$$

Der MATLAB-Befehl `[L, U]=lu(A)` verwendet im Prinzip das Gaußsche Eliminationsverfahren, aber liefert zu diesem Zahlenbeispiel eine andere „quasi“-LR-Zerlegung. Die U- bzw. R-Matrix[9] ist eine echte obere Dreiecksmatrix, L ist eine „durcheinandergeratene“ untere Dreiecksmatrix. Begründung: Spaltenpivotsuche vertauscht Zeilen in der Matrix und ändert im Rechengang Reihenfolge und Zahlenwerte. Sie verringert dadurch die Rundungsfehler.

[9] Dem deutschen Fachbegriff Links-Rechts-Zerlegung entspricht auf Englisch *lower–upper (LU) decomposition* oder *factorization*. Deswegen heißt der entsprechende MATLAB-Befehl `lu` und nicht `lr`.

Zerlegung mit dem Befehl `[L, U]=lu(A)` liefert

$$\begin{bmatrix} 5 & 6 & 7 \\ 10 & 20 & 23 \\ 15 & 50 & 67 \end{bmatrix} = \begin{bmatrix} 1/3 & 4/5 & 1 \\ 2/3 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 15 & 50 & 67 \\ 0 & -40/3 & -65/3 \\ 0 & 0 & 2 \end{bmatrix}$$

3.7 Weitere Anwendungen der LR-Zerlegung

3.7.1 Determinante

Determinante
Wenn eine Faktorisierung $A = LR$ gegeben ist, ist $\det A$ das Produkt der Hauptdiagonalelemente von R.

Begründung: Die Determinante einer Dreiecksmatrix ist das Produkt der Hauptdiagonalelemente. In L stehen dort lauter Einsen, somit ist $\det L = 1$. Nach den Rechenregeln für Determinanten ist dann

$$\det A = \det(LR) = (\det L)(\det R) = \det R \; .$$

Rechenaufwand für eine $n \times n$-Matrix: $n^3/3 + 2n/3 - 1$ Punktoperationen.

Vergleiche dazu das klassische Verfahren, Entwickeln nach Zeilen oder Spalten: Für eine $n \times n$-Matrix sei dafür $w(n)$ der Rechenaufwand an Punktoperationen. Im allgemeinen Fall sind n Unterdeterminanten von $(n-1) \times (n-1)$-Matrizen zu berechnen und noch mit den jeweiligen Elementen zu multiplizieren. Für den Rechenaufwand gilt also

$$w(n) = nw(n-1) + n = n(w(n-1) + 1) \; .$$

Die Funktion $w(n)$ wächst rasch, stärker als die Faktorielle $n!$. Für die Determinante einer 10×10-Matrix braucht ein schneller PC (10^7 Multiplikationen/sec) eine knappe Sekunde, schon bei einer 13×13-Matrix geht sich eine Viertelstunde Kaffeepause aus, auf die Determinante einer 15×15-Matrix warten Sie zweieinhalb Tage, dreizehn Jahrtausende bei einer 20×20-Matrix, und eine 25×25-Matrix wäre nach 80 Milliarden Jahren noch nicht fertig.
Das rasante Wachsen von $w(n)$ und den vergleichsweise geringen Aufwand der LR-Zerlegung illustriert die folgende Tabelle. Sie soll eindringlich auf die Bedeutung rechengünstiger Algorithmen und den Unterschied zwischen polynomialer und exponentieller Laufzeit hinweisen.

n	$w(n)$	$n^3/3 + 2n/3 - 1$
2	2	3
3	9	10
4	40	23
5	205	44
6	1 236	75
7	8 659	118
8	69 280	175
9	623 529	248
10	6 235 300	339
15	2 246 953 104 075	1 134
20	4 180 411 311 071 440 000	2 679
25	26 652 630 354 867 072 870 693 625	5 224

3.7.2 Inverse

Die zu einer (nichtsingulären) Matrix A inverse Matrix A^{-1} braucht kaum ein Rechenverfahren in expliziter Form. Ist etwa der Vektor $\mathbf{x} = A^{-1}\mathbf{y}$ gefragt, so erhält man $\mathbf{x}$ auch genauso gut als Lösung des Gleichungssystems $A\mathbf{x} = \mathbf{y}$, und das ist von Rechenaufwand und -genauigkeit her günstiger.

Warnung 1: Bevor Sie eine Inverse berechnen, fragen Sie dreimal nach, ob Sie diese wirklich brauchen.

Warnung 2: Falls Sie sich noch an die aus der linearen Algebra bekannte Formel (die mit den Determinanten der Kofaktoren) erinnern: vergessen Sie diese. Sie ist von theoretischer Bedeutung, weil sie die Existenz von Inversen nichtsingulärer Matrizen zeigt. Berechnen sollten Sie die Inverse so nicht (überlegen Sie: Rechenaufwand $O(n^5)$, wenn Sie die einzelnen Unterdeterminanten alle mittels LR-Zerlegung rechnen; exponentieller Rechenaufwand, wenn Sie die Determinanten entwickeln).

Wenn es denn sein muss, gehen Sie so vor: Nennen Sie die erste Spalte der Inversen $\mathbf{x}_1$. Die erste Spalte der Einheitsmatrix I ist der Einheitsvektor $\mathbf{e}_1 = (1, 0, \ldots, 0)^T$. Laut Definition gilt

$$AA^{-1} = I \ .$$

Daher gilt insbesondere

$$A\mathbf{x}_1 = \mathbf{e}_1 \ .$$

Das heißt: die erste Spalte der Inversen erhalten Sie als Lösung eines Gleichungssystems mit dem ersten Einheitsvektor als rechter Seite.

In offenkundiger Verallgemeinerung:

Inverse
Die i-te Spalte von A^{-1} ist Lösung des Gleichungssystems

$$A\mathbf{x}_i = \mathbf{e}_i \ .$$

Hier haben Sie also Gleichungssysteme mit derselben Matrix A und mehreren rechten Seiten zu lösen. Vorgangsweise:

Zerlegung $A = LR$; (Aufwand $(n^3 - n)/3$).
Für $i = 1, \ldots, n$
    Lösung $LR\mathbf{x}_i = \mathbf{e}_i$; (Aufwand jeweils n^2).

Rechenaufwand gesamt $(4n^3 - n)/3$.

Der *Gauß-Jordan-Algorithmus* ist eine speziell günstig organisierte Form des Eliminationsverfahrens; er eignet sich gut zur Berechnung mit Stift und Papier. (Sie kennen dieses Verfahren aus Mathematik 1.)

3.7.3 Symmetrisch positiv definite Matrizen

Einfache Argumente der linearen Algebra zeigen: Für symmetrisch positiv definite Matrizen bricht die einfache Gauß-Elimination nie wegen $a_{kk} = 0$ ab. Pivot-Suche ist daher unnötig. Umgekehrt kann man symmetrisch positiv definite Matrizen dadurch charakterisieren, dass die im k-ten Schritt der LR-Zerlegung auftretenden a_{kk} alle > 0 sind.

Allerdings führt man bei symmetrisch positiv definiten Matrizen (wie bereits auf Seite 36 erwähnt) eher Zerlegungen der Form $A = LL^T$ (Cholesky-Zerlegung) oder $A = LDL^T$ durch. Vorteil: bei geschickter Programmierung weniger Rechenzeit und Speicherplatz erforderlich.

3.7.4 Unvollständige Zerlegung

Auch wenn in einer Matrix A die meisten Elemente null sind, können die Faktoren L und R deutlich mehr Nichtnull-Einträge enthalten. Durch Gauß-Elimination werden zusätzliche *Fällterme* erzeugt (also Elemente $\neq 0$ an Stellen, wo die Ausgangsmatrix Elemente $= 0$ hatte). Wenn man alle (oder kleine) Fällterme einfach vernachlässigt, reduzieren sich der Rechenaufwand und benötigte Speicherplatz einer solchen *unvollständigen Zerlegung* drastisch. Natürlich gilt dann nicht mehr $LR = A$, sondern $LR = \tilde{A}$ für eine Näherung $\tilde{A}$ an A. Unvollständigen Zerlegungen sind leistungsfähige Präkonditionierer für iterative Gleichungslöser.

Das Prinzip lässt sich durch eine geringfügige Änderung im Beispielprogramm zur LR-Zerlegung illustrieren (mehr Informationen im Übungsteil): Man ersetze dort in der innersten Schleife

```
For j = k + 1 To n
   a(i, j) = a(i, j) - p * a(k, j)
Next
```

durch

```
For j = k + 1 To n
   if a(i, j) <> 0 then
      a(i, j) = a(i, j) - p * a(k, j)
Next
```

Damit ist aus der LR-Zerlegung eine unvollständige Zerlegung ohne zusätzliche Fällterme geworden. In dieser Form spart das Programm allerdings keinen Speicherplatz und nicht wirklich Rechenzeit. Dazu wären Datenstrukturen notwendig, die gezielt nur die Nichtnull-Elemente speichern, auf diese zugreifen und damit manipulieren (Speicherformate für spärlich besetzte Matrizen).

3.8 Fehlerempfindlichkeit

Rundungsfehler und Fehler in den Eingabedaten verfälschen eine Matrix A zu $A + \delta A$ und die rechte Seite $\mathbf{b}$ zu $\mathbf{b} + \delta\mathbf{b}$. Die Lösung dieses leicht veränderten Systems wird um ein (hoffentlich nicht zu großes) $\delta\mathbf{x}$ von der Lösung des unverfälschten Systems abweichen:

$$(A + \delta A)(\mathbf{x} + \delta\mathbf{x}) = \mathbf{b} + \delta\mathbf{b} \quad .$$

Wie hängt $\delta\mathbf{x}$ von δA und $\delta\mathbf{b}$ ab?

Konditionszahl
Die *Konditionszahl* $\kappa(A)$ misst, wie empfindlich in einem System $A\mathbf{x} = \mathbf{b}$ der relative Fehler von $\mathbf{x}$ von kleinen relativen Änderungen in A und $\mathbf{b}$ abhängt.

$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|} \leq \kappa(A) \left(\frac{\|\delta A\|}{\|A\|} + \frac{\|\delta\mathbf{b}\|}{\|\mathbf{b}\|} \right)$$

In der obigen Ungleichung bezeichnet $\|\cdot\|$ sowohl eine Vektornorm (zum Beispiel 1-, 2- oder ∞-Norm) als auch die entsprechende Matrixnorm. Aus den Rechenregeln für Vektor- und Matrixnormen (vergleiche Abschnitt 2.3) lässt sich herleiten:

$$\kappa(A) = \|A\|\|A^{-1}\|$$

Beweisskizze: Beginne mit dem gestörten System

$$(A + \delta A)(\mathbf{x} + \delta\mathbf{x}) = \mathbf{b} + \delta\mathbf{b}\,,$$

löse Klammern auf

$$A\mathbf{x} + A \cdot \delta\mathbf{x} + \delta A \cdot \mathbf{x} + \delta A \cdot \delta\mathbf{x} = \mathbf{b} + \delta\mathbf{b}\,;$$

weil $A\mathbf{x} = \mathbf{b}$, können wir links $A\mathbf{x}$ und rechts $\mathbf{b}$ streichen. Für kleine $\delta\mathbf{b}$ und δA ist das Produkt $\delta A \cdot \delta\mathbf{x}$ von höherer Ordnung klein; wir vernachlässigen ihn hier. Somit bleibt

$$A \cdot \delta\mathbf{x} + \delta A \cdot \mathbf{x} = \delta\mathbf{b}\ .$$

Daraus lässt sich $\delta\mathbf{x}$ ausdrücken:

$$\delta\mathbf{x} = A^{-1}\left(\delta\mathbf{b} - \delta A \cdot \mathbf{x}\right)\ ,$$

wende auf beiden Seiten eine Vektornorm an,

$$\|\delta\mathbf{x}\| = \|A^{-1}\left(\delta\mathbf{b} - \delta A \cdot \mathbf{x}\right)\|\,,$$

nütze eine Eigenschaft der Matrixnorm, Ungleichung (9),

$$\|\delta\mathbf{x}\| \le \|A^{-1}\| \cdot \|\left(\delta\mathbf{b} - \delta A \cdot \mathbf{x}\right)\|\,,$$

verwende die Dreiecksungleichung,

$$\|\delta\mathbf{x}\| \le \|A^{-1}\|\left(\|\delta\mathbf{b}\| + \|\delta A \cdot \mathbf{x}\|\right)\ ,$$

erweitere die Terme in der Klammer,

$$\|\delta\mathbf{x}\| \le \|A^{-1}\|\left(\frac{\|A\mathbf{x}\|}{\|\mathbf{b}\|}\|\delta\mathbf{b}\| + \frac{\|A\|}{\|A\|}\|\delta A \cdot \mathbf{x}\|\right)\ ,$$

vewende noch einmal eine Ungleichung für die Matrixnormen und hebe $\|A\|$ heraus,

$$\|\delta\mathbf{x}\| \le \|A^{-1}\| \cdot \|A\|\left(\frac{\|\delta\mathbf{b}\|}{\|\mathbf{b}\|}\|\mathbf{x}\| + \frac{\|\delta A\|}{\|A\|}\|\mathbf{x}\|\right)\ ,$$

eine letzte Division durch $\|\mathbf{x}\|$, und wir sind fertig:

$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|} \le \|A^{-1}\| \cdot \|A\|\left(\frac{\|\delta\mathbf{b}\|}{\|\mathbf{b}\|} + \frac{\|\delta A\|}{\|A\|}\right)\ .$$

Der relative Fehler in $\mathbf{x}$ kann also $\kappa(A)$-mal größer als der relative Fehler in A und $\mathbf{b}$ sein. Ein Gleichungssystem, dessen Matrix eine große Konditionszahl hat, wird sehr empfindlich auf Fehler in den Eingabedaten reagieren. Ein solches System heißt *schlecht konditioniert*. Geometrische Veranschaulichung: schleifender Schnitt zweier Geraden.

Die Berechnung der Konditionszahl direkt gemäß der Definition würde die Berechnung der Inversen erfordern und wäre unsinnig aufwendig. Viele Gleichungslöser liefern Schätzungen von $\kappa(A)$ als Nebenprodukt. Es gilt zum Beispiel

$$\kappa(A) \ge \frac{\max|\lambda|}{\min|\lambda|}$$

(Verhältnis von größtem zu kleinsten Eigenwert-Bertag; Eigenwerte werden in Abschnitt 9 behandelt).

4 Iterative Verfahren für lineare Gleichungssysteme

Gegeben sei ein lineares Gleichungssystem in n Gleichungen und Unbekannten.

$$\begin{array}{lcl} a_{11}x_1 + a_{12}x_2 + & \ldots & + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + & \ldots & + a_{2n}x_n = b_2 \\ \vdots & & \vdots \quad\; \vdots \\ a_{n1}x_1 + a_{n2}x_2 + & \ldots & + a_{nn}x_n = b_n \end{array} \tag{10}$$

In Matrixschreibweise

$$A\mathbf{x} = \mathbf{b}\,. \tag{11}$$

Das klassische Lösungsverfahren dafür, jedenfalls für Systeme von zwei bis einigen hundert Gleichungen, ist Gauß-Elimination. Sie wird in Kapitel 3 systematisch behandelt. Aus vielen Anwendungen (Strömungssimulation, Seismik, Computertomographie, Festigkeitsrechnungen mit finiten Elementen...) resultieren Gleichungssysteme mit vielen tausend oder sogar Millionen Unbekannten. Solche Systeme werden meist iterativ gelöst. Sie lernen hier nur einige Basis-Verfahren kennen, auf denen die leistungsfähigeren Methoden aufbauen.

4.1 Einfache iterative Verfahren: Jacobi, Gauß-Seidel, SOR

Angenommen, die Diagonalelemente a_{ii} einer $n \times n$-Matrix A sind alle ungleich null. Dann wäre folgendes Rezept zur Lösung von $A\mathbf{x} = \mathbf{b}$ möglich (ein Fixpunkt-Verfahren):

Jacobi-Verfahren für $A\mathrm{x} = \mathrm{b}$, einfach formuliert
Löse jede Gleichung nach ihrem Diagonal-Term auf, setze Startwerte ein, iteriere.

Ausführlicher in der komponentenweisen Schreibweise (10) formuliert: Bringen Sie jeweils in der i-ten Zeile alle Terme bis auf den i-ten auf die rechte Seite und lösen Sie nach x_i auf. Ein entsprechend umgeformtes 3×3-System sieht dann so aus:

$$\begin{aligned} x_1 &= (b_1 - a_{12}x_2 - a_{13}x_3)/a_{11} \\ x_2 &= (b_2 - a_{21}x_1 - a_{23}x_3)/a_{22} \\ x_3 &= (b_3 - a_{31}x_1 - a_{32}x_2)/a_{33} \end{aligned}$$

Angenommen, $\mathbf{x}^{(k)}$ ist ein näherungsweiser Lösungsvektor. Das Jacobi-Verfahren erzeugt eine neue Näherung durch

$$\begin{aligned} x_1^{(k+1)} &= (b_1 - a_{12}x_2^{(k)} - a_{13}x_3^{(k)})/a_{11} \\ x_2^{(k+1)} &= (b_2 - a_{21}x_1^{(k)} - a_{23}x_3^{(k)})/a_{22} \\ x_3^{(k+1)} &= (b_3 - a_{31}x_1^{(k)} - a_{32}x_2^{(k)})/a_{33} \end{aligned}$$

Vielleicht finden Sie Matrix-Schreibweise übersichtlicher. Dazu definieren wir eine Matrix $D = [d_{ij}]$ mit den gleichen Diagonalelementen wie A und null in allen Nichtdiagonalelementen. Die restlichen Elemente von A schreiben wir in eine Matrix E:

$$A = D + E \text{ mit } D = [d_{ij}], \quad d_{ij} = \begin{cases} a_{ii} & \text{falls } i = j, \\ 0 & \text{sonst.} \end{cases} \qquad E = A - D\,. \tag{12}$$

Das Gleichungssystem (11) lässt sich dann äquivalent umformen zu

$$\begin{aligned} A\mathbf{x} &= \mathbf{b} \\ (D+E)\mathbf{x} &= \mathbf{b} \\ D\mathbf{x} &= \mathbf{b} - E\mathbf{x} \\ \mathbf{x} &= D^{-1}(\mathbf{b} - E\mathbf{x}) \ . \end{aligned}$$

Die letzte Gleichung ist eine Fixpunktgleichung. Die entsprechende Fixpunkt-Iteration

$$\mathbf{x}^{(k+1)} = D^{-1}(\mathbf{b} - E\mathbf{x}^{(k)}) \tag{13}$$

heißt *Jacobi-Verfahren* .is called *Jacobi method* .

Iterationsschritt des Jacobi-Verfahrens
In Matrix-Schreibweise für Zerlegung $A = D + E$:

$$\mathbf{x}^{(k+1)} = D^{-1}(\mathbf{b} - E\mathbf{x}^{(k)})$$

Komponentenweise geschrieben: für $i = 1, \ldots, n$

$$x_i^{(k+1)} = \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right) / a_{ii}$$

Das Jacobi-Verfahren nützt nicht die aktuellste Information zur Berechnung von $x_i^{(k+1)}$. Beispielsweise verwendet es $x_1^{(k)}$ zur Berechnung von $x_2^{(k+1)}$, obwohl $x_1^{(k+1)}$ bereits verfügbar ist. Wenn wir das Verfahren so formulieren, dass es immer die aktuellsten Näherungswerte an die x_i verwendet, erhalten wir das *Gauß-Seidel-Verfahren* .

Für die Matrix-Schreibweise des Gauß-Seidel-Verfahrens definieren wir eine Matrix $C = [c_{ij}]$ mit den gleichen Elementen wie A in und unterhalb der Hauptdiagonale und Null oberhalb der Hauptdiagonale. Die restlichen Elemente von A schreiben wir in eine Matrix E:

$$A = C + E \text{ mit } C = [c_{ij}], \quad c_{ij} = \begin{cases} a_{ij} & \text{falls } i \geq j, \\ 0 & \text{sonst.} \end{cases} \qquad E = A - C \ . \tag{14}$$

Dieselben Schritte, die für das Jacobi-Verfahren zur Fixpunkt-Gleichung 13 geführt haben, können wir mit der Matrix C statt D wiederholen und erhalten die Iterationsvorschrift für das Gauß-Seidel-Verfahren:

$$\mathbf{x}^{(k+1)} = C^{-1}(\mathbf{b} - E\mathbf{x}^{(k)}) \tag{15}$$

Iterationsschritt des Gauß-Seidel-Verfahrens
einfach formuliert
Löse der Reihe nach jede Gleichung nach ihrem Diagonal-Term auf, setze Startwerte ein, iteriere, verwende jeweils neuerste Näherungswerte.

Matrix-Schreibweise für Zerlegung $A = C + E$

$$\mathbf{x}^{(k+1)} = C^{-1}(\mathbf{b} - E\mathbf{x}^{(k)})$$

Komponenten-Schreibweise
für $i = 1, \ldots, n$

$$x_i^{(k+1)} = \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right) / a_{ii}$$

Das Gauß-Seidel-Verfahren lässt sich oft deutlich beschleunigen, wenn man den aus der Iterationsformel erhaltenen Wert nicht direkt verwendet, sondern die Änderung von $x_i^{(k)}$ zu $x_i^{(k+1)}$ noch um einen Faktor $\omega > 1$ vergrößert. Dieses iterative Verfahren heißt *SOR-Verfahren* (SOR steht für *successive overrelaxation*)

Ein geeigneten Wert für ω lässt sich aber nicht so einfach angeben. Die Theorie sagt: $1 \leq \omega < 2$, mit Werten eher in der Nähe von 2. Für $\omega = 1$ reduziert sich SOR auf Gauß-Seidel.

Iterationsschritt des SOR-Verfahrens
einfach formuliert
Jeweils neuer Näherungswert zuerst als Zwischenresultat $y_i^{(k+1)}$ aus Gauß-Seidel-Schritt; endgültiger Näherungswert durch Extrapolation (Überrelaxation) aus alter Näherung und Zwischenresultat: $x_i^{(k+1)} = \omega y_i^{(k+1)} + (1 - \omega) x_i^{(k)}$

Die Komponenten-Schreibweise sieht hier bereits etwas unübersichtlich aus.

für $i = 1, \ldots, n$

$$y_i^{(k+1)} = \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right) / a_{ii}$$

$$x_i^{(k+1)} = \omega y_i^{(k+1)} + (1 - \omega) x_i^{(k)}$$

Auch dieses Verfahren lässt sich mit einer Zerlegung $A = B + E$ ähnlich wie die Gleichungen 13 und 15 anschreiben:

$$\mathbf{x}^{(k+1)} = B^{-1}(\mathbf{b} - E\mathbf{x}^{(k)}) \tag{16}$$

Darin ist B eine Kombination der Matrizen C und D von vorhin (12, 14), und zwar

$$B = C + \left(\frac{1}{\omega} - 1 \right) D$$

4.2 Konvergenz des Jacobi- und des Gauß-Seidel-Verfahrens

Nicht für jede beliebige Matrix A konvergieren die drei oben vorgestellten Verfahren. Die Konvergenz des Jacobi-Verfahrens lässt sich zeigen, indem man nachweist, dass es sich bei der Fixpunkt-Iteration um eine kontrahierende Abbildung handelt. Dazu definieren wir:

Eine $n \times n$-Matrix $A = [a_{ij}]$ heißt *stark diagonaldominant*, wenn

$$|a_{ii}| > \sum_{j=1, j\neq i}^{n} |a_{ij}| \quad \text{für} \quad i = 1, 2, \ldots, n$$

Es muss also in jeder Zeile die Summe der Beträge der Nichtdiagonalelemente kleiner sein als der Betrag des Diagonalelementes.

Konvergenz des Jacobi-Verfahrens
Das Jacobi-Verfahren konvergiert bei Gleichungssystemen mit stark diagonaldominanter Matrix für beliebige Startwerte zur eindeutigen Lösung.

Beweis: Wir zeigen, dass die zur Iterationsvorschrift (13) gehörige Funktion $\mathbf{\Phi}(\mathbf{x}) = D^{-1}(\mathbf{b} - E\mathbf{x})$ für beliebige $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ eine kontrahierende Abbildung in der Maximumsnorm ist. Laut Abschnitt 2.4 ist damit Konvergenz gewährleistet. Wir vereinfachen erst einmal

$$\mathbf{\Phi}(\mathbf{x}) - \mathbf{\Phi}(\mathbf{y}) = D^{-1}(\mathbf{b} - E\mathbf{x}) - D^{-1}(\mathbf{b} - E\mathbf{y}) = D^{-1}E(\mathbf{y} - \mathbf{x})$$

Die i-te Zeile der Matrix $D^{-1}E$ lautet

$$\frac{a_{i1}}{a_{ii}} \quad \frac{a_{i2}}{a_{ii}} \quad \ldots \quad \frac{a_{i,i-1}}{a_{ii}} \quad 0 \quad \frac{a_{i,i+1}}{a_{ii}} \quad \ldots \quad \frac{a_{in}}{a_{ii}}$$

Die Summe der Elementbeträge in dieser Zeile ist < 1 (Begründung: Aus der Summe $1/|a_{ii}|$ herausheben, Diagonaldominanz ausnützen). Damit ist auch für die Zeilensummen- oder Unendlich-Norm der Matrix $D^{-1}E$ gezeigt:

$$\|D^{-1}E\|_\infty < 1 \ .$$

Aus einer Eigenschaft der Matrixnorm, Ungleichung (9), folgt sofort die Kontraktionseigenschaft

$$\|\mathbf{\Phi}(\mathbf{x}) - \mathbf{\Phi}(\mathbf{y})\|_\infty = \|D^{-1}E(\mathbf{y} - \mathbf{x})\|_\infty \leq \|D^{-1}E\|_\infty \cdot \|\mathbf{y} - \mathbf{x}\|_\infty \leq C\|\mathbf{y} - \mathbf{x}\|_\infty$$

mit $C = \|D^{-1}E\|_\infty < 1$.

Mit beträchtlich mehr Aufwand lässt sich zeigen, dass auch für eine größere Klasse von Matrizen, nämlich *schwach diagonaldominante, irreduzible* Matrizen, das Jacobi-Verfahren konvergiert. Diese Aussage ist wichtig, weil viele Aufgaben aus der Praxis (numerische Lösung partieller Differentialgleichungen) genau solche Matrizen liefern. Der Vollständigkeit halber hier die Definitionen:
Eine $n \times n$-Matrix $A = [a_{ij}]$ ist *schwach diagonaldominant*, wenn

$$|a_{ii}| \geq \sum_{j=1, j\neq i}^{n} |a_{ij}| \quad \text{für} \quad i = 1, 2, \ldots, n$$

und zumindest für ein i echte Ungleichheit gilt. Um Irreduzibilität zu untersuchen, zeichnen Sie für jedes i einen Punkt. Für jedes Matrixelement $a_{ij} \neq 0$ in A verbinden Sie die Punkte i und j durch einen Pfeil in Richtung $i \to j$. Die Zeichnung stellt einen *gerichteten Graph* dar. Wenn man von jedem beliebigen Punkt zu jedem anderen gelangen kann, indem man den Pfeilen folgt, dann heißt dieser Graph *zusammenhängend* und die Matrix A ist *irreduzibel*.

In der Regel konvergiert das Gauß-Seidel-Verfahren rascher als das Jacobi-Verfahren. Es braucht für die gleiche Genauigkeit typischerweise nur halb so viele Iterationen. Das SOR-Verfahren mit optimal gewähltem Relaxationsparameter ω braucht größenordnungsmäßig $\sqrt{N}$ Iterationen, wo das Jacobi-Verfahren N Iterationen braucht.

Es gibt aber auch Matrizen, für die ein Verfahren konvergiert, das andere aber nicht.

Wir zitieren hier ohne Beweis zwei weitere Sätze, die Konvergenzbedingungen formulieren.

> Wenn A positive Elemente in der Hauptdiagonale hat und alle andern Elemente ≤ 0 sind, dann konvergiert das Gauß-Seidel-Verfahren genau dann, wenn das Jacobi-Verfahren konvergiert. Wenn beide Verfahren konvergieren, dann ist das Gauß-Seidel-Verfahren asymptotisch schneller (*Satz von Stein und Rosenberg*).
>
> Ist A symmetrisch positiv definit, dann konvergiert das Gauß-Seidel-Verfahren.

4.3 Moderne iterative Gleichungslöser

Gleichungssysteme aus dem Bereich der Strömungssimulation, der Festigkeitsanalyse, der Finanzmathematik und vieler weiterer Anwendungsgebiete erreichen leicht eine Größe von mehreren Millionen Unbekannten. Dafür sind aber pro Matrixzeile nur wenige Elemente von Null verschieden (So eine Matrix heißt *schwach besetzt*, englisch *sparse matrix*). Für die Lösung solcher Systeme werden heute fast ausschließlich iterative Verfahren eingesetzt. Die klassischen Methoden (Jacobi, Gauß-Seidel) konvergieren aber zu langsam und erfordern daher zuviel Rechenaufwand.

Diese Unterlagen können nur eine einführende Übersicht auf einige Prinzipien geben, die moderne iterative Gleichungslöser verwenden.

4.3.1 Splittings, Präkonditionierung

Angenommen, Sie sollen das System

$$A\mathbf{x} = \mathbf{b}$$

lösen.

Günstige Taktik: Sie ersetzen die Matrix A in dieser Aufgabe durch eine andere Matrix $\tilde{A}$, für die Sie Gleichungssysteme viel leichter lösen können. Sie können es sich dabei einfach machen und für $\tilde{A}$ die Einheitsmatrix I wählen, oder den diagonalen Anteil von A, oder gezielt nur bestimmte Matrixelemente aus A herausstreichen.

Schreiben Sie $A = \tilde{A} + E$. Eine solche Aufspaltung heißt ein *Splitting* von A in eine Näherung (auch: *Präkonditionierer*) und einen Restanteil E. Das ursprüngliche Gleichungssystem formulieren Sie dann als Fixpunkt-Aufgabe um.

$$\begin{aligned} A\mathbf{x} &= \mathbf{b} \\ (\tilde{A} + E)\mathbf{x} &= \mathbf{b} \\ \tilde{A}\mathbf{x} + E\mathbf{x} &= \mathbf{b} \\ \tilde{A}\mathbf{x} &= \mathbf{b} - E\mathbf{x} \\ \mathbf{x} &= \tilde{A}^{-1}(\mathbf{b} - E\mathbf{x}) \end{aligned}$$

Das Jacobi-Verfahren benützt diese Idee mit $\tilde{A} = D$, dem diagonalen Anteil. Das Gauß-Seidel-Verfahren lässt sich ebenfalls in dieser Form darstellen, indem man $\tilde{A}$ aus A durch Streichen sämtlicher Elemente oberhalb der Hauptdiagonale gewinnt.

Allgemein gilt, je besser $\tilde{A}$ die Original-Matrix approximiert, um so rascher konvergiert ein solches iterative Verfahren. Besonders gute Splittings entstehen aus *unvollständiger LR-Zerlegung*. Diese Methoden werden bei den Eliminationsverfahren in Abschnitt 3.7.4 kurz behandelt.

In der oben angegebenen Fixpunkt-Form wird das Verfahren aber nicht implementiert, da man die Matrix $\tilde{A}^{-1}$ nur in einfachsten Fällen (wie etwa $\tilde{A} = D$) explizit bilden sollte. Eine algebraisch gleichwertige, aber für Rechner geeignete Form lautet

Iterativer Gleichungslöser, Grundschema für $A = \tilde{A} + E$
Für ein geeignetes Splitting $A = \tilde{A} + E$, einen beliebigen Startvektor $\mathbf{x}^{(0)}$ und eine vorgegebene Genauigkeitsschranke $\epsilon > 0$ findet dieser Algorithmus eine Näherungslösung von $A\mathbf{x} = \mathbf{b}$.

Beginne mit Startvektor $\mathbf{x}^{(0)}$
setze $\mathbf{r}^{(0)} = \mathbf{b} - A\mathbf{x}^{(0)}$
iteriere für $k = 0, 1, 2, \ldots$
  löse $\tilde{A}\mathbf{d}^{(k+1)} = \mathbf{r}^{(k)}$
  setze $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{d}^{(k+1)}$
  setze $\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - A\mathbf{d}^{(k+1)}$
bis $\|\mathbf{r}^{(k+1)}\| < \epsilon$
Ergebnis: Näherungslösung $\mathbf{x}^{(k+1)}$

Bei Iterationen dieser Form nennt man $\tilde{A}$ auch die *Präkonditionierungsmatrix*.

Für einen Vektor $\mathbf{x}$ und gegebenes A und $\mathbf{b}$ bezeichnet man den Ausdruck $\mathbf{b} - A\mathbf{x}$ als *Residuum*. Die Aufgabe, ein Gleichungssystem $A\mathbf{x} = \mathbf{b}$ zu lösen, kann man also gleichwertig umformulieren in die Aufgabe, ein $\mathbf{x}$ mit verschwindendem Residuum zu finden. Man kann leicht nachprüfen, dass die Vektoren $\mathbf{r}^{(k)}$ im obigen Grundschema tatsächlich die jeweiligen Residuen sind: $\mathbf{r}^{(k)} = \mathbf{b} - A\mathbf{x}^{(k)}$. Das Abbruchkriterium des Verfahrens fordert also ein Residuum, das betragsmäßig kleiner als eine vorgegebene Schranke ist.

Vorsicht! Ein kleines Residuum bedeutet nicht automatisch, dass auch der Fehler $\mathbf{x}_{\text{exc}} - \mathbf{x}^{(k)}$ zwischen exakter und genäherter Lösung klein ist. Es gilt beispielsweise, falls A symmetrisch ist, die Abschätzung

$$\frac{\|\mathbf{r}^{(k)}\|_2}{|\lambda_{\max}|} \leq \|\mathbf{x}_{\text{exc}} - \mathbf{x}^{(k)}\|_2 \leq \frac{\|\mathbf{r}^{(k)}\|_2}{|\lambda_{\min}|}$$

wobei $\lambda_{\max}$ und $\lambda_{\min}$ den betragsgrößten bzw. -kleinsten Eigenwert von A bezeichnen. Wenn also $\lambda_{\min}$ nahe Null liegt, sagt ein kleines Residuum noch nicht viel über die Größe des Fehlers aus.

Ebensowenig kann man aus der Kleinheit der Korrekturen $\mathbf{d}^{(k+1)} = \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}$ unmittelbar auf die Kleinheit des Fehlers schließen. Moderne iterative Verfahren können allerdings praktischerweise Näherungen für die Eigenwerte mit geringem Zusatzaufwand mitrechnen und somit verlässliche Schranken für den Fehler liefern.

4.3.2 Konvergenzbeschleunigung durch Minimieren des Residuums

Häufig beobachtet man beim obigen Grundschema, dass sich die Vektoren $\mathbf{d}^{(k)}$, um die sich die Näherungsvektoren pro Iterationsschritt ändern, zwar in eine günstige Richtung zeigen, aber nicht die passende Länge haben. Anstatt den Näherungsvektor $\mathbf{x}^{(k)}$ pro Iterationsschritt nur um den Vektor $\mathbf{d}^{(k+1)}$ zu korrigieren, kann man daher versuchen, gleich ein Vielfaches ω dieser Korrektur anzubringen. (Eine ähnliche Idee verwendet auch schon das SOR-Verfahren.)

Wenn sich der Näherungsvektor beim Schritt von k nach $k+1$ um $\omega\mathbf{d}^{(k+1)}$ ändert, dann lässt sich leicht nachrechnen, dass sich der Restvektor um $-\omega A\mathbf{d}^{(k+1)}$ ändert. Man ändert also das Grundschema, setzt

$$\begin{aligned}\mathbf{x}^{(k+1)} &= \mathbf{x}^{(k)} + \omega\mathbf{d}^{(k+1)}\\ \mathbf{r}^{(k+1)} &= \mathbf{r}^{(k)} - \omega A\mathbf{d}^{(k+1)}\end{aligned}$$

und wählt ω in jedem Schritt so, dass der Betrag $\|\mathbf{r}^{(k+1)}\|$ dadurch möglichst klein wird. Wie geht das? Mit der üblichen Vorgehensweise, wenn man einen Extremwert sucht: Differenzieren und Nullsetzen der Ableitung. (Es bedeutet hier$\|\cdot\|$ immer die 2-Norm, die euklidische Länge eines Vektors.)

Wir arbeiten der Einfachheit halber mit dem Betragsquadrat von $\mathbf{r}^{(k+1)}$. Es lässt sich als Funktion von ω schreiben:

$$\begin{aligned}\|\mathbf{r}^{(k+1)}\|^2 &= (\mathbf{r}^{(k+1)} \cdot \mathbf{r}^{(k+1)})\\ &= \Big((\mathbf{r}^{(k)} - \omega A\mathbf{d}^{(k+1)}) \cdot (\mathbf{r}^{(k)} - \omega A\mathbf{d}^{(k+1)})\Big)\\ &= \Big(\mathbf{r}^{(k)} \cdot \mathbf{r}^{(k)} - 2\omega(\mathbf{r}^{(k)} \cdot A\mathbf{d}^{(k+1)}) + \omega^2(A\mathbf{d}^{(k+1)} \cdot A\mathbf{d}^{(k+1)})\Big)\end{aligned}$$

Die einzelnen inneren Produkte sind hier konstante skalare Größen. Differenzieren nach ω und Nullsetzen der Ableitung liefert

$$\begin{aligned}0 &= \frac{d}{d\omega}\|\mathbf{r}^{(k+1)}\|^2\\ &= \frac{d}{d\omega}\Big(\mathbf{r}^{(k)} \cdot \mathbf{r}^{(k)} - 2\omega(\mathbf{r}^{(k)} \cdot A\mathbf{d}^{(k+1)}) + \omega^2(A\mathbf{d}^{(k+1)} \cdot A\mathbf{d}^{(k+1)})\Big)\\ &= -2(\mathbf{r}^{(k)} \cdot A\mathbf{d}^{(k+1)}) + 2\omega(A\mathbf{d}^{(k+1)} \cdot A\mathbf{d}^{(k+1)}) \quad \text{, daraus}\\ \omega &= \frac{\mathbf{r}^{(k)} \cdot A\mathbf{d}^{(k+1)}}{A\mathbf{d}^{(k+1)} \cdot A\mathbf{d}^{(k+1)}}\end{aligned}$$

4.3.3 Konvergenzbeschleunigung durch orthogonale Suchrichtungen

Wir setzen nun also

$$\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - \omega A\mathbf{d}^{(k+1)}$$

wobei ω so optimal gewählt ist, dass der Betrag von $\mathbf{r}^{(k+1)}$ minimal wird. Das heißt, jede weitere Korrektur des Residuums in Richtung $A\mathbf{d}^{(k+1)}$ kann nur eine Verschlechterung bringen. Falls im nächsten Iterationsschritt

$$\mathbf{r}^{(k+2)} = \mathbf{r}^{(k+1)} - \omega A\mathbf{d}^{(k+2)}$$

die Korrektur $A\mathbf{d}^{(k+2)}$ eine Komponente in Richtung $A\mathbf{d}^{(k+1)}$ enthält, tritt aber genau das ein.

Daher: Ist das Residuum entlang einer Richtung bereits minimiert, dann darf es entlang dieser Richtung nicht mehr korrigiert werden. Wir brauchen also ein Verfahren, das aus der Residuums-Korrektur $A\mathbf{d}^{(k+2)}$ die unerwünschte Komponente in Richtung $A\mathbf{d}^{(k+1)}$ herausnimmt. Das läßt sich durch *Orthogonalisierung* erreichen.

Seien $\mathbf{p}$ und $\mathbf{q}$ zwei Vektoren $\neq 0$. Die Komponente von $\mathbf{p}$ in Richtung von $\mathbf{q}$ ist gegeben durch

$$\left(\frac{\mathbf{p}\cdot\mathbf{q}}{\mathbf{q}\cdot\mathbf{q}}\right)\mathbf{q}$$

Der Vektor

$$\mathbf{p}-\left(\frac{\mathbf{p}\cdot\mathbf{q}}{\mathbf{q}\cdot\mathbf{q}}\right)\mathbf{q}$$

enthält also keine Komponente mehr in Richtung $\mathbf{q}$, steht also orthogonal auf $\mathbf{q}$. (Sonderfall: wenn $\mathbf{p}$ ein skalares Vielfaches von $\mathbf{q}$ ist, liefert diese Rechnung den Nullvektor.)

In dieser Weise lassen sich aus einem Vektor $\mathbf{p}$ auch sukzessive alle Komponenten in Bezug auf ein System von Vektoren herausnehmen.

Orthogonalisieren
Gegeben m von 0 verschiedene Vektoren $\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_m \in \mathbb{R}^n$ und ein Vektor $\mathbf{p} \in \mathbb{R}^n$. Dieser Algorithmus entfernt aus $\mathbf{p}$ alle Komponenten in Richtung $\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_m$.

Für $i = 1 \ldots m$
  rechne inneres Produkt $r_i = \mathbf{p}\cdot\mathbf{q}_i / \mathbf{q}_i\cdot\mathbf{q}_i$
  subtrahiere Komponente: ersetze $\mathbf{p} = \mathbf{p} - r_i\mathbf{q}_i$

P. K. W. Vinsome, damals bei einer Erdölgesellschaft angestellt, veröffentlichte 1976 ein Verfahren, ORTHOMIN, welches alle diese Ideen (Präkonditionierun, Minimierung, Orthogonalisierung) verwendet. Inzwischen wurde eine Fülle von Verfahren entwickelt, die auf ähnlichen Prinzipien beruhen und heute alle als *Krylov-Unterraum-Verfahren* bezeichnet werden.

Für symmetrisch positiv definite Matrizen wurde aus diesen Ideen schon früher ein besonders elegantes und effizientes Verfahren entwickelt, die Methode der konjugierten Gradienten (Hestenes und Stiefel, 1952). Sie ist das Standardverfahren zur iterativen Lösung schwach besetzter, symmetrisch positiv definiter Systeme.

Für unsymmetrische Matrizen sind GMRES (für *generalized minimal residual method*), BiCG (für *biconjugate gradients*) oder CGS (für *conjugate gradient squared*) gängige Verfahren.

5 Überbestimmte Systeme

Ein lineares Gleichungssystem $A\mathbf{x} = \mathbf{b}$ mit mehr Gleichungen als Unbekannten heißt *überbestimmt*.

In so einem Fall ist A eine rechteckige $n \times m$-Matrix mit $n > m$, hat also mehr Zeilen als Spalten. In der Regel hat ein solches System keine exakte Lösung, aber eine eindeutig bestimmte „am wenigsten falsche“ *Ausgleichslösung nach der Methode der kleinsten Quadrate*.

Regelfall: m linear unabhängige Spalten in A, $m+1$ linear unabhängige Spalten in der erweiterten Koeffizientenmatrix $[A, \mathbf{b}]$.
Sonderfälle:

- $\operatorname{rank} A = \operatorname{rank}[A, \mathbf{b}] = m \rightarrow$ eindeutige exakte Lösung.

Systeme ohne vollen Spaltenrang; die nachfolgend beschriebenen Normalengleichungen sind nicht eindeutig lösbar.

- $\operatorname{rank} A = \operatorname{rank}[A, \mathbf{b}] < m \rightarrow$ unendlich viele exakte Lösungen;
- $\operatorname{rank} A < \operatorname{rank}[A, \mathbf{b}] < m \rightarrow$ keine exakte Lösung, unendlich viele gleichberechtigte Ausgleichslösungen.

Überbestimmte Systeme entstehen beispielsweise bei der Auswertung von Messergebnissen, wenn mehr Messungen vorliegen als Parameter gesucht sind.

5.1 Normalengleichungen

Überbestimmte Systeme
Ausgleichslösung nach der *Methode der kleinsten Quadrate*: suche jenes $\mathbf{x}$, für das die euklidische Norm des Residuenvektors

$$\mathbf{r} = \mathbf{b} - A\mathbf{x}$$

minimal wird. Führt auf die *Normalengleichungen*

$$A^T A\mathbf{x} = A^T \mathbf{b}$$

Herleitung 1: Differenzieren von $(\|\mathbf{r}\|_2)^2 = \mathbf{r}^T \cdot \mathbf{r}$ nach den einzelnen Komponenten von $\mathbf{x}$; Nullsetzen der Ableitung.

Herleitung 2: Nehmen wir an, $\hat{\mathbf{x}}$ sei ein Vektor, der $A^T(\mathbf{b} - A\hat{\mathbf{x}}) = 0$ (also die Normalengleichungen) erfüllt. Wir zeigen nun: für jeden anderen Vektor $\mathbf{x} \neq \hat{\mathbf{x}}$ gilt

$$\|\mathbf{b} - A\mathbf{x}\|_2 \geq \|\mathbf{b} - A\hat{\mathbf{x}}\|_2 .$$

Das heißt, für keinen anderer Vektor gibt es einen kleineres Residuum[10]. In diesem Sinn ist $\hat{\mathbf{x}}$ eine optimale Ausgleichslösung des Systems $A\mathbf{x} = \mathbf{b}$.

Beweis: Man setze $\hat{\mathbf{r}} = \mathbf{b} - A\hat{\mathbf{x}}$ und $\mathbf{r} = \mathbf{b} - A\mathbf{x}$. Dann können wir schreiben

$$\mathbf{r} = \mathbf{b} - A\mathbf{x} = (\mathbf{b} - A\hat{\mathbf{x}}) + A(\hat{\mathbf{x}} - \mathbf{x}) = \hat{\mathbf{r}} + A(\hat{\mathbf{x}} - \mathbf{x})$$

[10] Im Regelfall sind alle Spaltenvektoren in A linear unabhängig. Dann gilt sogar strikt $\|\mathbf{b} - A\mathbf{x}\|_2 > \|\mathbf{b} - A\hat{\mathbf{x}}\|_2$ und $\hat{\mathbf{x}}$ ist die eindeutig bestimmte beste Ausgleichslösung.

Damit berechnen wir das innere Produkt $\mathbf{r}^T \cdot \mathbf{r}$ und den Zusammenhang mit $\hat{\mathbf{r}}^T \cdot \hat{\mathbf{r}}$.

$$\begin{aligned}\mathbf{r}^T \cdot \mathbf{r} &= (\hat{\mathbf{r}} + A(\hat{\mathbf{x}} - \mathbf{x}))^T \cdot (\hat{\mathbf{r}} + A(\hat{\mathbf{x}} - \mathbf{x})) \\ &= \hat{\mathbf{r}}^T \cdot \hat{\mathbf{r}} + \hat{\mathbf{r}}^T \cdot (A(\hat{\mathbf{x}} - \mathbf{x})) + (A(\hat{\mathbf{x}} - \mathbf{x}))^T \cdot \hat{\mathbf{r}} + (A(\hat{\mathbf{x}} - \mathbf{x}))^T \cdot (A(\hat{\mathbf{x}} - \mathbf{x})) \\ &= \hat{\mathbf{r}}^T \cdot \hat{\mathbf{r}} + (\hat{\mathbf{r}}^T A) \cdot (\hat{\mathbf{x}} - \mathbf{x})) + (\hat{\mathbf{x}} - \mathbf{x})^T \cdot (A^T \hat{\mathbf{r}}) + (A(\hat{\mathbf{x}} - \mathbf{x}))^T \cdot (A(\hat{\mathbf{x}} - \mathbf{x}))\end{aligned}$$

Weil $\hat{\mathbf{x}}$ laut Voraussetzung die Normalengleichungen erfüllt, ist $A^T \hat{\mathbf{r}} = 0$, ebenso gilt $\hat{\mathbf{r}}^T A = 0$. Es bleibt also

$$\mathbf{r}^T \cdot \mathbf{r} = \hat{\mathbf{r}}^T \cdot \hat{\mathbf{r}} + (A(\hat{\mathbf{x}} - \mathbf{x}))^T \cdot (A(\hat{\mathbf{x}} - \mathbf{x})) \; .$$

Der letzte Term ist (als inneres Produkt eines Vektors mit sich selbst) immer ≥ 0[11]. Damit ist die Aussage bewiesen.

Herleitung 3: Geometrische Veranschaulichung im Fall $\mathbf{x} \in \mathbb{R}^2$ und A eine 3×2-Matrix. Die Vektoren aus der Menge$\{A\mathbf{x} : \mathbf{x} \in \mathbb{R}^2\}$ spannen (wenn die Spalten von A linear unabhängig sind) eine Ebene im Raum auf. Es soll also $A\hat{\mathbf{x}}$ jener Punkt (Ortsvektor) auf der Ebene sein, der von $\mathbf{b}$ minimalen Abstand hat. Der kleinstmögliche Abstand ist der Normalabstand. Der Residuumsvektor $\mathbf{b} - A\hat{\mathbf{x}}$ steht somit normal auf die Ebene, also auf alle Vektoren der Form $A\mathbf{x}$. Aus

$$\forall \mathbf{x}: \quad (A\mathbf{x})^T \cdot (\mathbf{b} - A\hat{\mathbf{x}}) = \mathbf{x}^T \cdot (A^T(\mathbf{b} - A\hat{\mathbf{x}})) = 0$$

folgt $A^T(\mathbf{b} - A\hat{\mathbf{x}}) = 0$, weil nur der Nullvektor auf alle Vektoren $\mathbf{x} \in \mathbb{R}^2$ orthogonal sein kann.

5.2 Weitere Verfahren

Die Lösung überbestimmeter Systeme über die Normalengleichungen ist das klassische Standardverfahren, aber deswegen nicht unbedingt der günstigste Algorithmus. Es ist bloß das einzige, das sich bei kleinen Beispielen mit vertrauten Methoden (Matrixmultiplikation, Elimination) händisch durchrechnen lässt. Programmpakete und Rechenumgebungen (wie MATLAB) verwenden zumeist die QR-Zerlegung. Bei Systemen *ohne vollen Spaltenrang* (seien sie überbestimmt oder nicht) ist die Singulärwertzerlegung (*singular value decomposition, SVD*) vorteilhaft. In diesem Fall gibt es nämlich unendlich viele Lösungen (exakt oder im Sinn der kleinsten Quadrate), und SVD liefert automatisch die betragskleinste. Aus der QR-Zerlegung lassen sich hingegen die Lösungen mit den meisten Null-Komponenten ablesen.

5.3 Beispiele zu überbestimmten Systemen

Gegeben sind drei lineare Gleichungen in zwei Unbekannten,

$$\begin{array}{rcrcl} 2x & + & y & = & 19 \\ -4x & + & 4y & = & 13 \\ 4x & - & y & = & 17 \end{array}$$

Das Gleichungssystem in Matrixschreibweise lautet

$$A\mathbf{x} = \mathbf{b} \text{ mit } A = \begin{bmatrix} 2 & 1 \\ -4 & 4 \\ 4 & -1 \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 19 \\ 13 \\ 17 \end{bmatrix}$$

[11] Sind alle Spaltenvektoren in A linear unabhängig, dann ist mit $\mathbf{x} \neq \hat{\mathbf{x}}$ auch $A(\hat{\mathbf{x}} - \mathbf{x}) \neq 0$; der letzte Term ist dann echt > 0.

Methode der Normalengleichungen

Bilde $A^T \cdot A$ und $A^T\mathbf{b}$:

$$A^T \cdot A = \begin{bmatrix} 2 & -4 & 4 \\ 1 & 4 & -1 \end{bmatrix} \cdot \begin{bmatrix} 2 & 1 \\ -4 & 4 \\ 4 & -1 \end{bmatrix} = \begin{bmatrix} 36 & -18 \\ -18 & 18 \end{bmatrix} = 18 \begin{bmatrix} 2 & -1 \\ -1 & 1 \end{bmatrix}$$

$$A^T\mathbf{b} = \begin{bmatrix} 2 & -4 & 4 \\ 1 & 4 & -1 \end{bmatrix} \cdot \begin{bmatrix} 19 \\ 13 \\ 17 \end{bmatrix} = \begin{bmatrix} 54 \\ 54 \end{bmatrix} = 18 \begin{bmatrix} 3 \\ 3 \end{bmatrix}$$

Die Normalengleichungen lauten daher (schon durch 18 gekürzt):

$$\begin{bmatrix} 2 & -1 \\ -1 & 1 \end{bmatrix} \mathbf{x} = \begin{bmatrix} 3 \\ 3 \end{bmatrix}$$, oder in ausgeschriebener Form $$\begin{array}{rcrcl} 2x & - & y & = & 3 \\ -x & + & y & = & 3 \end{array}$$

Addition der beiden Gleichungen liefert sofort $x = 6$, und daraus durch Einsetzen $y = 9$.

Minimaler Fehler in verschiedenen Normen

Die Normalengleichungen liefern $\mathbf{x} = [6; 9]$. Einsetzen in die ursprünglichen Gleichungen zeigt aber: diese „Lösung“ erfüllt keine der drei Gleichungen exakt. Der Fehlervektor $\mathbf{r} = \mathbf{b} - A\mathbf{x}$ lautet

$$\begin{bmatrix} 19 \\ 13 \\ 17 \end{bmatrix} - \begin{bmatrix} 2 & 1 \\ -4 & 4 \\ 4 & -1 \end{bmatrix} \begin{bmatrix} 6 \\ 9 \end{bmatrix} = \begin{bmatrix} -2 \\ 1 \\ 2 \end{bmatrix}$$

Der Fehlervektor hat Länge 3, und das ist die kleinstmögliche Länge. Gemeint ist hier die euklidische Länge, also die Zweinorm.

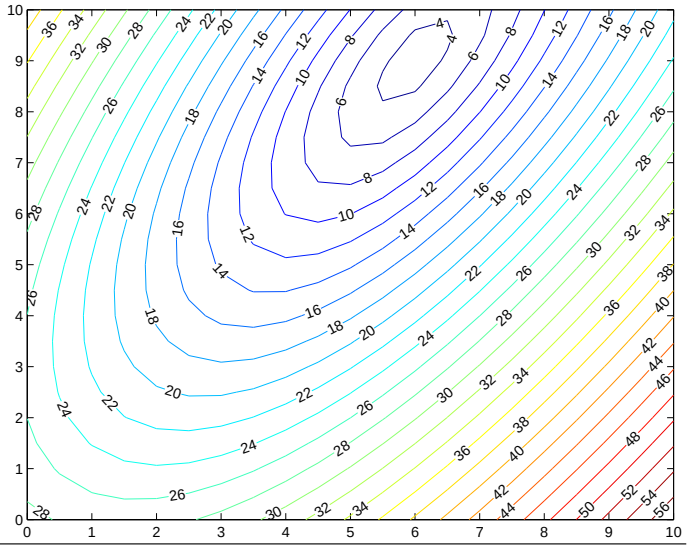
Ändert man beispielsweise den Vektor $\mathbf{x}$ geringfügig auf

$$\mathbf{x} = \begin{bmatrix} 6 \\ 8.8 \end{bmatrix} = \begin{bmatrix} 6 \\ 44/5 \end{bmatrix}$$

so lautet der Fehlervektor

$$\mathbf{r} = \mathbf{b} - A\mathbf{x} = \begin{bmatrix} -9/5 \\ 9/5 \\ 9/5 \end{bmatrix} = \begin{bmatrix} -1.8 \\ 1.8 \\ 1.8 \end{bmatrix}$$

mit euklidischer Länge $\|\mathbf{r}\|_2 = \frac{9}{5}\sqrt{3} = 3.1177$, also deutlich über dem Optimum.



Höhenschichtlinien der Länge des Fehlervektors in der Zweinorm. Das Fehlerminimum bei $[6; 9]$ ist deutlich zu erkennen.

In der Unendlichnorm (Maximum der Komponentenbeträge) ist der Fehler nun allerdings geringer: 1.8 statt 2.

Versuchen wir noch einen anderen Vektor $\mathbf{x}$:

$$\mathbf{x} = \begin{bmatrix} 6.15 \\ 9.4 \end{bmatrix}, \quad \text{zugehöriger Fehlervektor } \mathbf{r} = \mathbf{b} - A\mathbf{x} = \begin{bmatrix} -2.7 \\ 0 \\ 1.8 \end{bmatrix}$$

$$\|\mathbf{r}\|_2 = 3.2450, \quad \|\mathbf{r}\|_\infty = 2.7$$

Dieser Fehlervektor liegt also sowohl in der 2- als auch der ∞-Norm über den beiden vorherigen. Misst man aber die Summe der Absolutbeträge (die Einsnorm), so ergibt sich hier der Wert 4.5. Die beiden vorigen Fehlervektoren hatten Einsnormen von 5 bzw. 5.4. In der Einsnorm ist also die hier gewählte Lösung optimal.

Lösung durch QR-Zerlegung von A

$$A = Q \cdot R\,, \quad \begin{bmatrix} 2 & 1 \\ -4 & 4 \\ 4 & -1 \end{bmatrix} = \begin{bmatrix} -1/3 & 2/3 & -2/3 \\ 2/3 & 2/3 & 1/3 \\ -2/3 & 1/3 & 2/3 \end{bmatrix} \cdot \begin{bmatrix} -6 & 3 \\ 0 & 3 \\ 0 & 0 \end{bmatrix}$$

Das transformierte Gleichungssystem $R\mathbf{x} = Q^T\mathbf{b}$ ist ebenfalls überbestimmt und lautet ausgeschrieben

$$\begin{array}{rcrcl} -6x & + & 3y & = & -9 \\ & & 3y & = & 27 \\ & & 0 & = & 3 \end{array}$$

Wenn man die ersten beiden Gleichungen exakt löst (wegen Dreiecksform einfach durch Rücksubstitution), liefern sie keinen Beitrag zum Residuum. Die letzte Gleichung hängt nicht von $\mathbf{x}$ ab. Keine Wahl von $\mathbf{x}$ kann den Beitrag dieser Gleichungen zum Residuum ändern.

Daher ist die aus den ersten beiden Gleichungen bestimmte Lösung optimal für das transformierte System. Weil die Transformation die Norm des Fehlervektors nicht beeinflusst (wegen Orthogonalität von Q), ist diese Lösung auch optimale Lösung von $A \cdot \mathbf{x} = \mathbf{b}$.

Weiteres Beispiel: Lösung mit Singulärwertzerlegung

Ein anderes überbestimmtes System lautet

$$A\mathbf{x} = \mathbf{b} \text{ mit } A = \begin{bmatrix} 14 & -2 \\ -4 & 22 \\ 16 & -13 \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} -80 \\ 40 \\ -145 \end{bmatrix}$$

Die Singulärwertzerlegung von A ist

$$A = U \cdot S \cdot V^T, \begin{bmatrix} 14 & -2 \\ -4 & 22 \\ 16 & -13 \end{bmatrix} = \begin{bmatrix} -1/3 & -2/3 & -2/3 \\ 2/3 & -2/3 & 1/3 \\ -2/3 & -1/3 & 2/3 \end{bmatrix} \cdot \begin{bmatrix} 30 & 0 \\ 0 & 15 \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} -3/5 & -4/5 \\ 4/5 & -3/5 \end{bmatrix}^T$$

Das transformierte System lautet in diesem Fall

$$S \cdot \mathbf{y} = U^T \cdot \mathbf{b}, \begin{bmatrix} 30 & 0 \\ 0 & 15 \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 150 \\ 75 \\ -30 \end{bmatrix}$$

Wegen der Diagonalgestalt von S ist die Optimallösung direkt ablesbar: $\mathbf{y} = [5; 5]$. Die Lösung des Originalsystems erhält man über die Beziehung

$$\mathbf{x} = V \cdot \mathbf{y} = \begin{bmatrix} -3/5 & -4/5 \\ 4/5 & -3/5 \end{bmatrix} \cdot \begin{bmatrix} 5 \\ 5 \end{bmatrix} = \begin{bmatrix} -7 \\ 1 \end{bmatrix}$$

Wenn QR-Zerlegung oder SVD bereits gegeben sind, ist die Lösung von (überbestimmten) Gleichungssystemen relativ einfach. Der eigentliche Arbeitsaufwand steckt im Berechnen der Zerlegungen. Auf die dabei verwendeten Verfahren kann im Rahmen der Vorlesung nicht eingegangen werden.

MATLAB-Befehle

QR-Zerlegung einer Matrix A:	`[Q R]=qr(A)`
SVD, Singulärwertzerlegung $A = U \cdot S \cdot V^T$	`[U S V]=svd(A)`
Lösung eines überbestimmten Gleichungssystems	
... durch QR-Zerlegung	`A\b`
... durch SVD	`pinv(A)*b`

5.4 Anpassen eines linearen Modells (einer Ausgleichsebene)

Dieser Abschnitt erläutert anhand eines weiteren Beispiels die Lösung überbestimmter Systeme. Vom Thema her überschneidet er sich mit Kapitel 6; dort werden weitere Methoden zur Approximation von Daten behandelt. Was das Beispiel hier illustrieren soll: die unterschiedlichen Größenordnungen bei den Zwischenergebnissen und der daraus resultierene Einfluss der Rundungsfehler im Vergleich zwischen der Methode der Normalengleichung und der QR-Zerlegung.

Für die Blies (einen Nebenfluss der Saar) sollen die Hochwasserstände am Pegel Neunkirchen aus den Wasserständen des Pegels Ottweiler und des Pegels Hangard vorhergesagt werden. Es liegen die Daten der Scheitelwasserstände von 12 Winterhochwässern aus den Jahren 1963–1971 vor:

Wasserstand in cm												
Neunkirchen y	172	309	302	283	443	298	319	419	361	267	337	230
Ottweiler x_1	93	193	187	174	291	184	205	260	212	169	216	144
Hangard x_2	120	258	255	238	317	246	265	304	292	242	272	191

Daten-Quelle: U. Maniak, Hydrologie und Wasserwirtschaft, Springer, 1988

Wir unterstellen den Daten das lineare Modell $a_0 + a_1x_1 + a_2x_2 = y$. In diesem Ansatz sind a_0, a_1 und a_2 unbekannte Koeffizienten, die aus den zwölf gegebenen Werte-Tripeln möglichst gut bestimmt werden sollen. Geometrisch lassen sich die Tripel $(x_1|x_2|y)$ als Punkte im Raum interpretieren. Das lineare Modell entspricht dann einer Ebene, die möglichst gut an die Datenpunkte angepasst werden soll.

Einsetzen der Daten in den Ansatz liefert das Gleichungssystem

$$\begin{array}{rcrcrcr} a_0 & + & 93a_1 & + & 120a_2 & = & 172 \\ a_0 & + & 193a_1 & + & 258a_2 & = & 309 \\ a_0 & + & 144a_1 & + & 191a_2 & = & 230 \\ \vdots & & & & & & \vdots \\ a_0 & + & 187a_1 & + & 255a_2 & = & 302 \end{array}$$

Es liegt somit ein überbestimmtes Gleichungssystem $A\mathbf{x} = \mathbf{b}$ vor, mit 12×3-Matrix A und rechter Seite $\mathbf{b}$,

$$A = \begin{bmatrix} 1 & 93 & 120 \\ 1 & 193 & 258 \\ 1 & 187 & 255 \\ \vdots & \vdots & \vdots \\ 1 & 144 & 191 \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} 172 \\ 309 \\ 302 \\ \vdots \\ 230 \end{bmatrix}$$

Der klassische Weg zur Ausgleichslösung nach der Methode der kleinsten Quadrate führt über die Normalengleichungen $A^T \cdot A\mathbf{x} = A^T\mathbf{b}$, in diesem Beispiel

$$\begin{bmatrix} 12 & 2328 & 3000 \\ 2328 & 480202 & 609985 \\ 3000 & 609985 & 780572 \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 3740 \\ 766996 \\ 975996 \end{bmatrix}$$

Es treten recht unterschiedlich große Zahlen in diesem System auf (Grössenordnung 10^1 bis 10^6). Das ist typisch für Normalengleichungen und ein Indiz dafür, dass dieses System empfindlich gegenüber Rundungsfehlern ist. Numerisch günstiger, aber praktisch nur rechnergestützt durchführbar ist die QR-Zerlegung. Das transformierte System $R\mathbf{x} = Q^T\mathbf{b}$ lautet hier

$$\begin{bmatrix} -3.4641 & -672.0357 & -866.0254 \\ 0 & -169.0266 & -165.5656 \\ 0 & 0 & -56.2141 \\ 0 & 0 & 0 \\ \vdots & \vdots & \vdots \\ 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} -1079.6 \\ -245.14 \\ -7.2659 \\ -2.4029 \\ \vdots \\ -11.003 \end{bmatrix}$$

Die ersten drei Gleichungen liegen als System in oberer Dreiecksform vor und sind durch Rücksubstitution exakt auflösbar. Die restlichen neun Gleichungen von der Form $0 = -2.4029, \ldots$ sind klarerweise unlösbar. Sie liefern den Beitrag zum Restvektor.

locker gesagt: QR-Zerlegung transformiert ein überbestimmtes System in ein exakt lösbares System in Dreiecksform und einen unlösbaren Rest. Ignorieren der unlösbaren Gleichungen liefert die bestmögliche Lösung im Sinn der kleinsten Quadrate

Anmerkung: Es ist zwar unmittelbar einsichtig, dass obiges Rezept die Lösung mit betragsmäßig minimalem Restvektor für das *transformierte* System liefert. Allerdings hat das *Originalsystem* einen anderen Restvektor. Die eigentliche Pointe des Verfahrens ist, dass die Transformation von dem einen zum anderen Restvektor durch Multiplikation mit einer *orthogonalen* Matrix passiert. Multiplikation mit einer orthogonalen Matrix lässt den Betrag eines Vektors unverändert. Daher hat auch der Restvektor des Originalsystems minimalen Betrag.

In MATLAB verwendet der Standard-Gleichungslöser-Befehl `A\b` im Falle eines überbestimmten Systems automatisch das QR-Verfahren und liefert hier

```
>> A\b
ans =
   22.5505
    1.3237
    0.1293
```

(Die aufwändigere Lösung mit Singulärwertzerlegung (MATLAB: `pinv(A)*b`) liefert hier das gleiche Resultat.)

Das Modell zur Vorhersage der Hochwasser-Scheitelwerte lautet somit

$$y = 22.5505 + 1.3237x_1 + 0.1293x_2$$

Man erkennt, dass die x_2-Daten gut zehnfach weniger Einfluss auf das Vorhersagemodell haben (weil der entsprechende Koeffizient im linearen Modell nur 0.1293 im Vergleich zu 1.3237 beträgt). Eine mögliche Interpretation wäre, dass der Wasserstand in Neunkirchen hauptsächlich vom Pegel Ottweiler und nicht wirklich vom Pegel Hangard abhängt.

Wie vertrauenswürdig ist dieses Modell? Einsetzen der Datenpunkte liefert den Restvektor.

Messwert y	172	309	302	283	443	298	319	419	361	267	337	230
Modellvorhersage	161	311	303	284	449	298	328	406	341	278	344	238
Residuum	11	-2	-1	-1	-6	0	-9	13	20	-11	-7	-8

Der mittlere absolute Fehler liegt bei 7,3 cm, der maximale Fehler allerdings bei 20 cm.

Eine genauere Beurteilung des Modells im Hinblick auf

- die Richtigkeit des angenommenen linearen Zusammenhangs,
- Einfluss der einzelnen Modellgrößen
- Wahrscheinlichkeit, dass der Prognosewert mit gewisser Genauigkeit zutrifft
- mögliche Ausreißer

erfordert Methoden der multivariaten Statistik und der Regressionsanalyse (und wohl auch eine größere Datenmenge).

6 Approximation von Daten, Ausgleichsrechnung

6.1 Lineare Datenmodelle

Ein Beispiel dazu hat schon Kapitel 5.4 gebracht. Die Vorlesungsfolien und die Übungen bringen weitere Beispiele dazu. Das best-angepasste Modell ergibt sich dabei immer aus der kleinste-Quadrate-Näherung an ein überbestimmtes Gleichungssystem.

Aber nicht immer liefert die Methode der kleinsten Fehlerquadrate eine plausible Anpassung. Einige wenige grob falsche Werte in den Daten („Ausreißer“) können das Ergebnis gewaltig verzerren. Im Kapitel 6.6 wird eine robuste Methode vorgestellt. MATLAB bietet in seinen Toolboxen verschiedene Methoden zur robusten Anpassung (*robust fit*) an.

6.2 Polynomiale Regression

Hier handelt es sich um einen wichtigen Spezialfall der linearen Datenmodelle, für den sich die allgemeinen Formeln etwas vereinfachen.

Eine typische Aufgabe zu diesem Abschnitt könnte lauten: Gegeben sind Messergebnisse für einen Satz von Temperaturwerten T und die entsprechenden Widerstandswerte R eines Temperaturfühlers. Der Zusammenhang zwischen R und T lässt sich näherungsweise in der Form $R = a + bT + cT^2$ beschreiben. Wenn für genau drei (verschiedene) T-Werte Daten vorliegen, lassen sich die drei Parameter a, b und c eindeutig bestimmen. Es ist aber sinnvoll, mehr Messungen durchführen, damit Messfehler der Einzelmessungen nicht so stark ins Gewicht fallen. Die Parameter a, b und c werden dann durch *Ausgleichsrechnung* (man sagt auch *Regression*) bestimmt.

Polynomiale Regression (Ausgleich durch ein Polynom)
Gegeben: $m + 1$ Wertepaare $(x_i, y_i), i = 0, \ldots, m$
Gesucht: $p(x)$, ein Polynom n-ten Grades, $n < m$, so dass die Summe der Fehlerquadrate

$$\sum_{i=0}^{m} (p(x_i) - y_i)^2$$

minimal wird. Locker formuliert: $y = p(x)$ approximiert möglichst gut die Datenpunkte.

Nicht immer ist ein Polynomansatz ein geeignetes Modell, aber oft lässt sich ein scheinbar komplizierteres Modell auf ein polynomiales Modell zurückführen (Beispiele in den Übungen).

Direkter Lösungsweg: Ansatz des Polynoms mit unbestimmten Koeffizienten,

$$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1} + a_n x_n \ .$$

Einsetzen der gegebenen Wertepaare führt auf ein System von $m + 1$ linearen Gleichungen in den $n + 1$ unbekannten Koeffizienten $a_0, a_1, \ldots, a_n$.

Sofern $n < m$, liegt ein *überbestimmtes System* vor.

- klassische Lösung: Näherung nach der Methode der Normalengleichungen: lässt sich bei kleinen Beispielen mit Papier und Stift rechnen; bei großen Datenmengen Gefahr von Rundungsfehlern.

  Die Normalengleichungen sind aber nur dann eindeutig lösbar, wenn in den insgesamt $m + 1$ Wertepaaren mindestens $n + 1$ der x-Werte verschieden sind.

- moderner Lösungsweg: QR-Zerlegung. Praktisch nur am Rechner durchführbar. Bessere Konditionszahl, weniger anfällig für Rundungsfehler.

Dieser Lösungsweg (Ansatz mit unbestimmten Koeffizienten, Aufstellen des überbestimmten Systems, Bilden der Normalengleichungen) lässt sich für die polynomiale Regression etwas abkürzen. Setzt man

$$s_0 = m + 1, \quad t_0 = \sum_{i=0}^{m} y_i$$

und

$$s_k = \sum_{i=0}^{m} x_i^k, \quad t_k = \sum_{i=0}^{m} x_i^k y_i \quad \text{für } k > 0,$$

so lassen sich, wie man leicht herleiten kann, die Normalengleichungen in folgender Gestalt schreiben:

$$\begin{array}{lllllll}
s_0 a_0 & + & s_1 a_1 & + & \ldots\ldots & + & s_n a_n \quad = t_0 \\
s_1 a_0 & + & s_2 a_1 & + & \ldots\ldots & + & s_{n+1} a_n \quad = t_1 \\
 & & & & \ldots\ldots & & \\
s_n a_0 & + & s_{n+1} a_1 & + & \ldots\ldots & + & s_{2n} a_n \quad = t_n
\end{array}$$

Die Normalengleichungen können für größere n ziemlich schlecht konditioniert sein. Bessere Resultate lassen sich in solchen Fällen durch Entwicklung nach orthogonalen Polynomen erzielen.
Verwendet man beispielsweise als Datensatz die Punkte

$$(x, \exp(x)) \quad \text{für } x = 0;\, 0,01;\, 0,02; \ldots; 3,99; 4$$

und approximiert die Daten durch Regressionspolynome verschieden hohen Grades (Rechnung mit vierzehnstelliger Genauigkeit), so wird die Güte der Approximation vorerst mit steigendem Grad besser. Ab dem dreizehnten Grad aber wachsen die Fehler wieder an. Das aus den Normalengleichungen berechnete Polynom 25-ten Grades hat kaum mehr Ähnlichkeit mit der approximierten Funktion. Verwendet man orthogonale Polynome (Tschebyscheff-Polynome), treten diese numerischen Probleme nicht auf.

6.3 Ausgleichsgerade

Ein wichtiger Spezialfall der obigen Problemstellung: An $m + 1$ (mehr als zwei) gegebene Datenpunkte soll eine Gerade mit Gleichung $y = a + bx$ so angepasst werden, dass die Summe der Fehlerquadrate minimal wird.

$$a = \frac{s_2 t_0 - s_1 t_1}{s_0 s_2 - s_1^2}$$

$$b = \frac{s_0 t_1 - s_1 t_0}{s_0 s_2 - s_1^2}$$

Das Finden einer Ausgleichsgeraden wird oft auch als „lineare Regression“ bezeichnet. Das ist aber ein mißverständlicher Terminus, weil er leicht zur Verwechslung mit „linearen Datenmodellen“ führt.

6.4 Nichtlineare Datenmodelle

Dazu gibt es Material auf den Vorlesungsfolien und ausführlicher in den Übungsunterlagen. Kurzfassung: Jacobi-Matrix bilden und iterieren. Im Unterschied zum Newton-Verfahren, das Sie schon kennen, ist das lineare Gleichungssystem mit der Jacobimatrix nun überbestimmt und wird näherungsweise im Sinn der kleinsten Fehlerquadrate gelöst.

Dieses Verfahren wird als *Gauß-Newton-Verfahren* bezeichnet.

Bei nichtlinearen Ausgleichsproblemen gibt es neben dem Gauß-Newton-Verfahren noch weitere Verfahren (z. B. Levenberg-Marquardt-Algorithmus). Damit beschäftigt sich die Optimierung als Teilgebiet der Angewandten Mathematik. Dieses Skript kann nicht näher darauf eingehen.

6.5 Warum „kleinste Quadrate“

Die Methode der kleinsten Quadrate minimiert die euklidische Länge des Residuenvektors. Man kann aber die Größe des Residuenvektors durchaus auch anders messen und entsprechend andere Minimalbedingungen fordern. Wichtige Beispiele: Die Summe der *Absolutbeträge* der Fehler soll minimal werden, oder der *maximale Fehler* soll minimal werden („Minimax-Approximation“). Zwei Gründe sprechen für die kleinsten Quadrate:

- Einfache Herleitung und Durchführung: die Minimalwert-Aufgabe lässt sich mit elementarer Differentialrechnung lösen und führt auf ein einfaches algebraisches Problem

- Statistische Überlegungen: Wenn die Daten mit unabhängigen, zufälligen, normalverteilten Fehlern mit gleicher Standardabweichung behaftet sind, sind kleinste Quadrate in gewissem Sinn optimal (genauer: Die Methode liefert eine *maximum likelihood*-Schätzung der Parameter). Umgekehrt gilt aber: wenn die Fehler in den Daten *nicht* normalverteilt etc. sind, dann sind kleinste Quadrate unter Umständen ziemlich schlecht; siehe unten.

- Weitere statistische Eigenschaften: Wenn man eine Abschätzung für die Genauigkeit der Daten hat, kann man auf die Genauigkeit der berechneten Modellparameter schließen.

  > Angenommen, die Daten sind so skaliert, dass die Varianz der Messfehler gleich 1 ist. Sei $C = (A^T A)^{-1}$ die inverse Matrix des Systems der Normalengleichungen. Die Diagonalelemente von C sind die Varianzen der entsprechenden Modellparameter; die Elemente außerhalb der Hauptdiagonale sind die entsprechenden Kovarianzen.

Und was spricht dagegen?

- Die Methode reagiert empfindlich auf „Ausreißer“ in den Daten. Das Quadrieren der Fehler bestraft grosse Abweichungen streng. Deswegen ist die Methode der kleinsten Quadrate bereit, eine Kurve wild zu verzerren, um ein paar weit außen liegende Datenpunkte auch noch annähernd zu erreichen. Ein paar Ausreißer in ansonsten vernünftigen Daten können so eine völlig unsinnige Approximation bewirken.

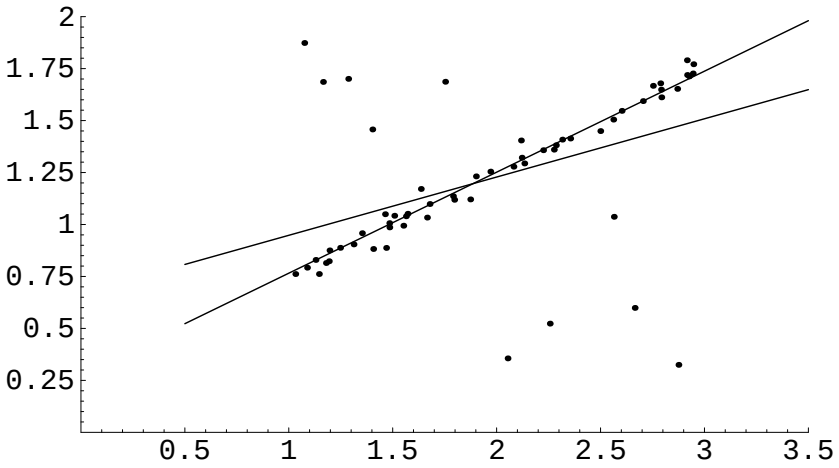


Abbildung 8: Anpassen einer Geraden an Datenpunkte. Die Ausgleichsgerade nach der Methode der kleinsten Quadrate lässt sich von den wenigen Ausreißern stark ablenken. Minimieren des absoluten Fehlers legt eine wesentlich plausiblere Gerade durch die Daten.

6.6 Ausgleichsgerade mit Minimieren der absoluten Fehler (L_1-Norm)

Gegeben: m Wertepaare $(x_i, y_i), i = 1, \ldots, m$
Gesucht: Eine Gerade in der Form $y = a + bx$, so dass die Summe der absoluten Fehler

$$\sum_{i=1}^{m} |p(x_i) - y_i|$$

minimal wird.
Die Lösung lautet: Für gegebenes b ergibt sich a als Median eines Datenfeldes,

$$a = \text{median}\{y_i - bx_i\}$$

Den Parameter b findet man als Lösung der Gleichung

$$0 = \sum_{i=1}^{m} x_i \,\text{sgn}(y_i - a - bx_i)$$

(wobei sgn(0) als Null interpretiert werden soll). Wenn man für a in dieser Gleichung die durch die vorigen Gleichung bestimmte Funktion $a(b)$ einsetzt, bleibt eine Gleichung in einer Unbekannten übrig. Intervallhalbierung (siehe Kapitel 1.7) ist die geeignete Lösungsmethode dafür.

6.7 Ausgleichsgerade mit Minimieren der Normalabstände

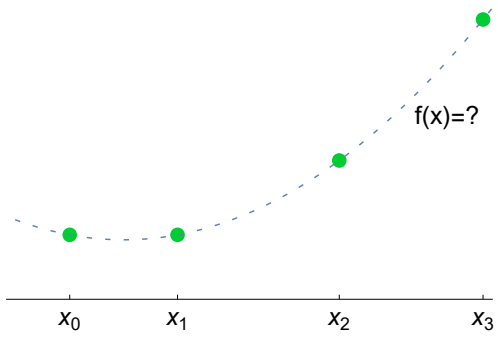
Dazu gibt es unter dem Titel *Total Least Squares* Material auf den Vorlesungsfolien und den Übungsunterlagen.

7 Interpolation

Interpolation, Kurzfassung
Gegeben: Datenpunkte
Gesucht:

- Eine Funktion, die *durch* die gegebenen Datenpunkte verläuft.
- Ein Wert *zwischen* den Datenpunkten
- Trend über den gegebenen Datenbereich hinaus: *Extrapolation*

Anwendungen: Zwischenwerte in Tabellen, „glatte“ Kurven für Graphik...



Sind die Werte einer Funktion f für einen Satz von x-Werten gegeben, also $f(x_i) = y_i$ für $i = 0, 1 \ldots, n$, aber kein analytischer (formelmäßiger) Ausdruck für f, so kann man ersatzweise eine geeignete, möglichst einfache Funktion suchen, die für alle gegebenen x_i-Werte – man nennt sie die *Stützstellen* – die entsprechenden Funktionswerte y_i annimmt.

Mit Hilfe dieser Funktion lassen sich dann leicht Werte an beliebigen anderen Stellen x berechnen. Sucht man einen Funktionswert für ein x zwischen x_0 und x_n, so spricht man von *Interpolation*; liegt x außerhalb des gegebenen Datenbereiches, heißt die Aufgabe *Extrapolation* und ist beträchtlich riskanter (was viele Pandemie-Prognose-Gurus inzwischen bestätigen können).

Beachten Sie den Unterschied zu den Approximationsaufgaben des vorigen Abschnittes: *Interpolation* legt eine Funktion *durch* gegebene Datenpunkte; *Approximation* findet eine Funktion, die sich an die Datenpunkte möglichst gut *annähert*.

Jede Formelsammlung bringt eine Vielzahl von Interpolationsformeln, verbunden mit klingenden Namen von Leuten, die schon lange tot sind (Newton, Bessel, Lagrange...). Nehmen Sie das als Indiz dafür, dass Interpolationsverfahren früher von großer Bedeutung waren. Zu einer Zeit, als mathematische Funktionen aus Tabellenwerken entnommen werden mussten, waren etwa trigonometrische Berechnungen ständig mit Interpolation in Tabellen verbunden. Winkelfunktionen bietet aber inzwischen jeder bessere Taschenrechner, und auch weniger elementare Funktionen stehen in gängigen Computerprogrammen zur Verfügung. Damit ist die Wichtigkeit der Interpolation in diesem Bereich ziemlich gesunken.
Interpolation und Extrapolation sind noch wichtig als Hilfsmittel zur Lösung anderer mathematischer Probleme (numerische Integration, numerisches Lösen von Differentialgleichungen). Sehr wichtig ist Interpolation bei Computergraphik und computerunterstütztem Design: Lara Croft existiert als ein Satz von Datenpunkten; damit sie am Bildschirm gut zur Geltung kommt, zeichnet der Rechner möglichst attraktive Kurven durch die Daten. Meist handelt es sich dabei um Spline-Interpolationsverfahren.

7.1 Polynomiale Interpolation

Durch zwei Punkte geht genau eine Gerade. Durch drei beliebige Punkte lässt sich eindeutig eine Parabel legen. Allgemein: Durch $n+1$ (verschiedene) Punkte ist ein Polynom n-ten Grades eindeutig bestimmt.

Polynom-Interpolation, Aufgabenstellung:
Gegeben: $n+1$ Wertepaare $(x_i, y_i), i = 0, \ldots, n$, wobei die x_i paarweise verschieden sind.
Gesucht: das eindeutig bestimmte Polynom n-ten Grades p, das durch die gegebenen Datenpunkte verläuft:

$$p(x_i) = y_i \quad \text{für } i = 0, \ldots, n$$

Oder: gesucht ist nicht die Form des interpolierenden Polynoms selbst, sondern dessen Wert an einer gegebenen Stelle x.

Lösung mit der Holzhammermethode: Ansatz des Polynoms mit unbestimmten Koeffizienten in der Standardform,

$$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1} + a_n x^n \ .$$

Einsetzes der gegebenen Wertepaare führt auf ein System von $n+1$ linearen Gleichungen in den $n+1$ unbekannten Koeffizienten $a_0, a_1, \ldots, a_n$.

In Matrix-Vektor-Form haben die Gleichungen $p(x_i) = y_i$ eine einfache Struktur:

$$\begin{bmatrix} 1 & x_0 & x_0^2 & x_0^3 & \ldots & x_0^n \\ 1 & x_1 & x_1^2 & x_1^3 & \ldots & x_1^n \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ 1 & x_n & x_n^2 & x_n^3 & \ldots & x_n^n \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}$$

Eine solche Matrix, in deren Spalten der Reihe nach die x_i hoch $0, 1, 2 \ldots$ stehen, heißt *Vandermonde-Matrix*.

Die gute Nachricht: Falls alle x_i-Werte verschieden sind, ist diese Matrix nicht singulär, das Gleichungssystem eindeutig lösbar.

Die schlechten Nachrichten: Die Vandermonde-Matrix kann sehr hohe Konditionszahl haben. Bei Polynomen höheren Grades entstehen gravierende Rundungsfehler. Der Rechenaufwand zur Lösung des Gleichungssystems steigt mit $O(n^3)$. Alle Numerik-Lehrbücher raten alternativ zu effizienteren Verfahren. Aber für kleine n und einen schnellen Computer sind diese Argumente nicht relevant, sogar MATLAB's Befehl `polyfit` verwendet diesen Ansatz.

Dabei lässt sich das Interpolationspolynom ganz einfach und elegant explizit hinzuschreiben. Die Formel kennen Sie aus den Mathematik-Einführungskursen:

Lagrangesche Interpolationsformel
Das Interpolationspolynom durch die $n+1$ Wertepaare (x_i, y_i), $i = 0, \ldots, n$ ist gegeben durch

$$p(x) = L_0(x)y_0 + L_1(x)y_1 + \ldots + L_n(x)y_n,$$

wobei

$$L_i(x) = \frac{(x-x_0)(x-x_1)\ldots(x-x_{i-1})(x-x_{i+1})\ldots(x-x_n)}{(x_i-x_0)(x_i-x_1)\ldots(x_i-x_{i-1})(x_i-x_{i+1})\ldots(x_i-x_n)}$$

Diese Darstellung ist für eine Reihe weiterführender Überlegungen wichtig, aber zum Programmieren oder zur numerischen Auswertung nicht die günstigste.

Neben dem Ansatz mit der Vandermonde-Matrix und der Lagrange-Formel lernen Sie gleich noch den Ansatz nach Newton kennen.

Welchen Rechenweg sollen Sie wählen? *Alle Wege führen zum Polynom*, in diesem Fall zu *demselben* Interpolationspolynom, bloß in unterschiedlicher Schreibweise. Warum sollen Sie dann noch weitere Rechenwege kennenlernen? *Der Weg ist das Ziel*: neben der praktischen Anwendung stoßen Sie auf einen mathematisch tiefgründigen Zusammenhang: Polynome verhalten sich in vielfacher Weise wie Vektoren; für Mathematiker *sind* sie Vektoren (Elemente eines Vektorraums). Das würde hier zu weit führen, aber wir halten fest:

Polynome in verschiedenen Basis-Darstellungen Ein Polynom $p(x)$ lässt sich auf unterschiedliche Art als Summe von Termen der Form *Koeffizient mal Basisfunktion* anschreiben.

- Standardbasis

  $$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

  p ist Linearkombination der Basis-Polynome $1,\ x,\ x^2, \ldots, x^n$

- Lagrange-Basis

  $$p(x) = y_0 L_0(x) + y_1 L_1(x) + \cdots + y_n L_n(x)$$

  p ist Linearkombination der Lagrange-Polynome $L_0, L_1, \ldots, L_n$

- Newton-Basis

  $$p(x) = c_0 + c_1(x - x_0) + c_2(x - x_0)(x - x_1) + \cdots + c_n(x - x_0)\cdots(x - x_{n-1})$$

  p ist Linearkombination der Basis-Polynome $1,\ (x - x_0),$
  $(x - x_0)(x - x_1),\ \ldots,\ (x - x_0)(x - x_1)\cdots(x - x_{n-1})$

Newtons Interpolations-Algorithmus ist eine schlauere Variante der anfangs vorgestellten Holzhammer-Methode. Newton macht ebenfalls auf einem Ansatz mit unbestimmten Koeffizienten, aber mit anderen Basispolynomen:

$$p(x) = c_0 + c_1(x - x_0) + c_2(x - x_0)(x - x_1) + \cdots + c_n(x - x_0)\cdots(x - x_n)$$

Gesucht sind die Koeffizienten $c_0, \ldots, c_n$. Der Ansatz sieht auf den ersten Blick komplizierter aus als die Standardform, aber das resultierende Gleichungssystem wird einfacher: Die Gleichungen $p(x_i) = y_i$ ergeben nun ein Gleichungssystem mit unterer Dreiecksmatrix.

$$\begin{bmatrix} 1 & & & & 0 \\ 1 & (x_1 - x_0) & & & \\ 1 & (x_2 - x_0) & (x_2 - x_0)(x_2 - x_1) & & \\ \vdots & \vdots & & \ddots & \\ 1 & (x_n - x_0) & \cdots & & \prod_{i=0}^{n-1}(x_n - x_i) \end{bmatrix} \cdot \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ \\ c_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ \\ y_n \end{bmatrix} \tag{17}$$

Die Koeffizienten $c_0, \ldots, c_n$ lassen daraus der Reihe nach von oben nach unten berechnen:

$$c_0 = y_0, \quad c_1 = \frac{y_1 - y_0}{x_1 - x_0}, \ldots$$

Beim Weiterrechnen würden Sie bemerken: es treten ständig Differenzen von y-Werten, dividiert durch x-Wert-Differenzen auf. Das lässt sich als effizientes Rechenschema organisieren: das Verfahren der *dividierten Differenzen*. Darum geht es im nächsten Abschnitt.

7.2 Das Newtonsche Interpolationspolynom

Newtons Interpolations-Algorithmus, Kurzfassung:

- Ansatz mit *Newton-Basisfunktionen*
- Das *Schema der dividierten Differenzen* berechnet mit wenig Aufwand die Koeffizienten
- Auswertung des Polynoms effizient mit *Horner-Schema*.

Häufig sind Werte einer Funktion tabellarisch gegeben, wie im folgenden Beispiel:

Spezifische Wärmekapazität von kohlenstoffarmem Stahl in J/kg K für Temperaturen zwischen 0 und 600C

T	cp
0	460.8
100	471.1
200	496.4
300	537.0
400	593.3
500	666.8
600	760.8

Gesucht sind Interpolationspolynome, mit denen sich Zwischenwerte berechnen lassen, zum Beispiel ein kubisches Polynom für den Bereich $0 < T < 300$. (Dieses Beispiel wird dann später durchgerechnet. Es ist möglich, aber nicht unbedingt klug, ein einziges Interpolationspolynom für den gesamten Bereich $0 < T < 600$ aufzustellen. Aus praktischer Sicht ist vielleicht lineare Interpolation zwischen benachbarten Datenpunkten völlig ausreichend. Das hängt von der Qualität der Daten ab.)

Das Schema der dividierten Differenzen ist ein optimierter Rechenablauf zur Lösung des Gleichungssystems (17). Mit Papier und Stift organisiert man die Rechnung am besten in Tabellenform nach folgendem Schema

$$
\begin{array}{cccccccc}
 & & & x_0 & y_0 & & & \\
 & & x_1 - x_0 & & & [x_0, x_1] & & \\
 & x_2 - x_0 & & x_1 & y_1 & & [x_0, x_1, x_2] & \\
 & & x_2 - x_1 & & & [x_1, x_2] & & \\
\ldots & x_3 - x_1 & & x_2 & y_2 & & [x_1, x_2, x_3] & \ldots \\
 & & x_3 - x_2 & & & [x_2, x_3] & & \\
 & x_4 - x_2 & & x_3 & y_3 & & [x_2, x_3, x_4] & \\
 & & x_4 - x_3 & & & [x_3, x_4] & & \\
 & & & x_4 & y_4 & & &
\end{array}
$$

Darin bezeichnet das Symbol in eckigen Klammern, $[x_0, x_1]$ zwischen x_0 und x_1, eine dividierte Differenz, definiert durch

$$[x_0, x_1] = \frac{y_1 - y_0}{x_1 - x_0}$$

Das ist die Steigung der Geraden durch die beiden Datenpunkte.

Damit lässt sich auch schon die Formel für lineare Interpolation (Interpolationspolynom ersten Grades) angeben:

$$p(x) = y_0 + (x - x_0)[x_0, x_1]$$

Die höheren dividierten Differenzen erklärt man durch die niedrigeren. Eine zweite Differenz $[x_0, x_1, x_2]$ wird erklärt durch

$$[x_0, x_1, x_2] = \frac{[x_1, x_2] - [x_0, x_1]}{x_2 - x_0}$$

und allgemein ist für $k > i$

$$[x_i, x_{i+1}, ..., x_{k+1}] = \frac{[x_{i+1}, ..., x_{k+1}] - [x_i, ..., x_k]}{x_{k+1} - x_i}$$

Dann hat $p(x)$ folgende, durch Induktion leicht beweisbare Darstellung,

$$\begin{array}{rcl} p(x) = y_0 & + & (x - x_0)[x_0, x_1] + (x - x_0)(x - x_1)[x_0, x_1, x_2] + \\ & ... & +(x - x_0)(x - x_1)...(x - x_{n-1})[x_0, x_1, ..., x_n] \end{array}$$

Die Formeln sehen kompliziert aus, aber der eigentliche Rechengang ist sehr einfach: Man berechnet zuerst die Einträge in der linken Hälfte der Tabelle. In der rechten Tabellenhälfte erhält man die dividierten Differenzen nach der Regel „linker unterer minus linker oberer Nachbar, dividiert durch den symmetrisch gelegenen Eintrag aus der linken Tabellenhälfte“.

Die Koeffizienten des Newtonschen Interpolationspolynoms liegen in diesem Schema rechts in der oberen Schrägzeile (von der Mitte oben nach rechts schräg abwärts).

Das durchgerechnete Tableau für vier Datenpunkte aus dem obigen Beispiel sieht so aus:

```
          0    460,8
      100              0,103
  200     100  471,1           0,000750
300   100              0,253              0,000000050
  200     200  496,4           0,000765
      100              0,406
          300  537,0
```

Das Newton-Interpolationspolynom hat in diesem Fall die Form

$$p(x) = 460{,}8 + x \cdot 0{,}103 + x(x - 100) \cdot 0{,}000\,750 + x(x - 100)(x - 200) \cdot 0{,}000\,000\,050$$

Es ist nicht ratsam, diese Formel zu „vereinfachen“, indem man Produkte wie $x(x - 100)(x - 200)$ ausmultipliziert und gleiche Potenzen in x zusammenfasst.

Man setzt x-Werte in die obige Formel direkt ein oder schreibt sie noch rechengünstiger (Horner-Schema) in der Form

$$p(x) = 460{,}8 + x(0{,}103 + (x - 100)(0{,}000\,750 + (x - 200) \cdot 0{,}000\,000\,050))$$

Umformen auf $p(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n$ kann ganz drastische Rundungsfehler hervorrufen!

Bei äquidistanten x-Werten stehen in der linken Tabellenseite spaltenweise gleiche Werte, und der Rechengang könnte vereinfacht werden. Es gibt (z.B. im Bronstein) speziell optimierte Formeln für Interpolation mit äquidistanten x-Werten; hier beschränken wir uns auf den allgemeinen Fall.

Die x-Werte in dieser Rechentabelle brauchen auch nicht der Größe nach geordnet zu sein. Außerdem lässt sich die Tabelle leicht nachträglich ergänzen. Der Rest der Tabelle muss nicht neu berechnet werden, und auch im Interpolationspolynom kommt nur ein Term dazu, die anderen bleiben gleich. Das ist ein Vorteil des Newtonschen (und auch des Nevilleschen) Interpolationsverfahrens gegenüber dem Lagrangeschen Interpolationspolynom, bei dem sich alle Terme ändern, wenn ein Datenpunkt hinzugefügt oder geändert wird.

Will man zum Beispiel die gegebenen Daten im Bereich $200 \le x \le 600$ zuerst einmal nur linear approximieren, stellt man das Schema

```
      200   496,4
400                0,661
      600   760,8
```

auf und erhält

$$p(x) = 496{,}4 + (x - 200) \cdot 0{,}661 \ .$$

Will man doch quadratisch interpolieren, fügt man ein Wertepaar an, ergänzt die Tabelle

```
            200   496,4
       400                0,661
100         600   760,8          0,00085
     -300                0,746
            300   537,0
```

und hängt dem Interpolationspolynom einen Term an:

$$p(x) = 496{,}4 + (x - 200) \cdot 0{,}661 + (x - 200)(x - 600) \cdot 0{,}000\,85 \ .$$

Bei Polynomdarstellung in der Standard-Form ändern sich hingegen alle Koeffizienten, wenn Sie von linearer zu quadratischer Interpolation erweitern: Lineares und quadratisches Polynome lauten hier jeweils

$$\begin{aligned} p(x) &= 364{,}2 + 0{,}661 \cdot x \\ p(x) &= 466{,}2 - 0{,}019 \cdot x + 0{,}000\,85 \cdot x^2 \end{aligned}$$

Das Kapitel 7.3 illustriert an einem ähnlichen Beispiel das Verfahren von Neville, das sich gut eignet, wenn ein einziger Zwischenwert interpoliert werden soll und als Hilfsmittel nur Papier, Bleistift und ein billiger Taschenrechner zur Verfügung stehen.

Wichtige Zusatzinformationen in den Übungsunterlagen

Die Unterlagen zur 6. Übung erläutern Vandermonde-Ansatz und das Netwtonsche Interpolationsverfahren ausführlich. Auch der Ansatz mit Orthogonalpolynomen wird dort vorgestellt. Lesen Sie dort nach!

7.3 Das Verfahren von Neville

Das Verfahren von Neville berechnet den Wert des Interpolationspolynoms an einer gegebenen Stelle x. (Zum Verwechseln ähnlich ist das auch manchmal angegebene Aitken-Verfahren.)

Man setzt für $k = 0, 1, ..., n$

$$P_i(x) = y_i$$

Dann ist P_i der Wert des eindeutig bestimmten Polynoms vom Grad 0, das durch (x_i, y_i) geht.

Nun sei P_{01} der Wert (an der gegebenen Stelle x) des eindeutig bestimmten Polynoms ersten Grades (der Geraden), das durch die Punkte (x_0, y_0) und (x_1, y_1) geht. Und dementsprechend: $P_{12}, P_{23}, \ldots$. In gleicher Weise sei P_{012} der Wert der Parabel (an der gegebenen Stelle x) durch die Punkte (x_0, y_0), (x_1, y_1) und (x_2, y_2). Dieses Schema für Polynome höherer Ordnung weiterführend gelangt man zu $P_{0,1,2\ldots n}$, dem Wert des Polynoms durch alle gegebenen Punkte und damit der gesuchten Antwort.

Der Trick an der Sache ist, dass sich die einzelnen P-Werte leicht aus jeweils zwei vorherigen berechnen lassen. Am einfachsten schreibt sich die Formel mit der Determinante einer 2×2-Matrix,

$$P_{i(i+1)\ldots(i+m)} = \frac{1}{x_{i+m} - x_i} \begin{vmatrix} x - x_i & P_{i(i+1)\ldots(i+m-1)} \\ x - x_{i+m} & P_{(i+1)(i+2)\ldots(i+m)} \end{vmatrix}$$

Durch Induktion zeigt man leicht, dass die so definierten P genau die behaupteten Eigenschaften haben. Zur tatsächlichen Berechnung bedient man sich dann eines Schemas (Tableaus) der Form

$$\begin{array}{lllll} x_0: & y_0 = P_0 & & & \\ & & P_{01} & & \\ x_1: & y_1 = P_1 & & P_{012} & \\ & & P_{12} & & P_{0123} \\ x_2: & y_2 = P_2 & & P_{123} & \\ & & P_{23} & & \\ x_3: & y_3 = P_3 & & & \end{array}$$

Man füllt dieses Tableau spaltenweise von links nach rechts aus. Zum Beispiel ergibt sich der Wert P_{123} aus den beiden linken Nachbarn als

$$P_{123} = \frac{1}{x_3 - x_1} \begin{vmatrix} x - x_1 & P_{12} \\ x - x_3 & P_{23} \end{vmatrix} = \frac{(x - x_1)P_{23} - (x - x_3)P_{12}}{x_3 - x_1}$$

Will man die Interpolationsordnung erhöhen, kann man Werte für x_4 und y_4 unten an das Tableau anfügen und die fehlenden Werte $P_{34}, P_{234}, P_{1234}$ und P_{01234} ergänzen, ohne das restliche Tableau neu berechnen zu müssen (Vorteil gegenüber der Lagrangeschen Formel). Übrigens müssen die x_i weder äquidistant noch irgendwie geordnet sein.

Beispiel: Dichte von Wasser

Die Dichte von Wasser in Abhängigkeit von der Temperatur ist in folgender Tabelle für den Bereich von 0 bis 100 Celsius angegeben

T	ρ	T	ρ
0	999.840	10	999.699
1	999.899	20	998.203
2	999.940	30	995.645
3	999.964	40	992.212
4	999.972	50	988.030
5	999.964	60	983.191
6	999.940	70	977.759
7	999.901	80	971.785
8	999.848	90	965.304
9	999.781	100	958.345

Dichtewerte für Temperaturen, die nicht in der Tabelle angeführt sind, findet man durch Interpolation oder Approximation. Welches Verfahren ist angebracht?

- Einzelne Werte werden benötigt: Neville-Verfahren.
- Ein formelmäßiger Zusammenhang $T = T(\rho)$ in einem engen Temperaturintervall wird benötigt: Newtonsches Interpolationspolynom
- Ein formelmäßiger Zusammenhang über den gesamten Bereich ist gesucht: Approximation durch ein Polynom oder eine rationale Funktion.

Neville

Gesucht sei $\rho(24)$. Ein durchgerechnetes Tableau lautet

$$
\begin{array}{lllll}
10: & 999.699 & & & \\
 & & 997.605 & & \\
20: & 998.203 & & 997.307 & \\
 & & 997.180 & & 997.297 \\
30: & 995.645 & & 997.285 & \\
 & & 997.705 & & \\
40: & 992.212 & & &
\end{array}
$$

Die einzelnen Schritte dabei:

$$P_{01} = \frac{(x - x_0)P_1 - (x - x_1)P_0}{x_1 - x_0}, \quad P_{12} = \frac{(x - x_1)P_2 - (x - x_2)P_1}{x_2 - x_1}, \ldots$$

Das sind die Werte, die sich aus linearer Interpolation ergeben. In vielen Fällen wird lineare Interpolation ausreichend genau sein; man hätte sich dann mit dem Wert P_{12} zufriedengeben können. Die nächste Spalte entspricht quadratischer Interpolation gemäß den Formeln

$$P_{012} = \frac{(x - x_0)P_{12} - (x - x_2)P_{01}}{x_2 - x_0}, \quad P_{123} = \frac{(x - x_1)P_{23} - (x - x_3)P_{12}}{x_3 - x_1}.$$

Die daraus gewonnenen Werte unterscheiden sich nur mehr um 2 Einheiten in der Hundertstel-Stelle. Der letzte Wert entsteht aus kubischer Interpolation:

$$P_{0123} = \frac{(x - x_0)P_{123} - (x - x_3)P_{012}}{x_3 - x_0},$$

Rechenprogramme, welche Stoffgrößen benötigen, können entweder umfangreiche Tabellen abspeichern und darin linear interpolieren; alternativ können sie Speicherplatz sparen, Tabellen mit weniger Stützstellen und Interpolation höherer Ordnung verwenden. Sehr oft steckt ein beträchtlicher Teil der Rechenzeit in der Auswertung von Stoffgrößen; eine rechengünstige Implementierung ist dann bedeutsam.

7.4 Fehlerabschätzung der Polynominterpolation

Nehmen wir an das eine Funktion $f : [a, b] \to \mathbb{R}$, $-\infty < a < b < \infty$, an den Stützpunkten $\pi = \{a = x_0 < x_2 < \ldots < x_n = b\}$ gegeben ist und $p_n(x)$ ei Polynom $n-1$ten Grades ist, das eben durch diese Stützpunkten läuft. Die Frage ist jetzt, wie groß ist jetzt der Fehler, oder

$$e(x) = \sup_{x \in [a,b]} |p_n(x) - f(x)|.$$

Der folgende Satz liefert eine Abschätzung des Fehlers, abhängig von der Funktion f und deren Ableitungen.

Fehlerabschätzung für das Interpolationspolynom
Angenommen f ist $(n+1)$– mal stetig differenzierbar und das Polynom p_n geht durch die Punkte $\{(x_i, f(x_i)) : i = 0, \ldots, n\}$. Dann gibt es für jedes $x^* \in (a, b)$ ein Punkt $\xi \in (a, b)$ mit

$$f(x^*) - p_n(x^*) = \frac{f^{(n+1)}(\xi)}{(n+1)!}(x^* - x_0)(x^* - x_1) \cdots (x^* - x_{n-1})(x^* - x_n).$$

Beweis des Satzes: Betrachten wir die Funktion

$$\Theta(x) := f(x) - p_n(x) - c(x^*)(x - x_0)(x - x_1) \cdots (x - x_{n-1})(x - x_n)$$

wobei

$$c(x^*) = \frac{f(x^*) - p_n(x^*)}{(x^* - x_0)(x^* - x_1) \cdots (x^* - x_{n-1})(x^* - x_n)}.$$

Die Funktion Θ ist so definiert, dass x^* eine Nullstelle ist, d.h. $\Theta(x^*) = 0$. Da $f(x_k) = p(x_k)$ gilt, hat aber die Funktion Θ noch zusätzlich die Nullstellen $x_0, x_1, \ldots, x_n$, also $n+2$ Nullstellen. Differenziert man Θ (dies ist erlaubt da f nach Voraussetzung $n+1$ mal differenzierbar ist), so findet man nach den Satz von Rolle zwischen den Nullstellen von Θ jeweils Zwischenwerte $\xi_0, \xi_2, \ldots, \xi_n$ mit $\Theta'(\xi_j) = 0$, $j = 1, \ldots, n$. Nennen wir diese Zwischenwerte $\xi_j^{(1)}$. Wieder mit den Satz von Rolle wissen wir, dass es n Zwischenwerte $\xi_0, \xi_2, \ldots, \xi_{n-1}$ mit $\Theta''(\xi_j) = 0$ gibt, $j = 0, \ldots, n-1$. Nennen wir diese Zwischenwerte $\xi_j^{(2)}$. Dies können wir immer weiter machen bis wir nur einen Zwischenwert $\xi = \xi_1^{(n+1)}$ haben sodass gilt

$$0 = \Theta^{(n+1)}(\xi) = f^{(n+1)}(\xi) - c(x^*)(n+1)!.$$

Daraus folgt nach einer einfachen Umformung

$$c(x^*) = \frac{f^{(n+1)}(\xi)}{(n+1)!}.$$

also die Abschätzung aus den Satz.

Haben wir nun ein equidistantes Gitter und $0 \le a < b \le 1$, so gilt

$$|f(x^*) - p_n(x^*)| \le \sup_{\xi\in[a,b]} \frac{|f^{(n+1)}(\xi)|}{(n+1)!} \frac{(a-b)^n}{n^n} \le \sup_{\xi\in[a,b]} \frac{|f^{(n+1)}(\xi)|}{(n+1)!n^n}.$$

Hier haben wir folgende Abschätzung vorgenomen:

$$(x^* - x_0)(x^* - x_1)\cdots(x^* - x_{n-1})(x^* - x_n) \le \left(\frac{(a-b)}{n}\right)^n.$$

7.5 Interpolation oder Approximation?

Polynome hohen Grades neigen zu Oszillationen und sind bei der Auswertung in Standardform anfällig gegen Rundungsfehler. Ein Interpolationspolynom durch alle Datenpunkte im Beispiel zur Dichte von Wasser aus Abschnitt 7.3 liefert völlig sinnlose Werte.

> Das mit den Matlab-Befehlen `polyfit` und `polyval` berechnete und ausgewertete Interpolationspolynom weicht in diesem Fall selbst an den Original-Datenpunkten (wo es genau stimmen sollte!) deutlich ab: $\rho(0) = 1000.2$ statt 999.8. Zwischen den Datenpunkten liefert es Phantasiewerte im Bereich $\pm 10^8$.
> Numerisch genauere Berechnungen des Interpolationspolynoms würden zwar an den Original-Datenpunkten bessere Werte liefern, dazwischen aber trotzdem Schrott.

Eine formelmäßige Darstellung für den gesamten Datenbereich lässt sich mit Regressionspolynomen finden. Eine quadratische Approximation liefert Genauigkeit $\pm 0,5$,

$$\rho(T) = 1000.35 - 0.0614512\,T - 0.00364033\,T^2$$

ein Regressionspolynom vierten Grades

$$\rho(T) = 999.867 + 0.0545396\,T - 0.00765475\,T^2 + 0.0000434548\,T^3 - 1.38985\,10^{-7}\,T^4$$

weicht von den Datenpunkten maximal um 0.03 ab.

Rationale Funktionen sind oft zur Approximation besser geeignet. Beispielsweise zeigt die Näherung

$$\rho(T) = \frac{999.844 + 9.63049\,T - 0.0244379\,T^2}{1 + 0.00956721\,T - 0.0000163504\,T^2}$$

einen maximalen Fehler von 0.004. Auch solche Näherungen lassen sich nach der Methode der kleinsten Quadrate finden: Ansatz mit unbestimmten Koeffizienten, Einsetzen der Daten, Umformen auf ein überbestimmtes lineares System und Lösen der Normalengleichungen. (Je nach Ansatz können aber auch *nichtlineare* Ausgleichsprobleme entstehen, die viel schwieriger zu behandeln sind.) Die obige Näherung minimiert das Maximum des Fehlers (sogenannte *Minimax-Näherung*). Für die Konstruktion solcher Näherungen sei auf weiterführende Literatur verwiesen (Das Computer-Algebra-System Mathematica bietet z. B. dafür ein Paket an).

7.6 Weitere Interpolationsverfahren

Nicht alle Funktionen lassen sich in vorteilhafter Weise durch Polynome interpolieren. Andere wichtige Verfahren sind

- Rationale Interpolation
- Spline-Interpolation

- Trigonometrische Interpolation
- Interpolation in zwei oder mehr Dimensionen

Rationale Funktionen sind Quotienten zweier Polynome. Bulirsch und Stoer (siehe deren Buch Numerische Mathematik 1) haben einen Algorithmus gefunden, der mit einem Tableau ähnlich wie bei der Aitken-Neville-Interpolation arbeitet. Die meisten Funktionen lassen sich besser durch rationale Funktionen approximieren als durch Polynome. Auch Ihr Taschenrechner wertet mathematische Funktionen wie Cosinus und Sinus höchstwahrscheinlich in Form rationaler Ausrücke aus.

Spline-Funktionen spielen eine wichtige Rolle überall dort, wo Computer eine „glatte“ Kurve oder Oberfläche durch gegebene Punkte zeichnen sollen.

Trigonometrische Interpolation verwendet Summen von Sinus- und Cosinusfunktionen, oder komplexen Exponentialfunktionen, um eine Kurve durch einen Satz von Datenpunkten zu legen. Eng verwandt und überaus wichtig ist das Verfahren zur schnellen Fourier-Transformation (*Fast Fourier Transform, FFT*).

7.7 Spline-Interpolation

Interpolationspolynome hohen Grades sind rechnerisch aufwendig, oft sehr empfindlich gegenüber geringfügigen Änderungen der Daten (Koordinaten der Stützpunkte) und neigen zum Überschwingen. Will man – etwa für zeichnerische Zwecke – Punkte durch eine „möglichst glatte“ Kurve verbinden, bietet sich die Idee an, einzelne Polynome niedrigeren Grades aneinanderzustückeln. Derartige Funktionen werden allgemein als Splines bezeichnet.

Splines
Unter einer Spline-Funktion versteht man eine stückweise auf Teilintervallen definierte Funktion, deren Teile an den Intervallgrenzen stetig oder sogar ein- bzw. mehrmals stetig differenzierbar aneinanderstoßen.

Kubische (natürliche) Splines
Ein kubischer Spline $s(x)$ durch die $n+1$ Wertepaare (x_i, y_i), $i = 0, \ldots, n$ ist folgendermaßen charakterisiert:

In den einzelnen Intervallen (x_{i-1}, x_i) ist $s(x)$ jeweils ein kubisches Polynom

An den Intervallgrenzen stimmen die Funktionswerte, die ersten und die zweiten Ableitungen rechts- und linksseitig überein.

Zusatzbedingung: (natürlicher Spline) Am linken und rechten Rand ist $s''(x) = 0$.

Die natürlichen kubischen Splines sind unter allen zweimal stetig differenzierbaren, interpolierenden Funktionen $f(x)$ diejenigen mit dem kleinsten

$$\int_{x_0}^{x_n} \left(f''(x)\right)^2 dx$$

Dieses Integral misst aber im gewissen Sinn die Welligkeit oder Gesamtkrümmung der Funktion. Dadurch kann es natürlich kaum zum Überschwingen kommen.

Mit natürlichen kubischen Splines lassen sich aber keine schleifen- oder kreisförmigen Kurven zeichnen. Dazu dienen *parametrische* Splines, die sowohl x- als auch y-Werte einer Kurve als Splinefunktion eines Parameters darstellen. Die Methode zum Kurvenzeichnen in Excel arbeitet mit solchen Splines.

Nicht alle Arten von Splines interpolieren Punkte. *Bézier*-Splines verwenden einen Teil der Punkte, um die Gestalt der Kurve zu beeinflussen. Kurvenzeichnen in MS-Paint funktioniert in dieser Weise.

Fehlerabschätzung:

Nehmen wir an, dass eine Funktion $f : [a, b] \to \mathbb{R}$, $-\infty < a < b < \infty$, an den Stützpunkten $\Pi = \{a = x_0 < x_2 < \ldots < x_n = b\}$ gegeben ist, und $s_n(x)$ sei ein kubisches Spline mit genau diesen Stützpunkten $\{(x_i, f(x_i)) : i = 0, \ldots, n\}$. Die Frage ist jetzt, wie groß ist

$$e(x) = \sup_{x \in [a,b]} |s_n(x) - f(x)|.$$

Angenommen, f ist vier mal stetig differenzierbar, mit $f''(a) = f''(b) = 0$.

- dann existiert genau ein natürlicher kubischer Spline s_n.
- es gelten folgende Fehlerabschätzungen:

$$\sup_{x \in [a,b]} |f(x) - s_n(x)| \leq cM_4h^4, \tag{18}$$

$$\sup_{x \in [a,b]} |f'(x) - s_n'(x)| \leq cM_4h^3, \tag{19}$$

$$\sup_{x \in [a,b]} |f''(x) - s_n''(x)| \leq cM_4h^2, \tag{20}$$

$$\sup_{x \in [a,b]} |f'''(x) - s_n'''(x)| \leq cM_4h, \tag{21}$$

  wobei $h = \max(|x_j - x_{j-1}|)$ und $M_4 = \sup_{x \in [a,b]} |f^{IV}(x)|$.

8 Numerische Integration

Seit dem Altertum beschäftigt sich die Mathematik mit der Berechnung des Inhalts krummlinig berandeter Flächen. Solche Aufgaben erfordern – in heutiger Sprechweise – die Auswertung bestimmter Integrale.

Die *Quadratur des Kreises* ist mit Zirkel und Lineal, den Werkzeugen der antiken Geometer, undurchführbar. Der Begriff „Quadratur“ als Synonym für numerische Flächen- oder Integralberechnug ist geblieben. Numerischen Integration heißt also auch *numerische Quadratur*; die dazu geeigneten Formeln heißen *Quadraturformeln* (engl.: *quadrature formulas*).

Die analytisch integrierbaren Funktionen mit elementaren Stammfunktionen lassen sich auf wenigen Seiten einer Formelsammlung auflisten. Darüber hinaus können Integrale nur näherungsweise numerisch berechnet werden[12].

Auch wenn eine Funktion nicht als formelmäßiger Ausdruck, sondern als Tabelle oder Resultat eines Rechenprogrammes gegeben ist, kommt numerische Integration zum Einsatz.

8.1 Die Integrationsformeln von Newton-Cotes-Typ

Newton-Cotes-Formeln
Gegeben: Eine Funktion $f(x)$ in einem Intervall (a, b) durch ihre Werte f_i an $n + 1$ äquidistanten Stützstellen,

$$f_i = f(a + ih), \quad \text{mit } h = \frac{b-a}{n}, \quad i = 0, \ldots, n.$$

Gesucht: Ein Näherungswert für das Integral $\int_a^b f(x)dx$.
Prinzip: Interpoliere $f(x)$ durch ein Polynom $p(x)$. Nähere das Integral von f durch das Integral von p. Die Näherung ist gegeben als gewichtete Summe der f_i,

$$\int_a^b f(x)dx \approx (b-a) \sum_{i=0}^{n} \alpha_i f_i$$

mit fixen Gewichten α_i

Beispiele: Trapezregel

$$\int_a^b f(x)dx \approx \frac{b-a}{2} \left(f(a) + f(b)\right)$$

Quadratische Interpolation (Lokal: „Keplersche Fassregel“, international: „Simpson-Regel“)

$$\int_a^b f(x)dx \approx \frac{b-a}{6} \left(f(a) + 4f(\frac{a+b}{2}) + f(b)\right)$$

Diese Näherungsformel unter Verwendung von Funktionswerten an drei äquidistanten Stützstellen ist Johannes Kepler eingefallen, als er 1612 in Linz beim Kauf einiger Fässer Wein über deren Rauminhalt nachdachte (wieviel er dabei gekostet hat, ist nicht überliefert).

[12] Allerdings: wenn ein Integral oft genug auftritt, wird es kurzerhand als eigene Funktion *definiert*, tabelliert, und man findet spezielle Näherungsformeln und Reihenentwicklungen zur Auswertung. Beispiele: Das Integral $\int_0^\infty e^{-x^2} dx$ hat keine elementare Stammfunktion, ist aber so wichtig, dass dafür eine spezielle Funktion, die Gaußsche Fehlerfunktion, definiert ist. Auch für die Bogenlänge einer Ellipse oder die Schwingungsdauer eines mathematischen Pendels bei großen Amplituden sind Integrale auszuwerten, für die spezielle Funktionen (elliptische Funktionen) definiert sind.

Die Trapezregel liefert natürlich für lineare Funktionen exakte Werte. Die Keplersche Fassregel ist logischerweise für Parabeln, aber sogar noch für Polynome dritten Grades exakt. Die Fehler lassen sich durch Taylorreihenentwicklung abschätzen und hängen von den höheren Ableitungen der Funktion f ab.

In der Regel benutzt man diese Formeln nicht für das gesamte Intervall $[a, b]$, sondern unterteilt es in eine Reihe kleinerer Intervalle, wendet dort jeweils die Formel an und addiert die Teilergebnisse. Man spricht dann von *zusammengesetzten* Newton-Cotes-Formeln.

Gegeben: Von einer Funktion $f(x)$ in einem Intervall (a, b) die Werte f_i an $n+1$ äquidistanten Stützstellen,

$$f_i = f(a + ih), \quad \text{mit } h = \frac{b-a}{n}, \quad i = 0, \ldots, n.$$

Zusammengesetzte Trapezregel

$$\int_a^b f(x)dx = \frac{h}{2}(f_0 + 2f_1 + 2f_2 + \cdots + 2f_{n-1} + f_n) + E$$

Zusammengesetzte Simpson-Regel

$$\int_a^b f(x)dx = \frac{h}{3}(f_0 + 4f_1 + 2f_2 + 4f_3 + \cdots + 2f_{n-2} + 4f_{n-1} + f_n) + E$$

Nur für gerades n möglich!

Die Fehlerterme sind

$$E = \frac{a-b}{12}h^2 f''(\xi) \quad \text{für ein } \xi \in [a, b]$$

bei der zusammengesetzten Trapezregel und

$$E = \frac{a-b}{180}h^4 f''''(\xi) \quad \text{für ein } \xi \in [a, b]$$

bei der zusammengesetzten Simpson-Regel

8.2 Romberg-Verfahren, Gauß-Quadraturformeln

Das Romberg-Verfahren wendet die zusammengesetzte Trapezregel mehrfach, mit unterschiedlichen Schrittweiten an. Aus den Ergebnissen extrapoliert es einen noch genaueren Wert. Ausführliche Behandlung in den Übungsunterlagen, 6. Einheit.

Gauß-Quadraturformeln werden heuer (2023) nicht behandelt. Im Unterschied zu Formeln vom Newton-Cotes-Typ, die mit äquidistanten Stützstellen arbeiten, wählen Gauß-Formeln die Stützstellen an optimalen Positionen so, dass möglichst hohe Fehlerordnung erzielt wird.

9 Eigenwerte und Eigenvektoren

9.1 Einführung, Definition, Grundlagen

Eigenwertproblem
Gegeben ist eine $n \times n$-Matrix A. Gesucht sind

- ein vom Nullvektor verschiedener Vektor $\mathbf{x}$ und
- ein Skalar λ (auch $\lambda = 0$ ist erlaubt),

welche die Gleichung

$$A\mathbf{x} = \lambda\mathbf{x} \tag{22}$$

erfüllen.
Ein solches λ heißt *Eigenwert* von A, ein passendes $\mathbf{x}$ heißt *Eigenvektor* von A zum Eigenwert λ.

Die Situation „Matrix mal Eigenvektor ist Null mal Vektor“, also $A\mathbf{x} = 0\mathbf{x}$, kann durchaus auftreten. In so einem Fall ist $\lambda = 0$ ein Eigenwert von A.

Wir verlangen aber ausdrücklich, dass $\mathbf{x}$ nicht der Nullvektor 0 sein darf, weil die Gleichung $A \cdot 0 = \lambda \cdot 0$ für jedes beliebige λ erfüllt wäre – das wäre keine sinnvolle Definition von λ.

Anschauliche Interpretation

Der Hauptberuf einer Matrix ist, Vektoren zu multiplizieren. Wenn sie das macht, hat der Ergebnisvektor gewöhnlich eine andere Länge und zeigt in eine andere Richtung als der Ausgangsvektor. Jeder Matrix hat aber ganz spezielle „eigene“ Vektoren, bei denen sie zwar die Länge ändert, die Richtung aber gleich lässt (falls $\lambda > 0$) oder genau umkehrt (falls $\lambda < 0$). Es kann auch passieren (falls $\lambda = 0$), dass ein Eigenvektor von der Matrix zum Nullvektor gemacht wird. Reguläre Matrizen tun das aber nicht, nur singuläre Matrizen sind so fies.

Beispiel:
Die Matrix $A = \begin{bmatrix} -1 & 0 \\ 1 & 2 \end{bmatrix}$ macht aus $\mathbf{u} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$

$$A\mathbf{u} = \begin{bmatrix} -1 & 0 \\ 1 & 2 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} -1 \\ 5 \end{bmatrix},$$

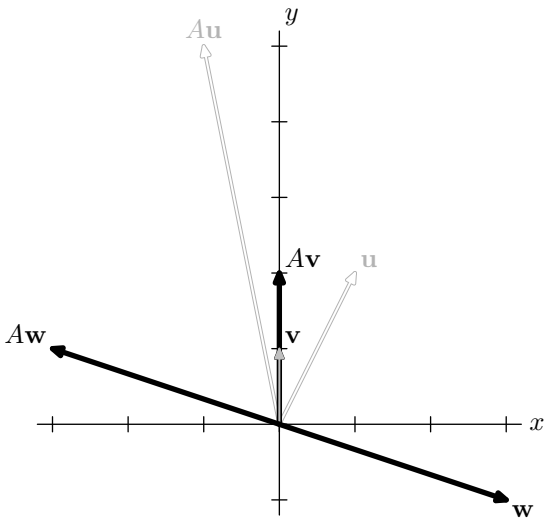
andere Länge und Richtung – kein Eigenvektor.
Der Vektor $\mathbf{v} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ wird hingegen zu

$$A\mathbf{v} = \begin{bmatrix} -1 & 0 \\ 1 & 2 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \end{bmatrix},$$

gleiche Richtung, doppelte Länge –
Eigenvektor zum Eigenwert 2.
Der Vektor $\mathbf{w} = \begin{bmatrix} 3 \\ -1 \end{bmatrix}$ wird zu

$$A\mathbf{w} = \begin{bmatrix} -1 & 0 \\ 1 & 2 \end{bmatrix} \cdot \begin{bmatrix} 3 \\ -1 \end{bmatrix} = \begin{bmatrix} -3 \\ 1 \end{bmatrix}$$

gleiche Länge, umgekehrte Richtung –
Eigenvektor zum Eigenwert -1.



Zu einem Eigenwert gibt es viele Eigenvektoren

Ist $\mathbf{x}$ ein Eigenvektor von A zum Eigenwert λ , so ist auch jedes Vielfache $\mu\mathbf{x}$, $\mu \neq 0$, ein Eigenvektor von A zum selben Eigenwert λ, denn da A, λ und $\mathbf{x}$ die Gleichung (22) erfüllen, gilt auch

$$A(\mu\mathbf{x}) = \mu(A\mathbf{x}) = \mu(\lambda\mathbf{x}) = \lambda(\mu\mathbf{x}).$$

Eigenvektoren sind also nur bis auf einen skalaren Faktor eindeutig bestimmt.

Eine $n \times n$-Matrix hat n Eigenwerte

Durch Umformung von (22) erhalten wir

$$\begin{aligned} A\mathbf{x} &= \lambda\mathbf{x} \\ A\mathbf{x} - \lambda\mathbf{x} &= 0 \\ (A - \lambda I)\mathbf{x} &= 0 \end{aligned} \tag{23}$$

wobei I die Einheitsmatrix ist. Der Vektor $\mathbf{x} = 0$ (der Nullvektor) ist immer eine Lösung von Gleichung (23), aber der interessiert uns nicht. Wir wollen, dass es noch andere, nicht triviale Lösungen gibt. Das ist nur genau dann der Fall, wenn

$$\det(A - \lambda I) = 0. \tag{24}$$

Entwickeln der Determinante liefert ein Polynom n-ten Grades in λ. Dieses Polynom heißt das *charakteristische Polynom* von A. Jedes Polynom n-ten Grades hat genau n reelle oder komplexe Nullstellen (sagt der Fundamentalsatz der Algebra; mehrfache Nullstellen zählt er dabei entsprechend ihrer Vielfachheit). Daraus folgt, dass jede $n \times n$-Matrix genau n (reelle oder komplexe, unter Umständen mehrfach gezählte) Eigenwerte hat.

Die Eigenwerte einer Matrix A lassen sich als Wurzeln (Nullstellen) ihres charakteristischen Polynoms berechnen. Dies ist ein klassische Verfahren, aber im allgemeinen nur für kleine Matrizen ($2 \times 2,\ 3 \times 3$) sinnvoll.

Rechenbeispiel: Wir berechnen die Eigenwerte der Matrix A vom vorigen Beispiel (Seite 76).

$$\begin{aligned} \det(A - \lambda I) &= \det\left(\begin{bmatrix} -1 & 0 \\ 1 & 2 \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right) = \det\begin{bmatrix} -1-\lambda & 0 \\ 1 & 2-\lambda \end{bmatrix}, \\ &= (-1-\lambda)\cdot(2-\lambda) - 0 \cdot 1 = \lambda^2 - \lambda - 2 \end{aligned}$$

Die Nullstellen des charakteristischen Polynoms $p(\lambda) = \lambda^2 - \lambda - 2$ und daher die Eigenwerte von A sind $\lambda_1 = -1\,,\ \lambda_2 = 2$.

Noch ein Beispiel: Wir berechnen die Eigenwerte der Matrix $A = \begin{bmatrix} 8 & 1 & 6 \\ 3 & 5 & 7 \\ 4 & 9 & 2 \end{bmatrix}$ als Wurzeln des charakteristischen Polynoms.

$$\det(A - \lambda I) = \det\left(\begin{bmatrix} 8 & 1 & 6 \\ 3 & 5 & 7 \\ 4 & 9 & 2 \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \right) = \det \begin{bmatrix} 8-\lambda & 1 & 6 \\ 3 & 5-\lambda & 7 \\ 4 & 9 & 2-\lambda \end{bmatrix}$$

Berechnen der Determinante, z.B. mit der Regel von Sarrus, liefert

$$\begin{aligned} \det(A - \lambda I) &= (8-\lambda)(5-\lambda)(2-\lambda) + 28 + 162 - 24(5-\lambda) - 63(8-\lambda) - 3(2-\lambda) \\ &= -360 + 24\,\lambda + 15\,\lambda^2 - \lambda^3 \end{aligned}$$

Das charakteristische Polynom von A lautet daher $p(\lambda) = -360 + 24\,\lambda + 15\,\lambda^2 - \lambda^3$, und seine Wurzeln (Nullstellen) können mit Verfahren des Kapitels 1 gefunden werden. Die drei Wurzeln, und damit die drei Eigenwerte von A sind $\lambda_1 = 15, \lambda_{2,3} = \pm 4,89898$.
Für größere Matrizen ist dieses Rechenverfahren zu umständlich und zu anfällig gegenüber Rundungsfehlern.

Kennt man einen Eigenwert λ, dann findet man einen dazu passenden Eigenvektor als Lösung des Gleichungssystems (23).

Für $\lambda = 15$ im obigen Beispiel suchen wir also eine Lösung des Systems

$$\begin{bmatrix} -7 & 1 & 6 \\ 3 & -10 & 7 \\ 4 & 9 & -13 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}.$$

In diesem System ergibt die Summe aus erster und zweiter Gleichung genau das Negative der dritten. Wenn Sie sich nun wundern, wie Sie aus nur zwei linear unabhängigen Gleichungen die drei Unbekannten $x_1,\ x_2,\ x_3$ bestimmen sollen, haben Sie die Pointe nicht verstanden: Das Gleichungssystem *darf nicht eindeutig lösbar sein!* Eben deswegen haben wir mit der Determinantenbedingung (24) dafür gesorgt, dass es außer der trivialen Lösung $x_1 = x_2 = x_3 = 0$ noch andere, sinvolle Lösungen gibt. Wir können zum Beispiel $x_1 = 1$ frei wählen und in die ersten beiden Gleichungen einsetzen, dann lassen sich x_2 und x_3 bestimmen:

$$\begin{array}{rcrcr} x_2 & + & 6x_3 & = & 7 \\ -10x_2 & + & 7x_3 & = & -3 \end{array} \qquad \text{Lösung: } x_2 = 1,\ x_3 = 1$$

9.2 Eigenwertaufgabe $Ax = \lambda x$: weitere Eigenschaften

Matrizen in physikalischen Anwendungen sind zumeist symmetrisch (Steifigkeitsmatrix, Spannungstensor, Trägheitstensor...). Die zugehörigen Eigenwerte entsprechen physikalischen Größen (Schwingungsfrequenzen, Hauptspannungen, Trägheitsmomenten...), von denen man erwartet, dass sie reell sind. Da ist es beruhigend zu wissen, dass die Mathematik garantiert:

Alle Eigenwerte einer symmetrischen Matrix sind reell.

Für eine symmetrische Matrix A vereinfacht sich die Eigenwertaufgabe beträchtlich. Dazu gibt es eine reichhaltige und elegante mathematische Theorie und eine Fülle von Verfahren.

Die Eigenvektoren einer symmetrischen Matrix bilden ein orthogonales System.

Sind die Eigenvektoren einer symmetrischen Matrix A zusätzlich auch auf Länge 1 normiert, dann lassen sie sich in einer orthogonalen Matrix Q zusammenfassen. Dann gilt:

$$Q^T A Q = D \quad \text{mit } D \text{ Diagonalmatrix aus Eigenwerten}\ .$$

Eigenwerte lassen sich „verschieben“

Ist λ ein Eigenwert von A und $\mathbf{x}$ ein zugehöriger Eigenvektor, dann ist für beliebiges $s \in \mathbb{R}$ der Wert $\lambda + s$ Eigenwert von $A + sI$ und dasselbe $\mathbf{x}$ ist zugehöriger Eigenvektor.

Eigenwerte der Inversen Matrix sind Inverse der Eigenwerte

Ist λ ein Eigenwert von A mit zugehorigem Eigenvektor $\mathbf{x}$, dann ist $1/\lambda$ ein Eigenwert von A^{-1} mit demselben Eigenvektor $\mathbf{x}$.

Ähnlichkeitstransformation

Ist X eine invertierbare Matrix[13] , dann haben A und $X^{-1}AX$ die gleichen Eigenwerte. Die Transformation von A zu $X^{-1}AX$ heißt *Ähnlichkeitstransformation* .

Moderne Rechenverfahren nützen Ähnlichkeitstransformationen, die Verschiebung von Eigenwerten und das Berechnen inverser Eigenwerte geschickt aus. Ohne solche Tricks wären die Verfahren deutlich langsamer.

Diagonalisieren

Hat eine $n \times n$-Matrix A genau n linear unabhängige Eigenvektoren, dann kann man sie als Spaltenvektoren zu einer $n \times n$-Matrix V zusammenfassen. Die Ähnlichkeitstransformation mit V erzeugt eine Diagonalmatrix D; in deren Haupdiagonale stehen die Eigenwerte.

$$V^{-1}AV = D$$

Beispiel: die Matrix $A = \begin{bmatrix} -1 & 0 \\ 1 & 2 \end{bmatrix}$ vom Beispiel auf Seite 76 hat Eigenvektoren $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$ und $\begin{bmatrix} 3 \\ -1 \end{bmatrix}$.
Hier sind also

$$V = \begin{bmatrix} 0 & 3 \\ 1 & -1 \end{bmatrix} \text{ und } V^{-1} = \begin{bmatrix} 1/3 & 1 \\ 1/3 & 0 \end{bmatrix}, \quad V^{-1}AV = \begin{bmatrix} 1/3 & 1 \\ 1/3 & 0 \end{bmatrix} \cdot \begin{bmatrix} -1 & 0 \\ 1 & 2 \end{bmatrix} \cdot \begin{bmatrix} 0 & 3 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & -1 \end{bmatrix}$$

Der MATLAB-Befehl zur Berechnung von Eigenwerten und -vektoren lautet `[V,D] = eig(A)`. Das im Hintergrund laufende Rechenverfahren führt iterativ Ähnlichkeitstransformationen durch, so dass sich die transformierte Matrix schrittweise einer Diagonalmatrix annähert. Als Endergebnis liefert es die Diagonalmatrix D und die Matrix V der Ähnlichkeitstransformation. Dann gilt: in der Hauptdiagonale von D stehen die Eigenwerten von A, die zugehörigen Eigenvektoren stehen in den Spalten von V.

Es kann sein, dass eine $n \times n$-Matrix A nicht n linear unabhängige Eigenvektoren hat. Das verkompliziert die Situation. Zum Glück treten – wie schon erwähnt – in wichtigen Anwendungen symmetrische Matrizen auf. Für sie gibt es immer eine Transformationsmatrix aus linear unabhängigen Eigenvektoren. Als zusätzlicher Bonus gilt: die Transformationsmatrix lässt sich als *orthogonale* Matrix Q wählen.

Kurzfassung:

> Symmetrische Matrizen lassen sich durch orthogonale Transformationen diagonalisieren:
>
> $$Q^T AQ = D \ .$$
>
> Die Spalten von Q sind Eigenvektoren von A mit den entsprechenden Diagonalelementen von D als Eigenwerten.

[13] gleichbedeutend sind: nichtsinguläre Matrix, reguläre Matrix

9.3 Vektoriteration

Die Berechnung von Eigenwerten und -vektoren ist in vielen Anwendungen von Bedeutung, beginnend mit der Mechanik (Eigenschwingungen) bis zur Ökonomie (Durchschnittspreise mit maximalem Gewinn) und zur Bewertung der Wichtigkeit von Internetseiten (PageRank einer Homepage als Eigenvektor der Google-Matrix, ein einfaches Beispiel findet sich in den Übungsaufgaben).

Dabei ist es nicht immer notwendig, alle Eigenwerte zu kennen, oft genügt es, den größten Eigenwert oder jenen mit größten Absolutbetrag zu berechnen.

Es kann in Ausnahmefällen mehrere Eigenwerte mit gleichem Absolutbetrag geben. Gibt es aber unter allen Eigenwerten *genau einen* mit maximalem Betrag, so nennen wir ihn *dominanten* Eigenwert. Ein solcher Eigenwert λ und ein zugehöriger Eigenvektor lassen sich durch ein einfaches iteratives Verfahren als Fixpunkt des Systems

$$\mathbf{x} = \frac{1}{\lambda} A\mathbf{x}$$

finden: die sogenannte Vektoriteration (englisch: *power iteration*).

Die Vektoriteration geht (ausgehend von früheren Ansätzen) auf Richard von Mises[14] und Hilda Geiringer[15] zurück. Kurzfassung: Man nehme irgendeinen Vektor $\mathbf{x}^{(0)}$ und multipliziere ihn mit A. Das Ergebnis skaliere man geeignet durch Division mit einem Faktor $\lambda^{(1)}$ und nenne es $\mathbf{x}^{(1)}$. Dies Verfahren setze man fort, bis sich die $\mathbf{x}^{(k)}$ nur mehr vernachlässigbar ändern. Dann ist $\lambda^{(k)}$ der Eigenwert mit größtem Absolutbetrag und $\mathbf{x}^{(k)}$ ein zugehöriger Eigenvektor.

Vektoriteration für den betragsgrößten Eigenwert
(Potenzmethode nach v. Mises und Geiringer, *power iteration*)
Gegeben eine $n \times n$-Matrix A, ein Startvektor $\mathbf{x}^{(0)} \neq 0$ und ein fix gewähltes $i \in \{1, \ldots, n\}$

Iteriere für $k = 1, 2, \ldots$
berechne $\mathbf{y}^{(k)} = A\mathbf{x}^{(k-1)}$
setze $\lambda^{(k)} = y_i^{(k)}$ (i-te Komponente von $\mathbf{y}^{(k)}$)
setze $\mathbf{x}^{(k)} = \mathbf{y}^{(k)}/\lambda^{(k)}$ (Skalierung)

Mit Papier und Bleistift ist die Multiplikation $A\mathbf{x}$ von tödlicher Kompliziertheit; Computer hingegen finden diese Methode recht einfach. Sie konvergiert, wenn es einen dominanten Eigenwert gibt und der Startvektor eine Komponente in Richtung des zugehörigen Eigenvektors hat.

Normaler Weise ist es irrelevant, welche Komponente von $\mathbf{x}^{(k)}$ zur Skalierung gewählt wird, außer der Eigenvektor hat genau für diese Komponente den Wert 0. Eine alternative Skalierunsvorschrift vermeidet die Festlegung auf eine bestimmte Komponente; das ist für Rechenprogramme einfacher:

Finde die betragsgrößte Komponente $y_i^{(k)}$ und setze $\lambda^{(k)} = y_i^{(k)}$.

[14] Richard von Mises, 1883 (Lemberg, Österreich; heute Lwiw, Ukraine) –1953 (Boston, USA). Mathematiker, studiert in Wien, ist Professor in Straßburg, Dresden, Berlin, Istanbul und ab 1939 an der Harvard University; richtungsweisende Arbeiten auf fast allen Gebieten der angewandten Mathematik, sowie der Strömungslehre, speziell unter Berücksichtigung des Flugzeugbaus (hält 1913 die erste Vorlesung über Motorflug!).

[15] Hilda Geiringer (Wien, 1893 – Santa Barbara, Kalifornien, 1973), studiert in Wien, arbeitet in Berlin als Assistentin von v. Mises am Inst. f. Angewandte Mathematik über Wahrscheinlichkeitstheorie und Statistik, emigriert 1933, lehrt in Brüssel, Istanbul und verschiedenen amerikanischen Universaitäten

Durch Anwenden der Potenzmethode auf die Matrix A^{-1} kann man den *betragskleinsten* Eigenwert ermitteln. Allgemein kann man jenen Eigenwert, welcher einem Wert μ am nächsten liegt, durch die Potenzmethode, angewandt auf $(A-\mu I)^{-1}$ finden (*inverse Iteration*). Dabei werden A^{-1} oder $(A-\mu I)^{-1}$ allerdings nicht explizit berechnet. Vielmehr wird der Iterationsschritt $\mathbf{y}^{(k)} = A^{-1}\mathbf{x}^{(k-1)}$ umformuliert zu $A\mathbf{y}^{(k)} = \mathbf{x}^{(k-1)}$ und $\mathbf{y}^{(k)}$ als Lösung dieses Systems bestimmt. Falls μ eine gute Näherung an ein λ_i ist, konvergiert das Verfahren rasch.

Für große, schwach besetzte symmetrische Matrizen gibt es ein besonders effizientes Verfahren, wenn nur der größte oder einige von den größten (oder kleinsten) Eigenwerten gesucht sind, das Lanczos-Verfahren. Im Kern stecken dort, wie bei der Potenzmethode, Iterationen der Form $\mathbf{y}^{(k)} = A\mathbf{x}^{(k-1)}$.

Auch dazu gibt es MATLAB-Befehle. Zum Beispiel liefert `d = eigs(A)` für schwach besetzte Matrizen die sechs betragsgrößten Eigenwerte und zugehörige Eigenvektoren.

9.4 Einfache Formen

Viele wichtige Verfahren zur Eigenwertberechnung bringen eine Matrix A durch eine Folge von Ähnlichkeitstransformationen $X^{-1}AX$ auf eine einfache Form und berechnen dann die Eigenwerte. (Die Eigenwerte ändern sich bei Ähnlichkeitstransformationen nicht.) Standard-Verfahren ist der sogenannte QR-Algorithmus.

Einfache Formen

- Diagonalmatrix
- Dreiecksmatrix
- Tridiagonalmatrix

Für Diagonal- und Dreiecksmatrizen sind die Eigenwerte genau die Diagonalelemente.

Für eine Tridiagonalmatrix lässt sich das charakteristische Polynom leicht auswerten, ohne dass man es explizit anschreiben müsste. Für eine symmetrische Tridiagonalmatrix T,

$$T = \begin{bmatrix} a_1 & b_1 & & \cdots & 0 \\ b_1 & a_2 & b_2 & & \vdots \\ & \ddots & \ddots & \ddots & \\ \vdots & & \ddots & \ddots & b_{n-1} \\ 0 & \cdots & & b_{n-1} & a_n \end{bmatrix},$$

zeigt Entwickeln der Determinante $\det(T - xI)$, dass der Wert p(x) des charakteristischen Polynoms rekursiv so bestimmt werden kann:

setze $p_0(x) = 1, p_{-1}(x) = 0$.
Für $k = 1, \ldots, n$
  $p_k(x) = (a_k - x)p_{k-1}(x) - b_{k-1}^2 p_{k-2}(x)$
Ergebnis $p(x) = p_n(x)$.

Ein Verfahren zur Nullstellenbestimmung, zum Beispiel Intervallhalbierung, liefert mit dieser Methode die Eigenwerte als Nullstellen des charakteristischen Polynoms.

10 Gewöhnliche Differentialgleichungen

Wichtiger Hinweis: Hier im Skriptum wird das Thema kurz und überblicksmäßig behandelt. Bitte sehen Sie sich auch die Vorlesungsfolien an, sie zeigen zusätzliches Material: Bilder, Erklärungen, Zusammenfassungen. Auch die Übungsunterlagen geben vertiefend Erläuterungen.

10.1 Aufgabenstellung, Beispiele

10.1.1 Differentialgleichungen erster Ordnung

Explizite gewöhnliche Differentialgleichung 1. Ordnung mit Anfangsbedingung
Gesucht ist eine Funktion $y(x)$, welche

$$\begin{aligned} y' &= f(x,y) \\ y(x_0) &= y_0 \end{aligned}$$

erfüllt. Wenn f in x stetig ist und einer Lipschitzbedingung genügt, dann existiert eine eindeutige Lösung in der Umgebung des Anfangspunktes x_0.

Die Schreibweise $y = y(x)$ ist Standard für die typische allgemeine Funktion und Variable. Je nach Anwendungsgebiet beschreiben Differentialgleichungen auch *zeitliche* Änderungen. Gesucht ist dann eine Funktion der Zeit, also $y = y(t)$ oder auch $x = x(t)$: Weg x als Funktion der Zeit t. Sie kennen aus der Physik die Schreibweise $\dot{x}(t)$ für Ableitungen nach der Zeit.

In der Schreibweise mit $y = y(t)$ lautet die Aufgabenstellung daher

> Gesucht ist eine Funktion $y(t)$, welche
>
> $$\begin{aligned} \dot{y} &= f(t,y) \\ y(t_0) &= y_0 \end{aligned}$$
>
> erfüllt.

Lassen Sie sich also nicht verwirren: je nach Kontext heißt die gesuchte Funktion y oder x oder (siehe zweites Beispiel) s, hängt von x oder t ab, und Ableitungen werden durch Punkte oder Striche bezeichnet.

Beispiel 1: Exponentielles Wachstum einer Population Eine Bakterienpopulation befinde sich in einer Nährflüssigkeit und habe zur Zeit $t > 0$ die Größe $y(t)$. Bei $t = 0$ hatte die Population die Größe $y(0) = y_0$. Ausgehend yom Zeitpunkt t wird sich nach Ablauf der Zeitspanne Δt die Population um $\Delta y = y(t + \Delta t) - y(t)$ Mitglieder vermehrt haben. Nun ist es sinnvoll, anzunehmen, dass bei kleinen Zeitspannen Δt die Vermehrung etwa proportional zu dem Bestand $y(t)$ und zu der Zeitspanne Δt ist. Also

$$\Delta y = y(t + \Delta t) - y(t) \sim a \cdot y(t) \cdot \Delta t.$$

mit einer Proportionalitätskonstanten $a > 0$. Berechnen wir nun die Ableitung mit Hilfe des Differentialquotienten, erhalten wir

$$\dot{y}(t) = \lim_{\Delta t \to 0} \frac{1}{\Delta t} \left(y(t + \Delta t) - y(t)\right) = \lim_{\Delta t \to 0} \frac{1}{\Delta t} a \cdot y(t) \cdot \Delta t = a\, y(t).$$

Neben der Differentialgleichung muss die Funktion $y(t)$ noch die *Anfangsbedingung* $y(0) = y_0$ erfüllen. Dieses *Anfangswertproblem*

$$\begin{aligned} \dot{y}(t) &= ay(t) \\ y(0) &= y_0 \end{aligned}$$

beschreibt das Wachstumsgesetz der Bakterienpopulation. Sie kennen sicher die Lösung: Es handelt sich um *exponentielles Wachstum*. Die Lösung des Anfangswertproblems lautet

$$y(t) = y_0 \exp(at) \ .$$

Dieselbe Differentialgleichung ist auch das Modell eines Sparbuchs, Kredits, oder des radioaktiven Zerfalls (mit $a < 0$).

Beschränktes Wachstum einer Population:

Sind die Ressourcen nicht unbegrenzt, so spricht man vom beschränkten Wachstum. Beim beschränkten Wachstum ist die Änderungsrate proportional zum so genannten Sättigungsmanko (Differenz zwischen oberen (bzw. unteren) Schranke S und dem alten Bestand $y(t)$). Wir erhalten damit folgende Differentialgleichung

$$\dot{y}(t) = k(S - y(t)), \quad t > 0.$$

Die zugehörige Lösung lautet

$$y(t) = S - ce^{-kt}, \quad t \geq 0, \, c = S - y(0).$$

10.1.2 Höherere Ordnung und Systeme von Differentialgleichungen

Beispiel 2: Bewegung im Schwerefeld der Erde: Ein Körper der Masse m befindet sich im Schwerefeld der Erde (Masse M). Ist s der Abstand Körper—Erdmittelpunkt, dann gilt für die Anziehungskraft nach dem Gravitationsgesetz:

$$K = \gamma \frac{M\, m}{s^2}, \quad \gamma = \text{Gravitationskonstante}.$$

Zur Zeit $t = 0$ ist der Abstand des Körpers s_0. Er bewegt sich mit der Anfangsgeschwindigkeit v_0, und zwar nach oben, wenn v_0 positiv ist, und nach unten, wenn v_0 negativ ist. Nach t Zeiteinheiten hat er den Abstand $s(t)$ vom Erdmittelpunkt. Luftreibung werde vernachlässigt. Nach dem Newtonschen Kraftgesetz folgt daher die Differentialgleichung *zweiter Ordnung*

$$m\ddot{s}(t) = -\gamma \frac{M\, m}{s(t)^2}. \tag{25}$$

Diese Differentialgleichung hat viele Lösungen. Um eine festzulegen, fordert man zusätzlich folgende Anfangsbedingungen (Anfangsposition, Anfangsgeschwindigkeit):

$$s(0) = s_0, \quad \dot{s}(0) = v_0.$$

Hier lässt sich eine explizite Lösung in der Form $s(t) = \ldots$ nicht mehr angeben. *Numerische Verfahren* können aber leicht Lösungen in jeder gewünschten Genauigkeit liefern.

Nur die allereinfachsten Differentialgleichungen lassen sich analytisch lösen. In der Praxis ist man zumeist auf numerische Lösungsverfahren angewiesen.

In der Differentialgleichung 25 treten *zweite* Ableitungen auf; es handelt sich um eine *Differentialgleichung zweiter Ordnung*. Führen wir zusätzlich zum Weg $s(t)$ noch die Geschwindigkeit $v(t)$ als zweite gesuchte Funktion ein, dann lässt sich diese Gleichung äquivalent als ein *System zweier Differentialgleichungen erster Ordnung* schreiben:

$$\begin{aligned} \dot{s}(t) &= v(t) \\ \dot{v}(t) &= -\gamma \frac{M}{s(t)^2}. \end{aligned}$$

Begründung: Die erste Gleichung definiert die Geschwindigkeit v als zeitliche Ableitung des Weges s; die zweite Gleichung formuliert, weil $\dot{v}(t) = \ddot{s}(t)$, das Kraftgesetz 25.

Die allgemeine Aufgabenstellung lautet:

System von n gewöhnlichen Differentialgleichungen 1. Ordnung, Anfangswertproblem
Gesucht sind n Funktionen $y_1(x), \ldots, y_n(x)$, welche das System

$$\left.\begin{array}{rcl} y_1' &=& f_1(x, y_1, \ldots, y_n) \\ y_2' &=& f_2(x, y_1, \ldots, y_n) \\ \vdots & & \vdots \\ y_n' &=& f_n(x, y_1, \ldots, y_n) \end{array}\right\} \text{Differentialgleichungen}$$

$$\left.\begin{array}{rcl} y_1(x_0) &=& y_{10} \\ y_2(x_0) &=& y_{20} \\ \vdots & & \vdots \\ y_n(x_0) &=& y_{n0} \end{array}\right\} \text{Anfangsbedingungen}$$

erfüllen.

Für $n = 2$ ist eine Schreibweise mit zwei Funktionen $y(x)$ und $z(x)$ (oder, so wie im vorigen Beispiel, $s(t)$ und $v(t)$) einfacher als die Index-Schreibweise $y_1(x)$ und $y_2(x)$: Differentialgleichungen

$$\begin{aligned} y' &= f(x, y) \\ z' &= g(x, y) \end{aligned}$$

Bei mehr Gleichungen gehen aber rasch die Buchstaben aus, und die Formeln für Rechenverfahren werden unübersichtlich. Eine systematische Schreibweise, die auch Computerprogramme sehr vereinfacht, faßt Funktionen vektoriell zusammen:

$$\mathbf{y}(x) = \begin{bmatrix} y_1(x) \\ y_2(x) \\ \vdots \\ y_n(x) \end{bmatrix}, \quad \mathbf{y}\,'(x) = \begin{bmatrix} y_1'(x) \\ y_2'(x) \\ \vdots \\ y_n'(x) \end{bmatrix}, \quad \mathbf{f}(x, \mathbf{y}) = \begin{bmatrix} f_1(x, y_1, \ldots, y_n) \\ f_2(x, y_1, \ldots, y_n) \\ \vdots \\ f_n(x, y_1, \ldots, y_n) \end{bmatrix}$$

Dann lautet das Anfangswertproblem schlicht

$$\begin{aligned} \mathbf{y}\,' &= \mathbf{f}(x, \mathbf{y}) \\ \mathbf{y}(x_0) &= \mathbf{y}_0 \end{aligned}$$

Gewöhnliche Differentialgleichung höherer Ordnung
Eine DG höherer Ordnung lässt sich durch Einführen von Hilfsfunktionen in ein äquivalentes System von DGs 1. Ordnung transformieren.
Diese Umformung ist bei praktischen Aufgaben häufig notwendig (und auch eine beliebte Prüfungsfrage). Die Übungsunterlagen bringen eine ausführliche Anleitung und Beispiele.

10.2 Numerische Verfahren

Explizite Einschrittverfahren

Wichtige Verfahren sind das Eulersche Polygonzugverfahren (weil es das einfachste ist: Verfahren 1. Ordnung), das Verfahren von Heun und das modifizierte Euler-Verfahren (weil sie genauer sind: Verfahren 2. Ordnung) und das klassische Runge-Kutta-Verfahren (weil man damit in der Praxis oft rechnet; Verfahren 4. Ordnung).

Zur numerischen Lösung einer DG bestimmt ein explizites Einschrittverfahren ausgehend von den Anfangsbedingungen eine Folge von Wertepaaren $(x_0, y_0), (x_1, y_1), (x_2, y_2), \ldots$, die den Verlauf der gesuchten Funktion $y = y(x)$ annähern sollen. Schema:

Wähle Schrittweite h und maximale Schrittzahl N;
setze x_0 und y_0 laut Anfangsbedingung;
für $i = 0, 1, \ldots, N$
$x_{i+1} = x_i + h$;
$y_{i+1} = y_i + hF(x_i, y_i, h)$.

Die Funktion $F(x, y, h)$ heißt die *Verfahrensfunktion* des jeweiligen Verfahrens. Geometrisch interpretiert gibt $F(x, y, h)$ die *Fortschreitungsrichtung*. Beim klassischen Euler-Verfahren (Eulersche Polygonzugverfahren) ist sie gleich der Tangentensteigung im Ausgangspunkt,

$$F(x, y, h) = f(x, y),$$

beim modifizierten Euler-Verfahren

$$F(x, y, h) = f\left(x + \frac{h}{2}, y + \frac{h}{2} f(x, y)\right),$$

beim Verfahren von Heun

$$F(x, y, h) = \frac{1}{2}(k_1 + k_2)$$

mit

$$\begin{aligned} k_1 &= f(x, y) \\ k_2 &= f(x + h, y + hf(x, y)), \end{aligned}$$

beim klassischen Verfahren von Runge-Kutta

$$F(x, y, h) = \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

mit

$$\begin{aligned} k_1 &= f(x,y) \\ k_2 &= f\left(x+\frac{h}{2}, y+\frac{h}{2}k_1\right) \\ k_3 &= f\left(x+\frac{h}{2}, y+\frac{h}{2}k_2\right) \\ k_4 &= f(x+h, y+hk_3). \end{aligned}$$

10.3 Illustrationen zu einem einfachen Anfangswertproblem

Geometrische Interpretation
Eine Differentialgleichung gibt ein Richtungsfeld vor

Gegeben sei die Differentialgleichung

$$y' = xy/4 - 1\ .$$

Sie ordnet jedem Punkt (x, y) in der Ebene eine Steigung y' zu. Fassen wir diese Steigungen als Richtungsvektoren (mit einheitlicher Länge) auf, dann erhalten wir ein *Richtungsfeld*, wie in Abbildung 9 dargestellt.

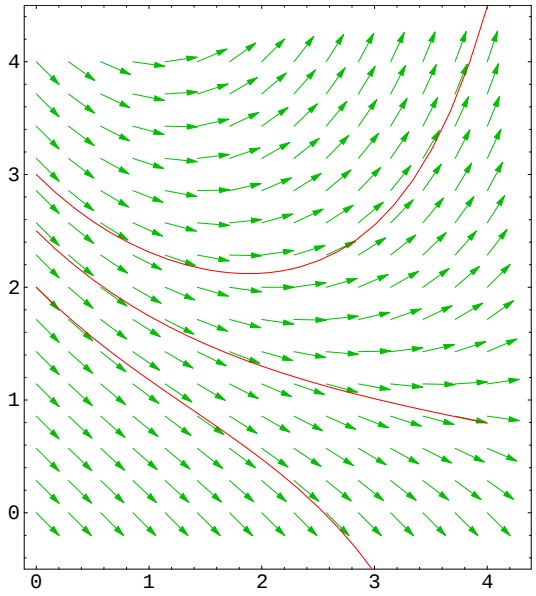


Abbildung 9: Die Differentialgleichung gibt ein Richtungsfeld vor. Drei Lösungen zu verschiedenen Anfangsbedingungen sind eingetragen

Eine Lösungskurve startet an einem Punkt (x_0, y_0), der durch die Anfangsbedingung festgelegt ist, in der dort vorgegebenen Richtung. Im weiteren Verlauf folgt die Kurve immer den Richtungspfeilen. Sie orientiert sich also in jedem Punkt (x, y), den sie erreicht, völlig opportunistisch an der dort herrschenden Richtung.

Ein Einschrittverfahren, wie beispielsweise das Eulersche Polygonzugverfahren, startet ebenfalls am Punkt (x_0, y_0). Das Eulersche Polygonzugverfahren schlägt auch genau die in (x_0, y_0) gegebene Richtung ein und steht eine Schrittweite h lang stur zu dieser Entscheidung. Es gelangt damit an einen Punkt (x_1, y_1). Dort erst überdenkt es seinen Weiterweg und orientiert sich neu, wieder genau nach der in (x_1, y_1) herrschenden Richtung. Je kleiner die Schrittweite, also um so öfter sich das Verfahren den herrschenden Gegebenheiten anpasst, um so besser sollte es der exakten Lösungskurve folgen. Die Abbildung 10 illustriert diese Gedanken. Eine Präzisierung der Idee und eine Untersuchung der Fehler bringt das nächsten Kapitel.

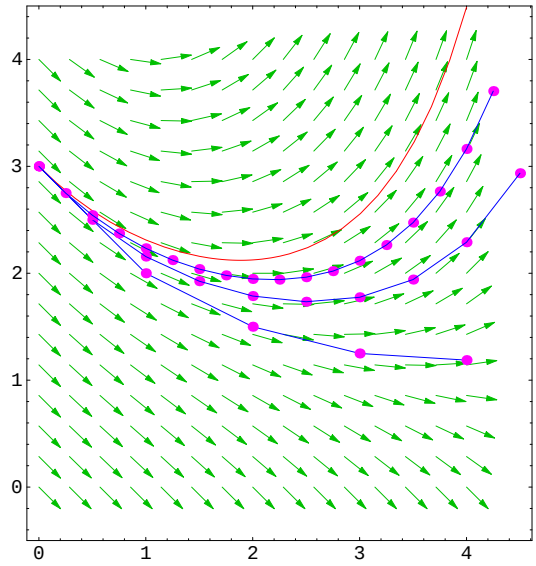


Abbildung 10: Für die Differentialgleichung $y' = xy/4 - 1$ mit Anfangsbedingung $y(0) = 3$ sind die exakte Lösung sowie drei Näherungen nach dem Eulerschen Polygonzugverfahren mit Schrittweiten $h = 1; \frac{1}{2}; \frac{1}{4}$ eingetragen

Das Eulersche Polygonzugverfahren ist allerdings kurzsichtig, in dem Sinn, dass es als Fortschreitungsrichtung (Verfahrensfunktion) einfach die am Ausgangspunkt gegebene Richtung wählt. Das modifizierte Eulerverfahren versucht, vorausschauender zu agieren: es tastet sich zuerst eine halbe Schrittweite voran, erkundet die dort gegebene Richtung und wählt diese als Fortschreiterichtung (Verfahrensfunktion). Siehe dazu die Abbildungen 11 und 12.

Ähnlich geht das Verfahren von Heun vor. Es macht versuchsweise einen Schritt in die selbe Richtung wie das gewöhnliche Polygonzugverfahren, erkundet am Zielpunkt die Richtung und wählt als endgültige Fortschreitungsrichtung den Mittelwert der Richtungen an Start- und (vorläufigem) Zielpunkt. Siehe dazu Abbildung 13

Es gibt in dieser Art noch weitere Möglichkeiten, das einfache Eulersche Polygonzugverfahren zu verbessern. In der Literatur werden dafür nicht einheitliche Namen verwendet. Auch die folgende Methode geht auf Heun zurück und wird gelegentlich als Heunsches Verfahren bezeichnet:

$$F(x, y, h) = \frac{1}{4}(k_1 + 3k_2)$$

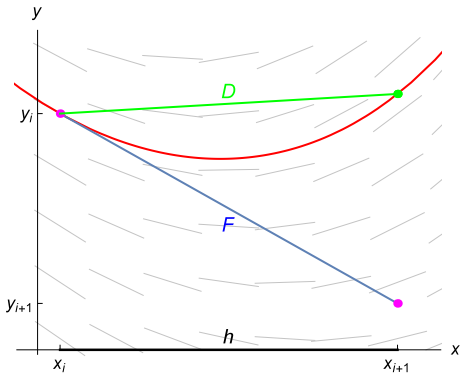


Abbildung 11: Verfahrensfunktion F des expliziten Euler-Verfahrens und exakte Richtung D.

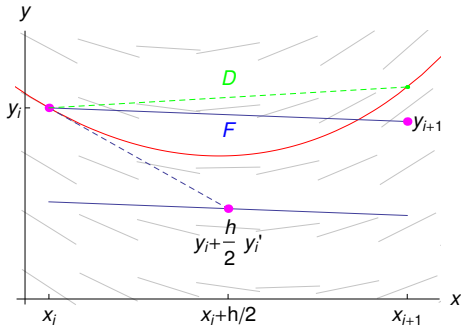


Abbildung 12: Wahl der Verfahrensfunktion F (geometrisch: Fortschreit-Richtung) beim modifizierten Eulerverfahren. F nähert den exakten Differenzenquotient D besser an als beim expliziten Euler-Verfahren, vergleiche Abbildung 11.

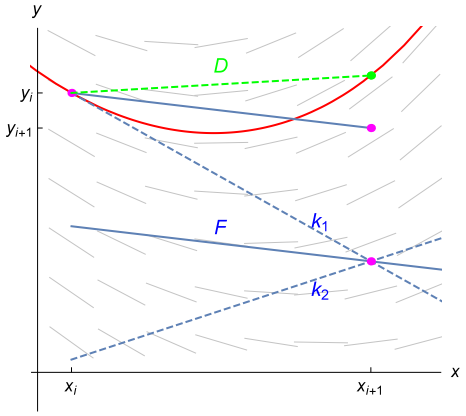


Abbildung 13: Wahl der Verfahrensfunktion F (geometrisch: Fortschreit-Richtung) beim Heun-Verfahren: Mittelwert aus Anfangsrichtung und Richtung am (genäherten) Endpunkt. F nähert den exakten Differenzenquotient ähnlich genau wie das modifizierte Euler-Verfahren D an.

mit

$$\begin{aligned} k_1 &= f(x,y) \\ k_2 &= f(x+\frac{2}{3}h, y+\frac{2}{3}hf(x,y)) \end{aligned}$$

Das klassische Runge-Kutta-Verfahren treibt das Wechselspiel aus versuchsweisem Vortasten und Erkunden der Richtung zur Perfektion: es berechnet vier verschiedene Steigungen $k_1, \ldots, k_4$, eine am Ausgangspunkt, zwei in der Mitte und eine am Ziel, und wählt als Verfahrensfunktion ein gewogenes Mittel der so berechneten Richtungen.

Allgemein bezeichnet man Methoden dieser Art als Runge-Kutta-Verfahren (deswegen oben der Zusatz „klassisch“). Moderne Runge-Kutta-Verfahren versuchen, mit möglichst geringem Rechenaufwand eine hohe Ordnung und gleichzeitig Abschätzungen über die Größe des Fehlers oder die optimale Schrittweite zu gewinnen.

10.4 Diskretisierungsfehler

Wenn ein numerisches Verfahren näherungsweise Werte y_i der Lösung y eines Anfangswertproblems berechnen soll, sind Fragen nach der Größe des Fehlers und nach der Anzahl der korrekten Stellen im Resultat wichtig.

Zwei Arten von Fehlern sind beim Einschrittverfahren zu unterscheiden:

- In jedem einzelnen Schritt wählt die Verfahrensfunktion eine Fortschreitrichtung, die normalerweise nicht exakt den Wert der Lösung trifft. Dieser Richtungsunterschied ist der lokaler Diskretisierungsfehler.
- Die Effekte der lokalen Diskretisierungsfehler in den einzelnen Schritten akkumulieren sich. Daraus resultierend weicht die numerische Lösung von der exakten Lösung ab. Letztlich ist natürlich dieser Fehler, der *globale Diskretisierungsfehler* interessant. Er lässt sich mit dem lokalen Diskretisierungsfehler abschätzen.

Die Abbildung 11, 12 und 13 zeigen unterschiedlich große lokale Diskretisierungsfehler. Eingezeichnet ist jeweils die von der Verfahrensfunktion F gewählte Richtung, sowie (für die von y_i ausgehenden exakte Lösung) die vom Differenzenquotienten D bestimmte exakte Richtung.

Sowohl F als auch D hängen von x_i, y_i und der Schrittweite h ab und nähern sich für $h \to 0$ der Tangentensteigung $f(x_i, y_i)$ an. Es gilt

$$\lim_{h \to 0} D(x_m, y_m, h) = f(x_m, y_m).$$

Die Abweichung d der von der Verfahrensfunktion F bestimmten Richtung von der exakten Richtung D

$$d(x_m, y_m, h) = |F(x_m, y_m, h) - D(x_m, y_m, h)|$$

heißt *lokaler Diskretisierungsfehler*. Natürlich soll dieser Fehler um so kleiner werden, je kleiner die Schrittweite h gewählt wird. Information darüber, wie stark d von der Schrittweite abhängt, gibt die *Ordnung des Verfahrens*.

Ordnung eines Einschrittverfahrens
Die größte natürliche Zahl p mit

$$d(x_m, y_m, h) = O(h^p)$$

heißt Ordnung des Verfahrens.

Ordnung 1: Fehler d proportional Schrittweite
Ordnung 2: Fehler d proportional dem Quadrat der Schrittweite
usw.

Die Ordnung des lokalen Diskretisierungsfehlers ist wichtig, weil sie sich durch mathematische Methoden (wie etwa Taylorreihenentwicklung) abschätzen lässt. Die Effekte der lokalen Diskretisierungsfehler in den einzelnen Schritten akkumulieren sich. Daraus resultierend weicht die numerische Lösung von der exakten Lösung ab. Letztlich ist natürlich dieser Fehler, der *globale Diskretisierungsfehler* interessant. Er lässt sich mittels dem lokalen Diskretisierungsfehler abschätzen.

Lokaler und globaler Diskretisierungsfehler
Der Fehler d zwischen Verfahrensfunktion F und exakter Richtung D

$$d(x_m, y_m, h) = |F(x_m, y_m, h) - D(x_m, y_m, h)|$$

heißt *lokaler Diskretisierungsfehler* im Punkt (x_m, y_m).
Ist Y die exakte Lösung der Anfangswertaufgabe

$$y' = f(x, y), \quad y(x_0) = y_0 \ ,$$

und y_m die Näherungslösung an der Stelle x_m, so nennt man den Fehler

$$g(x_m, h) = |y_m - Y(x_m)|$$

den *globalen Diskretisierungsfehler* .

Von jedem Einschrittverfahren verlangen wir dann, daß an einer gegebenen Stelle x der Fehler $g(x, h)$ mit h gegen Null geht und nennen die größte natürliche Zahl p mit

$$g(x, h) = |y(x, h) - Y(x)| = O(h^p)$$

die Konvergenzordnung des Verfahrens. Es gilt folgender Zusammenhang zwischen dem lokalen und dem globalen Diskretisierungsfehler:

Konvergenz des Einschrittverfahrens
Ist der lokale Diskretisierungsfehler von der Ordnung $p \geq 1$ und genügt F einer Lipschitzbedingung, so konvergiert das Einschrittverfahren mit Ordnung p.

Ordnung einzelner Verfahren
Ordnung 1: Eulersches Polygonzugverfahren
Ordnung 2: Modifiziertes Euler-Verfahren, Verfahren von Heun
Ordnung 4: Klassisches Runge-Kutta-Verfahren

10.5 Wann soll man welches Verfahren benützen

Suchen Sie z.B. `ode45` in der MATLAB-Hilfe, so listet MATLAB Ihnen folgende Gleichungslöser (solver) für Differentialgleichungen auf: `ode23`, `ode45`, `ode113`, `ode15s`, `ode23s`, `ode23t`, `ode23tb`. Welches Verfahren soll man benützen? Das erste Auswahlkriterium ist die Konvergenzordnung (je höher, desto genauer). Ist aber die zu erwartende Lösung allerdings nicht glatt (das heißt zum Beispiel, f hat Sprungstellen, Ecken, oder höhere Ableitungen existieren nicht), arbeitet ein Verfahren mit niedriger Konvergenzordnung schneller und braucht weniger Rechenleistung als ein Verfahren mit hoher Konvergenzordnung.

Auswahlkriterien des Solvers

- Konvergenzordnung - Genauigkeit der Approximation: MATLABs `ode45` ist ein Verfahren der Ordnung 5, allerdings nur, wenn die Funktion, die Sie an `ode45` übergeben, auch genügend oft differenzierbar ist. Wenn nicht, dann arbeitet `ode23` effizienter.
- Steifheit des Systems: Ist das System steif, sollte man `ode15s`, `ode23t` oder `ode23tb` verwenden.
- Komplizierte rechte Seite: explizite Solver (`ode45`, `ode23`, `ode23`, `ode113`) rechnen oft schneller, weil sie – im Gegensatz zu impliziten Verfahren – nicht in jedem Schritt Gleichungssysteme lösen müssen. Die MATLAB-Hilfe empfiehlt dazu: *"ode113 can be more efficient than ode45 [...] when the ODE function is expensive to evaluate"*.

Die folgenden Abschnitte gehen auf Probleme bei der Anwendung numerischer Lösungsverfahren ein. Wichtige Themen sind:

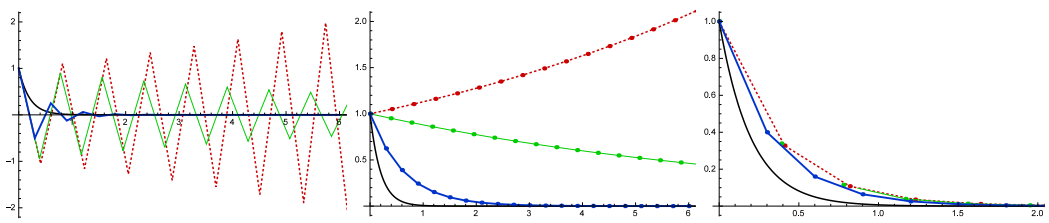
- explizite im Vergleich zu impliziten Verfahren
- Stabilität
- Steifheit

10.5.1 Stabilität

Zu diesen Punkt möchte ich als erstes den Unterschied zwischen expliziten Verfahren und impliziten Verfahren erklären. Gegeben ist folgende Differentialgleichung

$$\begin{cases} y'(x) &= -5\,y(x), \\ y(0) &= 1. \end{cases}$$

In den nachfolgenden Bildern wurde die exakte Lösung e^{-5x} (schwarz gezeichnete Kurve) mit verschiedenen Verfahren approximiert. Dabei wurden jeweils als Schrittweiten 0,3 (blau gezeichnet), 0,39 (grün), und 0,41 (rot) gewählt. Von links nach rechts: explizites Euler-Verfahren, Verfahren von Heun, implizits Euler-Verfahren.



Schaut man sich die Graphen an, sieht man, dass das explizite Euler-Verfahren für Schrittweite 0,41 osziliert und divergiert. Auch das Heun-Verfahren divergiert bei dieser Schrittweite. Man sagt, die Verfahren sind *instabil* bei dieser Schrittweite.

Bei Schrittweiten $< 0{,}4$ geben die Verfahren zumindest grob qualitativ das Verhalten der exakten Lösung wieder: sie klingen exponentiell ab. Man bezeichnet dieses Verhalten als *stabil*.

> Ein Verfahren heißt stabil, wenn es exponentiell abklingende Lösungen qualitativ richtig berechnet. Stabilität kann von der Schrittweite abhängen.

- Es gibt verschiedene Definitionen für Stabilität eines numerischen Verfahrens.
- Allgemein kann man sich merken: Implizite Verfahren – immer stabil, expliziten Verfahren – nur bei kleinen Schrittweiten stabil.
- Stabilitätsprobleme treten vor allem bei *steifen Differentialgleichungen* auf. Für solche Probleme sind implizite Verfahren besser geeignet.

Wenn man die einzelnen Schritte des expliziten Euler-Verfahrens nachrechnet, erkennt man auch, was vorgeht: die Rechenschritte sind nämlich

$$\begin{aligned}
y_1 &= y_0 + h \cdot (-5) y_0 = (1-5h) y_0, \\
y_2 &= y_1 + h \cdot (-5) y_1 = (1-5h) y_1 = (1-5h)^2 y_0, \\
y_3 &= (1-5h) y_2 = \ldots = (1-5h)^3 y_0, \\
\ldots &\quad \ldots \quad \ldots \\
y_k &= (1-5h) y_{k-1} = \ldots = (1-5h)^k y_0.
\end{aligned}$$

Das explizite Euler-Verfahren berechnet also den jeweils nächsten Wert aus dem vorhergehenden Wert durch Multiplikation mit $(1-5h)$.

Für $h = 0{,}41$ ist $(1-5h) = -1{,}05$, und die Folge $(1-5h)^k$ strebt für $k \to \infty$ mit alternierendem Vorzeichen gegen $\pm\infty$.

Andererseits, für $h = 0{,}39$ ist $(1-5h) = -0{,}95$, und die Folge $(1-5h)^k$ pendelt zwar zwischen positiven und negativen Werten, strebt aber für $k \to \infty$ nach Null.

Analysiert man das implizite Euler Verfahren, so kommt man auf folgende Rechnung:

$$\begin{aligned}
y_1 &= y_0 + h \cdot (-5) y_1, \quad \to \quad y_1 = \frac{y_0}{(1+5h)}, \\
y_2 &= y_1 + h \cdot (-5) y_2, \quad \to \quad y_2 = \frac{y_1}{(1+5h)} = \frac{y_0}{(1+5h)^2}, \\
y_3 &= y_2 + h \cdot (-5) y_3, \quad \to \quad y_3 = \frac{y_2}{(1+5h)} = \frac{y_0}{(1+5h)^3}, \\
\ldots &\quad \ldots \quad \ldots \\
y_k &= \ldots = \frac{y_{k-1}}{(1+5h)} = \frac{y_0}{(1+5h)^k}.
\end{aligned}$$

Hier gilt für $h = 0{,}41$: $\frac{1}{(1+5h)} = 0{,}327$, und und die Folge $\frac{1}{(1+5h)^k}$ strebt für $k \to \infty$ monoton fallend nach Null.

Tatsächlich gilt hier: Für alle $h > 0$ strebt $\frac{1}{(1+5h)^k}$ monoton nach Null.

Analysiert man das Heun-Verfahren, kommt man auf folgende Folge:

$$\left\{ \left(1 - 5h + \frac{(5h)^2}{2} \right)^k y_0 : k \in \mathbb{N} \right\}.$$

Für $h = 0{,}41$ ist der Ausdruck innerhalb der Klammer 1,051 und deswegen wächst die Folge monoton an und divergiert.

Für $h = 0{,}39$ ist der Ausdruck innerhalb der Klammer 0,951, die Folge konvergiert nach Null.

Stabilitätsgebiet

Für systematische Untersuchungen der Stabilität ist nützlicher, eine allgemeinere Modellgleichung $y' = \lambda y$ mit Parameter λ zu betrachten. Dabei untersucht man auch komplexe λ; die zugehörigen exakten Lösungen sind gedämpfte Sinus-Schwingungen. Diese einfache Modellgleichung beschreibt also, je nach λ-Wert, die wichtigsten Prozesse, die in praktischen Anwendungen auftreten.

Je nach Kombination von h und λ können Verfahren stabil sein oder nicht. Dabei kommt es aber nur auf den Wert des Produkts $\xi = h \cdot \lambda$ an. Man definiert daher:

Das Stabilitätsgebiet eines Verfahrens ist die Menge der komplexen Zahlen $\xi = h \cdot \lambda$, für die es bei der Lösung der Testgleichung

$$y' = \lambda y, \quad y(0) = 1$$

bei fester Schrittweite h eine beschränkte Folge von Näherungen liefert.

Im Falle des expliziten Euler Verfahrens gilt

$$\begin{aligned} \mathcal{B} &= \left\{ \lambda \in \mathbb{C} : \lim_{k\to\infty} y_k^\lambda < \infty \right\} = \left\{ \lambda \in \mathbb{C} : \lim_{k\to\infty} (1+\lambda)^k < \infty \right\} \\ &= \{ \lambda \in \mathbb{C} : |1+\lambda| \le 1 \} = \left\{ a + ib = \lambda \in \mathbb{C} : (a+1)^2 + b^2 \le 1 \right\}. \end{aligned}$$

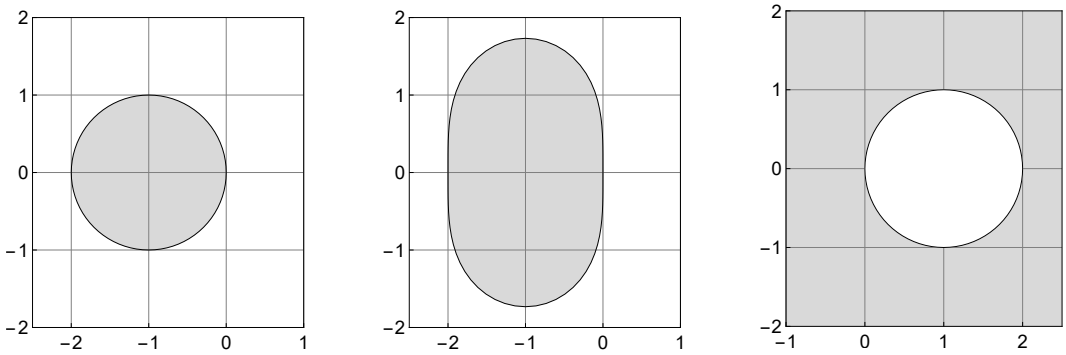
Im Falle des impliziten Euler Verfahrens gilt

$$\begin{aligned} \mathcal{B} &= \left\{ \lambda \in \mathbb{C} : \lim_{k\to\infty} y_k^\lambda < \infty \right\} = \left\{ \lambda \in \mathbb{C} : \lim_{k\to\infty} (1+\lambda)^{-k} < \infty \right\} \\ &= \left\{ a + ib = \lambda \in \mathbb{C} : (a-1)^2 + b^2 > 1 \right\}. \end{aligned}$$

Die Stabilitätsgebiete des expliziten, des modifizierten und des impliziten Eulerverfahrens sind hier (von links nach rechts) als graue Bereiche in der komplexen Zahlenebene dargestellt. Das Heun-Verfahren hat gleiches Stabilitätsgebiet wie das modifizierte Euler-Verfahren.

Praktisch relevant ist die linke Halbebene[16]; sie entspricht abklingenden Exponential- und Schwingungsprozessen. Für Stabilität muss λh in den grauen Bereichen liegen.

[16] In die rechte Halbebene fallen exponentiell anwachsende Prozesse. Dabei wachsen auch die Fehler aller numerischen Verfahren exponentiell an. Die Frage nach Stabilität ist hier nicht sinnvoll.



Bei expliziten Verfahren gibt es Einschränkungen: man muss die Schrittweite genügend klein wählen, sonst liegt λh außerhalb des Stabilitätsgebietes. Beim impliziten Verfahren liegt die gesamte linke Halbebene im Stabilitätsgebiet, man kann bei $\lambda \leq 0$ (abklingenden Exponential- und Schwingungsprozesse) die Schrittweite beliebig groß wählen.

10.5.2 Steifheit

Differentialgleichungen, die chemische oder physikalische Prozesse beschreiben, haben oft Lösungen, die sich aus sehr unterschiedlich schnell abklingenden Komponenten zusammensetzen. Das passiert, wenn Teilprozesse mir sehr unterschiedlichen Geschwindigkeiten ablaufen. Nehmen wir folgendes einfache Beispiel

$$\begin{aligned} \dot{y}_1(t) &= -y_1(t) + 50y_2(t), \\ \dot{y}_2(t) &= -70y_2(t), \end{aligned}$$

mit Anfangswerten $y_1(0) = 1$ und $y_2(0) = 10$. Da die Matrix

$$\begin{pmatrix} -1 & 50 \\ 0 & -70 \end{pmatrix}$$

Eigenwerte $\lambda_1 = -1$ und $\lambda_2 = -70$ mit Eigenvektoren $v_1 = (1, 0)^T$ und $v_2 = -(50, 96)^T$ hat, erhält man als Lösung

$$y_1(t) = 8.24638e^{-t} - 7.2464e^{-70t}, y_2(t) = 10e^{-70x}.$$

Um die am schnellsten abklingende Komponente mit einer Genauigkeit von 10^{-3} mittels einer numerischen Lösung zu berechnen, muss die Schrittweite so gewählt werden, dass e^{-70h} mit $F(-70h)$ auf fünf Stellen übereinstimmt. Aber nach einer relativen kurzen Zeit ist e^{-70t} verglichen zu e^{-t} praktisch völlig abgeklungen. Trotzdem muss der Solver auch dann noch mit der sehr kleinen Schrittweite rechnen, obwohl die e^{-70t}-Komponente eigentlich gar nicht mehr da ist.

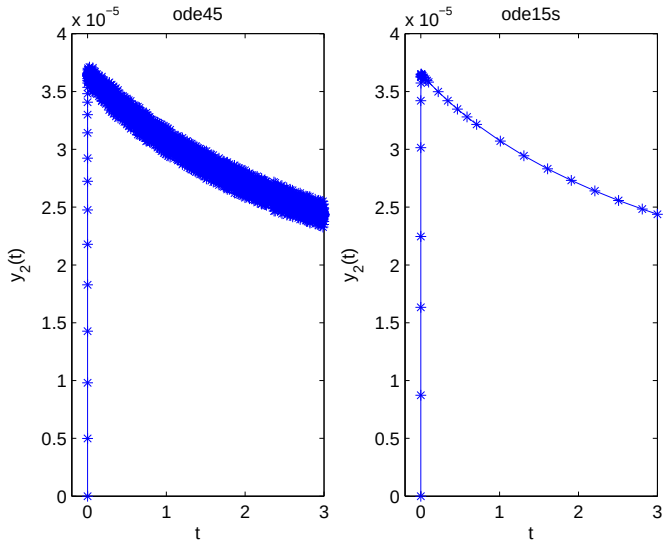
Definition 1: Ein lineares System

$$\begin{cases} \dot{x}(t) &= Ax(t), \\ x(0) &= x_0. \end{cases}$$

nennt man steif (stiff), falls der Wert

$$S := \frac{\max\{\Re(\lambda_j) : \lambda_j \text{ ist Eigenwert von } A \text{ mit negativen Realteil}\}}{\min\{\Re(\lambda_j) : \lambda_j \text{ ist Eigenwert von } A \text{ mit negativen Realteil}\}}$$

in der Größenordnung (oder größer als) 1000 ist.



Bemerkung 1: Man beachtet nur die Eigenwerte mit negativen Realteil, da diese Komponenten ergeben, die für grosse t gegen 0 laufen. Die Eigenwerte mit positiven Realteil führen zu Komponenten, die nicht konvergieren.

Ein weiteres (nicht konstruiertes) Beispiel ist folgende Differentialgleichung, die **Robertson Differentialgleichung**: Sei $y(0) = (1, 0, 0)$ und

$$\begin{aligned} \dot{y}_1(t) &= -0.04\, y_1(t) \\ &\quad +10^4 y_2(t)\, y_3(t), \\ \dot{y}_2(t) &= 0.04\, y_1(t) - 10^4 y_2(t)\, y_3(t) \\ &\quad -3 \times 10^7 y_2^2(t), \\ \dot{y}_3(t) &= 3 \times 10^7 y_2^2(t). \end{aligned}$$

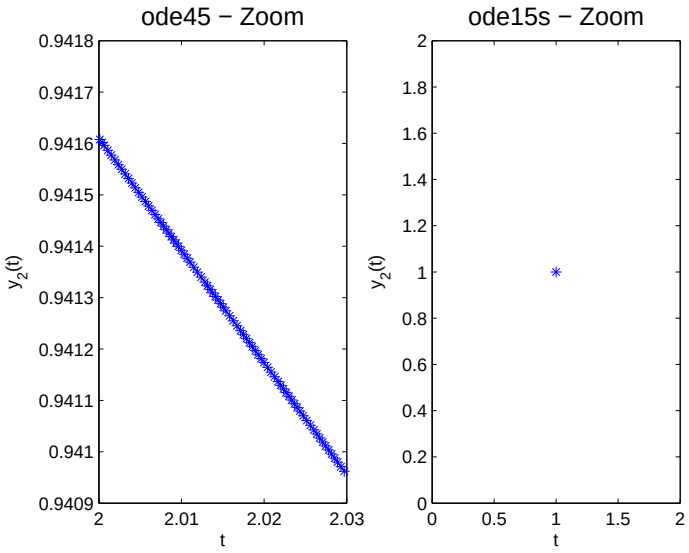
Am folgenden Bild sehen sie wie einmal die Differentialgleichung mit dem `ode45` numerisch berechnet, das andere mal mit dem `ode15s`. Obwohl der `ode45` ein Verfahren höherer Ordnung verwendet, konvergiert das Verfahren schlechter als das implizite Euler Verfahren, das der `ode15s` solver verwendet. Im zweiten Bild wurde ein Teilinterval herausgegriffen. In dem Zeitraum in dem der `ode45` Solver 81 Punkte berechnete, berechnete der `ode15s` solver einen Punkt.

Woran sieht man einen System an ob es sinnvoll wäre einen einfachen aber impliziten Solver zu verwenden.

Gegeben sei die lineare Differentialgleichung zweiter Ordnung

$$\begin{cases} \dot{x}(t) &= \begin{pmatrix} 0.5 & -0.5 \\ 10 & 10 \end{pmatrix} x(t), \\ x(0) &= \begin{pmatrix} 1 \\ 1 \end{pmatrix}. \end{cases} \tag{26}$$

Löst man dieses mit dem `ode45` solver auf dem Zeitinterval $[0, 20]$ diskretisiert MatLab die Zeit auf 265 Punkten, der `ode15s` solver berechnet die Funktion an nur 65 Punkten bei gleicher Genauigkeit. Schaut man das System genauer an, sieht man das die Matrix zwei Eigenwerte



-0.5 und -10 hat. Da der `ode45` solver ein explizites Verfahren ist, müssen die Zeitschritte in Richtung des Eigenwertes -10 aus Stabilitätsproblemen sehr klein gewählt werden, andererseits verläuft die Lösung in dieser Richtung sehr glatt und konvergiert sehr schnell gegen Null, sodass man eigentlich mit sehr großen Schritten vorangehen könnte. Der `ode15s` solver kann eine Schrittweite wählen die dem Problem angepasst ist.

Die meisten Systeme sind nicht linear, wie kann man aber jetzt den Begriff Steifheit (stiffness) auf ein nichtlineares System übertragen. Sei

$$\begin{cases} \dot{x}(t) &= f(x(t)), \\ x(0) &= x_0. \end{cases} \tag{27}$$

Die Steifigkeit für das linearisierte System zu definieren, indem man das *lokale* Verhalten der exakten Lösung $x(t)$ in der Umgebung eines Punktes t_0 analysiert. Sei

$$x(t) \quad = \quad x(t_0) + z(t), \quad t_0 \leq t \leq t_0 + h,$$

das linearisierte System an der Stelle t_0. Entwickeln wir System (27) mittels der Taylor approximation so folgt

$$\dot{x}(t) \quad = \quad f(x(t_0)) + \nabla f(x(t_0))(x(t_0) - x(t)) + O(x(t_0) - x(t)).$$

Setzt man für $x(t)$ die Näherung $x(t_0) + z(t)$ ein, erhält man

$$\dot{x}(t) \quad = \quad f(x(t_0)) + \nabla f(x(t_0))z(t) + O(x(t_0) - x(t)).$$

Andererseits gilt $\dot{x}(t) = \dot{x}(t_0) + \dot{z}(t) = f(x(t_0)) + \dot{z}(t)$. Dies oben eingesetzt ergibt

$$f(x(t_0)) + \dot{z}(t) \quad = \quad f(x(t_0)) + \nabla f(x(t_0))z(t) + O(x(t_0) - x(t)),$$

und damit

$$\dot{z}(t) \quad = \quad \nabla f(x(t_0))z(t).$$

Die infinitesimale Änderung zum Zeitpunkt t_0 löst also das obige lineare System. Andererseits kann man für ein lineares System die Steifigkeit wie oben definieren. Diese Definition kann man durch die obige Rechnung auch auf nichtlineare System übertragen.

Definition 2: Sei

$$A(x_0) := \frac{\partial f}{\partial x}\Big|_{x=x_0}.$$

und

$$S(x_0) := \frac{\max\{\Re(\lambda_j) : \lambda_j \text{ ist Eigenwert von } A(x_0) \text{ mit negativen Realteil}\}}{\min\{\Re(\lambda_j) : \lambda_j \text{ ist Eigenwert von } A(x_0) \text{ mit negativen Realteil}\}}$$

Ein System nennen wir steif in x_0, falls $S(x_0)$ in der Größenordnung von 1000 oder größer ist.

Beispiel 3: Robertson Differentialgleichung Sei $y(0) = (1, 0, 0)$ und

$$\begin{aligned}
\dot{y}_1(t) &= -0.04\, y_1(t) + 10^4 y_2(t)\, y_3(t), \\
\dot{y}_2(t) &= 0.04\, y_1(t) - 10^4 y_2(t)\, y_3(t) - 3 \times 10^7 y_2^2(t), \\
\dot{y}_3(t) &= 3 \times 10^7 y_2^2(t).
\end{aligned}$$

Das heißt mit $\mathbf{y} = (y_1, y_2, y_3)^T$ können wir auch schreiben

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

mit

$$f(\mathbf{x}) = f\left(\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}\right) = \begin{pmatrix} f_1(\mathbf{x}) \\ f_2(\mathbf{x}) \\ f_3(\mathbf{x}) \end{pmatrix} = \begin{pmatrix} -0.04\, x_1 + 10^4 x_2\, x_3 \\ 0.04\, x_1 - 10^4 x_2\, x_3 - 3 \times 10^7 x_2^2, \\ 3 \times 10^7 x_2^2 \end{pmatrix}$$

Die Jacobi Matrix lautet jetzt

$$\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} = \begin{pmatrix} \frac{df_1}{dx_1} & \frac{df_1}{dx_2} & \frac{df_1}{dx_3} \\ \frac{df_2}{dx_1} & \frac{df_2}{dx_2} & \frac{df_3}{dx_3} \\ \frac{df_3}{dx_1} & \frac{df_3}{dx_2} & \frac{df_3}{dx_3} \end{pmatrix} = \begin{pmatrix} -0.04 & 10^4\, x_3 & 10^4\, x_2 \\ 0.04 & -10^4 x_3 - 6 \cdot 10^7\, x_2 & -10^4 x_2 \\ 0 & 6 \cdot 10^7\, x_2 & 0 \end{pmatrix}$$

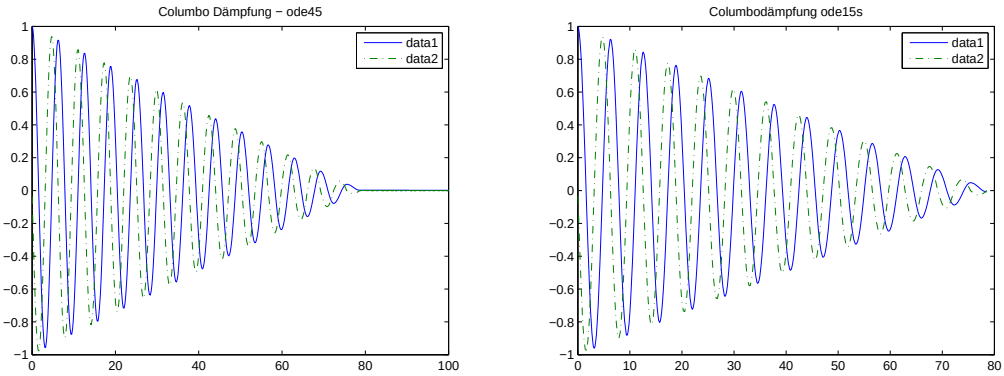
Berechnet man die Eigenwerte zum Punkt $\mathbf{x} = (1, 0, 0)^T$ so erhält man in Mathematika die Werte $-0.04, 0, 0$, zum Punkt $\mathbf{x} = (1, 0.05, 0.05)^T$, die Werte $-3 \cdot 10^6, -500, 1.4 \cdot 10^{-14}$, zum Punkt $\mathbf{x} = (1, 0.1, 0.1)^T$, die Werte $-6.0 \cdot 10^6, -1000, 5.26 \cdot 10^{-14}$). und $\mathbf{x} = (1, 1, 1)^T$ erhält man die Werte $-6 \cdot 10^7, -10000, 1.94 \cdot 10^{-12}$. Der Quotient zwischen den kleinsten und größten negativen reellen Eigenwert lautet jeweils

$$\begin{aligned}
S((1,0,0)^T) &\sim \infty, \\
S((1,0.1,0.1)^T) &\sim \frac{3 \cdot 10^6}{500} \sim 10^4, \\
S((1,0.1,0.1)^T) &\sim \frac{6.0 \cdot 10^6}{1000} \sim 10^3, \\
S((1,1,1)^T) &\sim \frac{6 \cdot 10^7}{10000} \sim 5 \cdot 10^2.
\end{aligned}$$

Beispiel 4: Die Coloumb-Dämpfung Gegeben sei folgende Differentialgleichung zweiter Ordnung:

$$m\ddot{x} + cx = -\mu m g\, \operatorname{sign}(\dot{x}). \tag{28}$$

Die Differentialgleichung wurde links mit dem ode45 solver berechnet. Die Anzahl der Stützstellen war 112221, die Differentialgleichung wurde rechts mit dem ode15s solver berechnet. Die Anzahl der Stützstellen war 945. Der Grund ist die Funktion $x \mapsto \operatorname{sign}(x)$, die in 0 nicht differenzierbar ist.



Eine Schwierigkeit bei impliziten Systemen ist die Auswertung der linken Seite. Angenommen, man berechnet das Beispiel oben mit den impliziten Euler Scheme. Es wird also zu jeden Zeitschritt folgendes Gleichungssystem gelöst:

$$\hat{\mathbf{y}}_k \quad = \quad \begin{pmatrix} \hat{y}_k^1 \\ \hat{y}_k^2 \end{pmatrix} = \hat{\mathbf{y}}_{k-1} + h \begin{pmatrix} \hat{y}_k^2 \\ -c\,\hat{y}_k^1 - \nu m g\,\mathrm{sign}(c\,\hat{y}_k^1) \end{pmatrix}.$$

Dieses System muss nach $\hat{\mathbf{y}}_k = (\hat{y}_k^1, \hat{y}_k^2)^T$ aufgelöst werden. Zumeist wird dies mit Fixpunktiteration gelöst, wobei zu berücksichtigen ist, dass die Funktion $x \mapsto \mathrm{sign}(x)$ nicht Lipschitzstetig ist.

11 Fourierreihen, Fourieranalyse, Fast Fourier Transform (FFT)

Es gibt im Hauptteil des Skriptums (noch) kein eigenes Kapitel dazu. Der Stoff wird anhand der Vorlesungsfolien (klick!) und im Übungsteil Ü 11 erklärt und erarbeitet.

Ü 1 Erste Übungseinheit

Vier wichtige Regeln:
1. Es funktioniert nie beim ersten Mal.
2. Es wird auch beim zweiten Mal wahrscheinlich nicht funktionieren.
3. Es funktioniert besser, wenn alle Kabel angesteckt sind.
4. Wenn sonst nichts mehr hilft, lesen Sie die Anleitung.

Ü 1.1 MATLAB

Die Übungen beginnen mit einer Einführung in die Rechen- und Programmierumgebung MATLAB (steht abkürzend für „MATrix LABoratorium“). Dieses Programm ist im universitären Bereich und in der industriellen Praxis ein Standardwerkzeug für technisch-wissenschaftliche Berechnungen. Für viele numerischen Aufgaben bietet MATLAB Lösungsfunktionen und Methoden zur Visualisierung. Gleichzeitig ist es eine moderne Programmiersprache, in der Sie eigene Anwendungen entwickeln können.

Diese Übungseinheit soll Ihnen eine rasche Einführung geben.

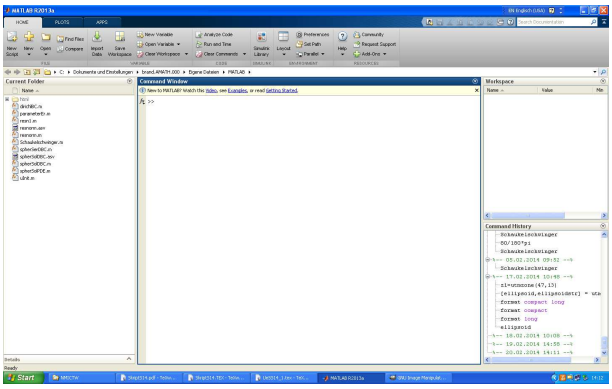
Inzwischen bietet auch die Programmiersprache Python mit ihren Zusatzpaketen (numpy, scipy) nahezu die gleiche Funktionalität wie MATLAB. Alle Übungsaufgaben lassen sich ebensogut in Python lösen. Wenn Sie mit Python vertraut sind, wird Ihnen numerisches Rechnen in MATLAB leicht fallen, und umgekehrt.

Einige weitere kostenlose Programmier- und Rechenumgebungen funktionieren ebenfalls mehr oder weniger ähnlich wie MATLAB: Scilab, Octave, FreeMat, Julia, R. Auch dafür gilt: Wenn Sie mit MATLAB gut vertraut sind, wird Ihnen der Einstieg zu diesen Softwarepaketen leicht fallen (und umgekehrt).

Ü 1.2 Der Anfang

Starten Sie MATLAB auf Ihren Rechnern wie übliche MS-Windows-Anwendungen durch Doppelklick auf das MATLAB-Zeichen. Sobald die MATLAB-Benutzeroberfläche (MATLAB *Desktop*) am Schirm erscheint, sind Sie bereit für die erste Lektion.

Sie sehen die MATLAB-Benutzeroberfläche mit der *Command Window* als großes Fenster in der Mitte.



Wenn die MATLAB-Benutzeroberfläche anders aussieht als hier nebenan abgebildet – es gibt sehr viele Konfigurationsmöglichkeiten – dann stellen Sie zuerst einmal die voreingestellte Standard-Oberfläche wieder her. Finden Sie das *Layout*-Symbol in der Mitte der oberen Leiste und klicken Sie auf *Layout–SELECT LAYOUT–Default*

Probieren Sie, wenn Sie wollen, einige der sonst noch angebotenen Möglichkeiten aus. Damit die Oberfläche wirklich so aussieht, wie hier gezeigt, müssen Sie *Layout–SHOW–Command History–Docked* einstellen.

Aber zum Schluss stellen Sie bitte mit *Default* das Standard-Layot wieder her. Sie sehen dann – von links nach rechts – vier Fenster: *Current Folder, Command Window, Workspace, Command History*. Wir arbeiten erst einmal im *Command Window*. Um die weiteren Fenster kümmern wir uns später.

Das *Command Window* liegt wie ein großes leeres Blatt vor Ihnen. Aber anders als in einem Textverarbeitungsprogramm können Sie nicht einfach irgendwohin schreiben, sondern immer nur in die aktuelle Befehlszeile. Die ist am Schirm durch die Eingabeaufforderung (den *prompt*) `>>` markiert.

Fangen Sie an und geben Sie die folgenden Befehle ein. Was Sie eintippen sollen, ist auch in den Unterlagen durch vorangestelltes `>>` markiert. Darunter folgt die Antwort von MATLAB (das, was Sie auf dem Schirm sehen sollten, nachdem Sie die Eingabetaste gedrückt haben).

```
>> 3-2

ans =

     1
```

Geben Sie 3-2 ein. Das Resultat wird unter dem Standard-Namen „ans“ (wie „eins“, oder „answer“, je nach Muttersprache) gespeichert.

```
>> ans

ans =

     1
```

Sie können das Resultat unter der Bezeichnung „ans“ jederzeit wieder abrufen...

```
>> zwa=ans+ans

zwa =

     2
```

...oder in weiteren Rechenausdrücken verwenden. Sie können den Wert eines Ausdrucks auch einer Variablen zuweisen.

```
>> y=2^2 + log(pi)*sin(zwa);
>> y

y =

    5.0409

>>
```

Sie berechnen $2^2 + \log(\pi) \cdot \sin(2)$. Ein **Strichpunkt** am Ende unterdrückt Ergebnis-Ausgabe und Zeilenvorschub. MATLAB erinnert sich trotzdem an y. Sie können den Wert von y jederzeit abrufen, indem sie einfach „y“ eintippen

```
>> format long g
>> y
y =
          5.04089993961332
>> y^10
ans =
          10594507.4098595
>>
>> format long e
>> y^10
ans =
     1.059450740985953e+07
>> format short g
>> y^10
ans =
   1.0595e+07

>>
```

MATLAB rechnet mit etwa sechzehnstelliger Genauigkeit. In der Standard-Einstellung zeigt es nur vier Nachkommastellen an. Sie können unterschiedliche Anzeigeformate einstellen. Das `e`-Format verwendet immer die Exponentialschreibweise, das `g`-Format schreibt, wenn möglich, das Ergebnis als Gleitkommazahl ohne Exponent.
`format compact` unterdrückt Leerzeilen bei der Ausgabe, damit passt mehr Ausgabe auf eine Bildschirmseite. `format loose` fügt Leerzeilen zur besseren Lesbarkeit ein.

```
>> theta=acos(-1)

theta =

    3.1416
```

MATLAB kennt trigonometrische Funktionen. Das ist der Arcus Cosinus von -1 (**Im Bogenmaß!**)

```
>> help elfun

  Elementary math functions.

  Trigonometric.
    sin         - Sine.
    .
    .
    .
```

MATLAB kennt eine große Menge sogenannter elementarer Funktionen. Sie können so eine Liste verfügbarer Funktionen aufrufen. Zu jeder einzelnen Funktion lässt sich weitere Hilfe anfordern.

Versuchen Sie, über die verschiedenen Möglichkeiten der MATLAB-Hilfe herauszufinden: Wie heißt die Funktion `sinh` mit vollem Namen?

Arbeitsauftrag: Verwenden Sie den *help browser* und zeichnen Sie einen Funktionsgraph Die ausgiebige MATLAB-Hilfe (der *help browser*) gibt mehr Information zu `sinh` und zeigt auch gleich, wie sich die Funktion plotten lässt.

Lassen Sie MATLAB nach der Anleitung in der Hilfe einen Funktionsgraph von `sinh` zeichnen.

Ü 1.2.1 Eingabe wiederholen, frühere Befehle kopieren

Wenn Sie sich irgendwo vertippen und die Eingabe wiederholen müssen, brauchen Sie nicht alles erneut eintippen. Sie können mit der Taste ↑ früher eingegebene Befehle hereinholen und gegebenenfalls korrigieren.

Wenn Sie den oder die ersten Buchstaben eines früheren Befehls eingeben und dann ↑ tippen, erinnert sich MATLAB und vervollständigt jene Befehle, die mit diesen Buchstaben begannen.

Sie können auch aus dem Fenster rechts unten, der *Command History*, Befehle mit Rechtsklick-Copy kopieren und mit Rechtsklick-Paste in das *Command Window* einfügen. Noch schneller geht Rechtsklick-*Evaluate Selection* oder Markieren-F9

Arbeitsauftrag: Zeichnen Sie Ihnen vertrautere Funktionen: Sinus, Cosinus, Exponentialfunktion, nach dem Muster des vorigen Abschnitts. Tippen Sie nicht alles wieder neu ein. Verwenden Sie die Pfeiltaste und/oder die *Command History*, um frühere Befehle herzuholen und nur den Funktionsnamen auszubessern.

Ü 1.2.2 Ausgabe unterdrücken, Schirm reinigen, Notbremse

Geben Sie folgenden Befehl ein:

```
>>  0:10000

ans =

  Columns 1 through 13

    0     1     2     3
  .
  .
  .
       9990        9991

  Columns 9997 through 10001

       9996        9997

>>
```

Dieser Befehl erzeugt eine lange Liste. Er dient hier nur als Beispiel für eine Anweisung, die den Schirm mit Unmengen von Output vollramscht. Solche Befehle sollten Sie einen Strichpunkt abschließen. Dann werden intern die Berechnungen durchgeführt, aber Bildschirmausgabe unterdrückt.

```
>>  0:10000;
>>
```

So ists besser.

Wenn aber doch einmal versehentlich eine Anweisung nicht und nicht enden will: Die Strg C -Taste beendet die laufende MATLAB-Berechnung. Sie können damit auch begonnene Befehlszeilen löschen.

Wenn MATLAB auch auf Strg C nicht reagiert, lässt es sich nur mehr über den *Windows Task Manager* stoppen. Dann ist aber die gesamte Sitzung mit allen bisher berechneten Werten verloren.

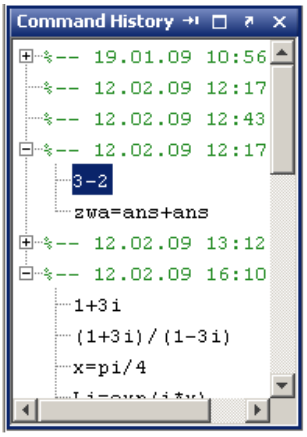
Wenn Sie alle Ein- und Ausgaben im *Command Window* löschen wollen, verwenden Sie den Befehl

```
>> clc
>>
```

Jetzt wird reiner Tisch gemacht

Ü 1.3 Speichern von Befehlen in einer Skript-Datei

Wenn Sie bis jetzt emsig gearbeitet haben, wollen Sie vielleicht auch Ihr Werk abspeichern. Das Fenster links unten, die *Command History*, hat alle Befehle soweit getreulich bewahrt. (Wenn Sie dieses Fenster nicht sehen, müssen Sie *Layout–SHOW–Command History–Docked* einstellen!)



Scrollen Sie durch die Befehle der *Command History*. Sie sehen alle von Ihnen eingegebenen Befehle. Wenn Sie weiter in die Vergangenheit zurückscrollen, finden Sie – durch Datum- und Zeitangabe gekennzeichnet – Befehle früherer Sitzungen. Sie lassen sich durch Anklicken öffnen oder schließen.

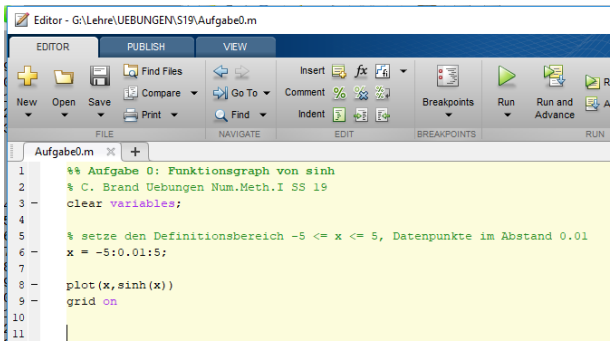
Angenommen, Sie wollen die Befehle der ersten Übungsaufgaben (Funktionsgraph von Sinus hyperbolicus zeichnen) speichern. Gehen Sie so vor:

Markieren Sie (Klick!) den ersten Befehl; halten Sie die Umschalttaste ⇑ gedrückt und scrollen sie in der Befehlsliste nach unten; markieren Sie (Klick!) den letzten Befehl. Die markierten

Befehle sind nun dunkelblau hinterlegt. Recktsklick→*Create Script*. Es öffnet sich ein Fenster des Editors.

Im Editor können Sie – anders als im *Command Window* – Befehle bearbeiten, wie in einem Texteditor. Sie können Zeichen an beliebiger Stelle einfügen und löschen

Empfehlenswert ist, *Kommentare* (beginnen mit %) einzufügen. Ihre Datei könnte dann so aussehen:



Speichern Sie Ihre Datei in der Windows-üblichen Weise (Menü *Save* – *Save as...*) – am besten gleich direkt auf einen mitgebrachten USB-Stick. MATLAB Skript-Dateien bekommen dabei automatisch den Typ `-.m`.

Mit dem Start-Symbol oben in der Menüleiste können Sie ihre Befehle als *Skript-M-Datei* starten. Das wirkt so als würden Sie die Befehle direkt ins *Command Window* eingeben. Die entsprechende Ausgabe erscheint im *Command Window*.

Die Skript-Datei soll aber tatsächlich alle Befehle enthalten, die zur Lösung einer Aufgabe notwendig sind. Typischer Anfängerfehler: das Skript verwendet Variablen, die im Skript nicht definiert sind. Es funktioniert, weil Sie die Variablen vorher definiert haben und daher die Variablen bereits im im Workspace vorhanden sind. Aber wenn Sie oder jemand anderer in einer neuen Matlab-Sitzung das Skript startet, kommt eine Fehlermeldung.

> Stellen Sie sicher, dass alle benötigten Variablen im Skript definiert werden!
> Tipp: Geben Sie ganz zu Beginn des Skripts den Befehl `clear variables`. Damit löschen Sie alle Variablen im Workspace und merken sofort, ob sie Definitionen im Skript vergessen haben

Aufgabe 1: Skript-Datei für Funktionsplot

Suchen Sie sich in MATLAB aus den Bereichen *Trigonometry, Exponents and Logarithms* oder, wenn Sie besonders neugierig sind, *Special Functions* eine Funktion aus, die Sie noch nicht kennen. Erstellen Sie ein Skript, das den Graphen dieser Funktion zeichnet. (Orientieren Sie sich am Screenshot auf Seite Ü-5)

> Tipp: Legen Sie sich für die Matlab-Dateien, die Sie in diesen Übungen erstellen, ein eigenes Unterverzeichnis an.
> Speichern Sie – falls Sie auf Uni-Rechnern arbeiten – am besten auf einem mitgebrachten USB-Datenträger!

Im Editor-Fenster sehen Sie, wenn die Skript-Datei fehlerfrei gespeichert ist, oben Mitte ein grünes „Run“-Pfeil-Symbol. Klick auf „Run“ führt alle Befehle im Skript erneut aus.

(Wenn eine Dialog-Box erscheint mit einem Text der Art *File D:/work/Aufgabe1.m is not found in the current folder blabla blabla*, dann klicken Sie einfach auf „Change Folder“)

Ü 1.4 Funktionen vom Typ $y = f(x)$ zeichnen

In diesem Abschnitt sollen Sie lernen, mit der Anweisung `plot` einfache Funktionsgraphen in der xy-Ebene (Die MATLAB-Hilfe spricht von „linear 2D-plots“) zu erstellen.

Arbeiten Sie im *Command Window* das folgende Beispiel durch. Es sollen die Graphen zweier Funktionen,

$$y = 3\cos x \quad \text{und } z = \log x$$

für den Definitionsbereich $0 < x \leq 25$ gezeichnet werden.

```
>> x=linspace(0.1,25,50)
```

Damit erzeugen Sie einen Zeilenvektor x, der 50 äquidistante Werte im Intervall $[0,1; 25]$ enthält. Entsprechend lang ist der Output am Schirm.

```
x =

  Columns 1 through 7

    0.1000    0.6082
1.1163    1.6245    2.1327
2.6408    3.1490
  .......

  Columns 43 through 49

   21.4429   21.9510   22.4592
22.9673   23.4755   23.9837
24.4918

  Column 50

   25.0000
```

Sie sehen, es werden tatsächlich 50 Werte erzeugt. In MATLABs Sichtweise ist x eine Matrix mit einer Zeile und 50 Spalten (deswegen „Columns“ im Output).

```
>> x=linspace(0.1,25,100);
```

Wiederholen Sie die Eingabe (mithilfe der Pfeiltaste), aber lassen Sie 100 Werte berechnen und fügen Sie einen Strichpunkt zum Abschluss an: Dadurch wird die Ausgabe unterdrückt und der Bildschirm bleibt übersichtlicher.
Übrigens werden, auch wenn Sie die Anzahl der Werte nicht angeben, also den Befehl in der Form `x=linspace(0.1,25);` verwenden, standardmäßig 100 Werte erzeugt.

```
>> y = 3*cos(x);
```

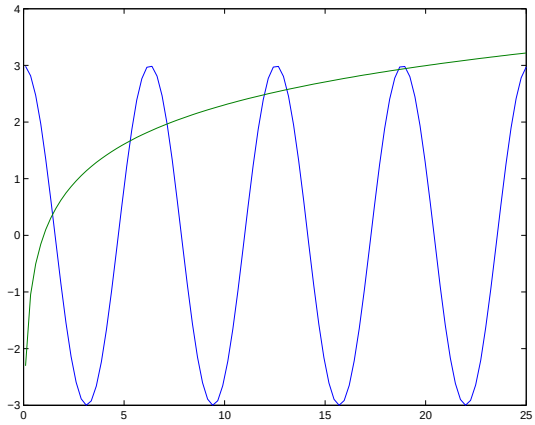
Dieser Befehl berechnet für jede Komponente im Vektor x den entsprechenden y-Wert.

```
>> plot(x,y)
```

Und damit erhalten Sie eine Zeichnung des Funktionsgraphen.

```
>> z=log(x);
>> plot(x,y,x,z)
```

Erzeugen Sie gleich noch einen zweiten Vektor, der den Funktionswerten von $\log x$ entspricht. Sie können beide Funktionsgraphen in ein Schaubild zeichnen. Achten Sie darauf, dass für die zweite Funktion nochmal die Angabe der x-Werte notwendig ist, obwohl beiden Funktionsgraphen die gleichen x-Werte zugrunde liegen.

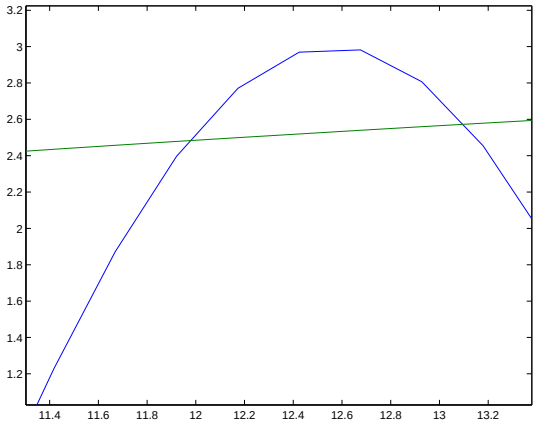


Der Graph sollte der obigen Abbildung entsprechen. Die Schnittpunkte der beiden Kurven entsprechen Lösungen der Gleichung

$$3\cos x = \log x$$

Vergleichen Sie dazu Abbildung 1 und Kapitel 1.4 im Vorlesungsskriptum, wo dieses Beispiel ausführlich diskutiert wird.

Wenn Sie die Zoom-in-Schaltfläche (mit dem Lupen-Symbol) anklicken, können Sie danach mit der Maus einen Bereich der Graphik vergrößert darstellen. Sie können also die Lösungen der Gleichung $3\cos x = \log x$ nähherungsweise aus der graphischen Darstellung ablesen. Das untenstehende Beispiel zeigt zwei Lösungen in der Nähe von $x = 11,9$ und $x = 13,1$.



Die Vergrößerung zeigt aber auch, dass nicht die „wirklichen“ Funktionen dargestellt werden, sondern die einzelnen Datenpunkten der Vektoren x, y und z stückweise durch Geraden verbunden sind. Deswegen kann man aus der Graphik die Lösungen der Gleichung $3\cos x = \log x$ natürlich nur im Rahmen der Zeichengenauigkeit darstellen.

Ü 1.5 Weitere Übungsbeispiele

Sie können, so weit Sie kommen, die folgenden Beispiele während dieser Übungseinheit ausarbeiten oder sonst auf Ihrem eigenen Rechner fertigstellen.

Speichern Sie die Aufgaben als Skript-Dateien auf einem USB-Datenträger, sodass Sie bei Befragungen Ihre Beispiele vorweisen können.

Aufgabe 2: Lösungen von $3\cos x = \log x$

Fertigen Sie als Skript-Datei eine genauere Version der obigen Funktionsgraphen mit 1000 Datenpunkten an. Geben Sie Ihrem Bild auch einen Titel und beschriften Sie die Achsen. (Informieren Sie sich in der MATLAB-Hilfe, beim Übungsleiter oder einer hilfreichen Nachbarin, wie das geht!)

Bestimmen Sie aus der Graphik alle Lösungen der Gleichung.

Hinweis: Beginnen Sie mit `x=linspace(0.1,25,1000);`

> Wenn Sie die Unterlagen in Papierform vor sich haben oder in Ihre digitale Kopie hineinschreiben können, tragen Sie hier Antworten ein. Schreiben Sie geforderte Antworten jedenfalls auch als Kommentarzeile in Ihr Skript.

Die Lösungen sind (auf drei Nachkommastellen genau):

_____

_____

Aufgabe 3: Quadratische Gleichung

Gesucht ist die betragskleinere Wurzel von

$$x^2 - 12345678x + 9 = 0$$

Verwenden Sie Rechenbefehle im *command window* und speichern Sie zum Schluss in übersichtlicher und kommentierter Form als Skript-Datei.

(Sinnvoller Weise setzen Sie dazu das Anzeigeformat `format long e`)

- Die gängige Lösungsformel $x_{1,2} = -p/2 \pm \sqrt{p^2/4 - q}$ liefert

  _____

- Die numerisch korrekte Berechnung der betragsmäßig *kleineren* Lösung einer quadratischen Gleichung berechnet zuerst die *betragsgrößere* Lösung x_{gr} mit der Standard-Formel. Die betragskleinere Lösung x_{kl} ergibt sich dann aus

  $$x_{\mathrm{kl}} = \frac{q}{x_{\mathrm{gr}}}$$

  Ergebnis: _____

- Auflösen nach dem linearen Term,

  $$x = \frac{x^2 + 9}{12345678}$$

  Fixpunkt-Iteration in der Form

```
>> x=0;
>> x = (x^2+9)/12345678
x =
    7.290000597780049e-007
```

Wiederhole, bis sich Wert nicht mehr ändert

Ergebnis: _____

Aufgabe 4: Fixpunkt-Iteration:

Finden Sie wie in der vorigen Aufgabe durch wiederholtes Auswerten einen Fixpunkt der Funktion $\phi(x) = 1/\exp x$. Startwert 5, Genauigkeit `format long e`.

Ergebnis: _____

Untersuchen Sie auch, wie sich von einem Schritt zum nächsten die Genauigkeit erhöht. Können Sie eine Regel der Art „pro Dezimalstelle Genauigkeit sind durchschnittlich x Iterationen notwendig“ finden?

Aufgabe 5: Wurzelbehandlung:

Schon die alten Babylonier berechneten Quadratwurzeln $\sqrt{a}$ mit der (oft als Heron-Verfahren bezeichneten) Iteration

$$x^{(0)} = a; \quad x^{(k+1)} = \frac{1}{2}\left(x^{(k)} + \frac{a}{x^{(k)}}\right) \text{ für } \quad k = 0, 1, 2, \ldots$$

Berechnen Sie $\sqrt{2}$ durch wiederholtes Auswerten der Iterationsvorschrift (Genauigkeit `format long e`). Testen Sie auch andere Wurzelberechnungen, z.B. $\sqrt{2005}$, $\sqrt{4711}$, $\sqrt{0{,}815}$ ... Untersuchen Sie bei allen Beispielen, wie sich von einem Schritt zum nächsten die Anzahl richtiger Stellen erhöht. Welche der folgenden Regeln beschreibt das Konvergenzverhalten am besten?

1. Der Fehler halbiert sich mit jeder Iteration
2. Pro Iteration gewinnt man zwei korrekte Dezimalstellen
3. Pro Iteration gewinnt man drei korrekte Dezimalstellen
4. Pro Iteration verdoppelt sich annähernd die Anzahl korrekter Stellen

Aufgabe 6: Newton-Verfahren:

Finden Sie mit dem Newton-Verfahren für $f(x) = x \tan x - 1$ die Nullstelle in der Nähe des Startwertes $x^{(0)} = 1$. Anleitung:

```
>> x=1;
>> f = x*tan(x)-1
f =
    5.574077246549023e-001
>> fstr = ...
fstr =
    4.982926545469661e+000
>> x = x - f/fstr
x =
    8.881364757099055e-001
```

Hinweis: Die Ableitung von $\tan x$ ist $1/\cos^2 x$.

Wiederholen Sie, bis das Ergebnis auf die volle Stellenanzahl genau ist. Prüfen Sie anhand des Endresultates die Anzahl korrekter Dezimalstellen bei den einzelnen Iterationsschritten.

Untersuchen Sie das Konvergenzverhalten (die zunehmende Anzahl korrekter Stellen), indem Sie eine Tabelle erstellen:

Schritt	korrekte Dezimalstellen
0	0
1	
2	
⋮	

Wie lässt sich das Konvergenzverhalten beschreiben?

Die wiederholte Ausführung eines Befehls der Art `x = x - f/fstr` in einem Programm schreit nach einer Schleife. MATLAB kennt natürlich solche Kontrollstrukturen, und wenn Sie damit vertraut sind, dann schreiben Sie in Ihrem Skript eine `for`-Schleife. Sonst reicht es auch, die Anweisung mit copy/paste mehrfach zu wiederholen.

So könnte in einer Skript-Datei der Code aussehen, wenn Sie eine Schleife verwenden:

```
format long e
x=1
for i=1:7
    f = x*tan(x)-1;
    fstr =  ...
    x = x-f/fstr
end
```

Aufgabe 7: Kepler-Gleichung

Die Vorlesungsfolien zur 1. Vorlesung zeigen als Beispiel die Kepler-Gleichung

$$x - \epsilon \sin x = m$$

(sie setzt verschiedene Parameter einer elliptischen Umlaufbahn in Beziehung – aber das müssen Sie nicht wissen!) Angenommen, $\epsilon = 1/10$ und $m = 2$ sind gegeben; x ist gesucht. Verschiedene Lösungswege sind möglich:

- Ablesen aus geeigneter graphischer Darstellung.
- Durch Fixpunkt-Iteration
- Newtonsches Verfahren
- Sekanten-Methode
- Intervallhalbierung

Suchen Sie sich zwei davon aus und schreiben Sie dazu ein Skript. Ihr Skript soll eine Folge von Näherungswerten ausgeben. Auf den Vorlesungsfolien sind auch Zahlenwerte angegeben, damit können Sie vergleichen.

Analysieren Sie das Konvergenzverhalten ihres Verfahrens und geben Sie an, wie sich der Fehler von einem Schritt zum nächsten reduziert.

Ü 1.6 Kurven vom Typ $\mathrm{x} = \mathrm{f}(\theta); \mathrm{y} = \mathrm{g}(\theta)$ zeichnen

Eine Funktion $f : x \mapsto f(x)$ weist jedem x-Wert genau einen y-Wert zu. Ein Kreis lässt sich in dieser Form nicht beschreiben, weil hier zum gleichen x-Wert zwei y-Werte gehören können.

In solchen Fällen kann man Kurven in Parameterform darstellen. Dabei sind sowohl die x- als auch die y-Werte Funktion eines Parameters (der hier θ heißt).

Wir wollen einen Kreis mit Radius 1 zeichnen. Damit der Kreis auch wirklich jenes Aussehen hat, das wir erwarten, erzeugen wir 100 Punkte, die auf dem Kreis liegen. Dazu verwenden wir die Beziehungen:

$$x = \cos\theta, \qquad y = \sin\theta, \qquad 0 \le \theta \le 2\pi$$

Die Folge der MATLAB-Anweisungen, die unsere Aufgabe lösen, könnte wie folgt aussehen:

```
>> theta = linspace(0, 2*pi, 100);
```
Erzeugt 100 äquidistante Werte zwischen 0 und 2π.

```
>> x=cos(theta);
>> y=sin(theta);
```
Erzeugt die x- und y- Koordinaten der 100 Punkte.

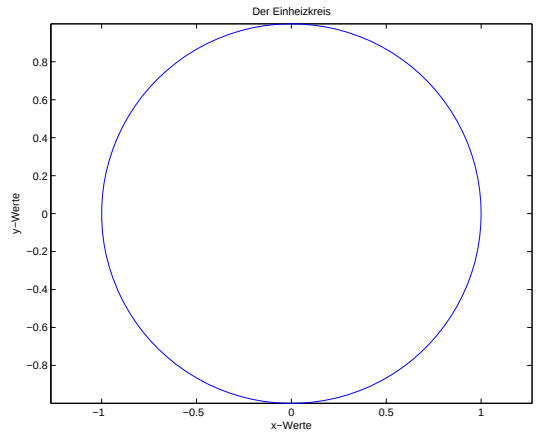
```
>> plot(x,y);
```
Zeichnet die Verbindunglinien von Punkt 1 zu Punkt 2 zu ... Punkt 100

Weitere Befehle, mit denen Sie Ihre Zeichnung formatieren können, sind

```
>> axis('equal');
```
Setzt die Skalierung der beiden Achsen gleich.

```
>> xlabel('x-Werte')
>> ylabel('y-Werte')
>> title('Der␣Einheizkreis')
```
Die x- und y-Achse werden beschriftet und Ihr Werk betitelt. (Strichpunkt oder nicht als Abschluss macht bei diesen Befehlen keinen Unterschied.)

Ihr Werk sollte nun so aussehen:



Sie können die Graphik auch interaktiv formatieren, wie Sie es von anderen Windows-Anwendungen gewohnt sind. Oben, neben dem Drucker-Symbol finden Sie die Schaltfläche „Edit plot“. Alternativ können Sie auch aus dem Menue „Tools–Edit Plot“ wählen. Danach können Sie durch Doppelklick Achsen oder andere Elemente der Zeichnung weiter editieren. Andere Schaltflächen erlauben Ihnen, Text, Pfeile oder Linien einzufügen.

Wichtig sind noch die Möglichkeiten, die Graphik zu speichern. Unter dem Menüpunkt „File–Export“ lässt sich die Graphik in verschiedenen Formaten speichern.

Ü 1.7 Elementweise Rechenoperationen

Es ist Ihnen vielleicht noch gar nicht bewusst gewesen: Sie haben beim Auswerten von Funktionstermen Rechenoperationen auf Datenvektoren angewandt. Der Befehl `x = cos(theta)` funktioniert sowohl für skalares θ, als auch für einen Datenvektor mit n Elementen. Auch Addition und Subtraktion lassen sich gleichermaßen auf skalare wie vektorielle Operanden anwenden.

Für Multiplikation, Division und die Potenzfunktion müssen Sie beim Verknüpfen von Datenvektoren aber unbedingt die elementweisen Operatoren `.*` `./` `.^` verwenden. Grund: Die „gewöhnlichen“ Operator-Symbole `*` `/` `^` verknüpfen in MATLAB Matrizen und Vektoren nach den Regeln der Matrix-Algebra. Diese Regeln wollen Sie *nicht* anwenden, wenn Sie Datenvektoren in Funktionsterme einsetzen.

Anfangs ist es oft schwer zu durchblicken, wann MATLAB Rechenoperationen von sich aus elementweise interpretiert, und wann man explizit den Punkt setzen muss.

> **Faustregel:** in Funktionstermen immer die elementweisen Operatoren verwenden, wenn Datenvektoren verknüpft werden.

Die Aufgaben 8 und 10 erfordern diese Unterscheidung. Dazu noch Hinweise:

```
>> t = linspace(0, 2*pi, 100);
>> x = sin(t)*2 - sin(t*2);
```

Erster Funktionsterm in Aufgabe 8: Hier reicht der „gewöhnliche“ `*` Operator, weil er nur in Operationen vom Typ „Skalar mal Vektor“ auftritt.

```
>> x = sin(t).*2 - sin(t.*2);
```

Es wäre aber auch nicht falsch, die Punkt-Form zu verwenden. Das Ergebnis bleibt gleich.

```
>> r = 2 - 2*sin(t) + sin(t) .* abs(cos(t)).^0.5 ./ (sin(t)+7/5)
```

Ein anderer Funktionsterm in Aufgabe 8: Aber hier muss überall, wo Vektoren verknüpft werden, ein Punkt-Operator stehen!

Ü 1.8 Weitere Übungsbeispiele

Dokumentieren Sie Ihre Übungsbeispiele als Skript-M-Dateien. Speichern Sie Zeichnungen als `JPEG`-Dateien ab, dann sind diese Dateien außerhalb MATLABs mit jedem Bildbetrachtungsprogramm anzusehen. Wenn Sie Zeichnungen als `.fig`-Dateien abspeichern – das ist MATLABs Voreinstellung – dann können Sie diese Zeichnungen nur innerhalb Matlabs öffnen.

Aufgabe 8: Zum Valentinstag (ist zwar schon vorbei, aber besser spät als nie)

Zeichnen Sie die Kardioide oder Herzkurve. Das ist eine Kurve, die durch folgende Parameterdarstellung für $0 \leq t < 2\pi$ gegeben ist.

$$x = 2\sin(t) - \sin(2t)$$
$$y = 2\cos(t) - \cos(2t)$$

Der Name Kardioide geht auf Giovanni di Castiglione, einen italienischen Mathematiker im 18. Jahrhundert, zurück. Für mich sieht sie mehr wie ein Apfel ohne Stängel aus. Eine schönere Herzkurven-Gleichung ist hier gegeben[17]:

$$r = 2 - 2\sin t + \sin t \frac{\sqrt{|\cos t|}}{\sin(t) + 7/5}$$
$$x = r\cos(t)$$
$$y = r\sin(t)$$

Hinweis: Verwenden Sie Sie für $0 \leq t < 2\pi$ mindestens 500 Teilpunkte und achten Sie darauf, an den richtigen Stellen `.*` und `./` zu verwenden!

Aufgabe 9: 3D-Plots

Zeichnen Sie mittels `plot3(x,y,z)` eine Spirale:

$$x(t) = \sin(t), \quad y(t) = \cos(t), \quad z(t) = t, \quad 0 \leq t \leq 20.$$

Verwenden Sie die Schaltfläche „Rotate 3D“, um Zeichnungen der Kurve aus zwei verschiedenen Blickwinkeln anzufertigen.

Aufgabe 10: Für die Fisch

Diskutieren Sie die Funktion

$$y = 3 - \frac{1}{2}\arctan(\frac{2\,x - 5}{x - 2}) + \frac{1}{10}\sin(7\,x) \quad 0 \leq x \leq 4$$

anhand eines Graphen. Wo liegen Supremum und Infimum? In welchen Bereichen ist die Funktion stetig? differenzierbar? An welcher Stelle ragt eine Haifischflosse aus den Wellen?

Wenn die letzte Frage für Sie sinnlos erscheint, haben Sie die Funktion vermutlich falsch gezeichnet. Beachten Sie bei der Berechnung der y-Werte, dass alle Rechenoperationen elementweise (für jeden einzelnen x-Wert) berechnet werden müssen. Wenn einer der Partner in einer Rechenoperation ein Skalar, der andere ein Vektor ist, funktioniert das automatisch, wie z.B. in der Anweisung `2*x-5`. Wenn beide Partner Vektoren sind, müssen Sie unbedingt die elementweisen Operatoren `.*` und `./` verwenden. Beispiel: der Ausdruck $(2x-5)/(x-2)$ muss geschrieben werden `(2*x-5)./(x-2)`.

[17] Quelle: `http://mathworld.wolfram.com/HeartCurve.html`

Ü 2 Zweite Übungseinheit

Inhalt der zweiten Übungseinheit:

- Zeilen- und Spaltenvektoren
- Matrizen, Abbildungen
- Funktionen
  - Funktionen als Funktions-M-Datei programmieren
  - Fixpunkte und Nullstellen von Funktionen
  - Befehle `fzero, roots`
  - Funktions-Einzeiler (*anonymous functions*)
- Kontrollstrukturen
  - Schleifen
  - Verzweigungen
- Fixpunkt-Iteration, ein- und mehrdimensional

Ü 2.1 Zeilen- und Spaltenvektoren

Sie haben in der vorigen Einheit bereits mit Vektoren gearbeitet: Beim Zeichnen von Funktionsgraphen haben Sie Bereiche für x- und y-Werte als Zeilenvektoren erzeugt. MATLAB unterscheidet zwischen Zeilen- und Spaltenvektoren.

Hier folgt eine kurze Zusammenfassung zum Erzeugen von und Rechnen mit Vektoren. Klicken Sie sich durch diese Anleitung, dann lernen Sie die wichtigsten Befehle dazu kennen. Überlegen Sie sich bei jedem Ergebnis, welche Rechenregel MATLAB dabei verwendet hat!

- Standard-Operationen der Vektorrechnung (Addition `+` , Subtraktion `-` , Multiplikation `*` mit Skalar, inneres Produkt `*` von Zeilen- mit Spaltenvektor, Vektor transponieren `'`)
- elementweise Multiplikation `.*` und Division `./` bei Vektoren gleicher Form. Ergebnis ist ein Vektor gleicher Form; die entsprechenden Elemente werden multipliziert bzw. dividiert.
- Wenn ein Operand Zeilen- und der andere Spaltenvektor ist, erzeugt Addition oder Subtraktion eine Matrix nach dem Muster „jedes Element mit jedem“.
- Bei der Multiplikation kommt es drauf an: linker Partner Zeile, rechter Partner Spalte berechnet inneres Produkt; linker Partner Spalte, rechter Partner Zeile berechnet eine Matrix nach dem Muster „jedes Element mit jedem“ (heißt auch äußeres, Kronecker- oder Tensor-Produkt - da ist sich die Fachwelt nicht einig).
- Bei der Division `/` zweier Zeilen- bzw. zweier Spaltenvektoren passieren völlig bizarre Dinge – was MATLAB da berechnet, versuchen wir hier erst gar nicht zu erklären.

Vektoren erstellen

```
>> x=[1 2 3]
x =
     1     2     3
```

x ist ein Zeilenvektor mit drei Elementen.
Vektoren sind von eckigen Klammern `[ ]` begrenzt.

```
>> x=[1, 2, 3]
x =
     1     2     3
```

So geht 's auch: Die einzelnen Komponenten in einer Zeile können Sie durch Leerzeichen (so wie gerade vorher) oder durch Beistriche (so wie hier) trennen.

```
>> y=[2;1;5]
y =
     2
     1
     5
```

y ist ein Spaltenvektor mit drei Elementen. Die Strichpunkte ; trennen die Zeilen im Vektor.

```
>> y = [2
1
5]
y =
     2
     1
     5
```

Statt des Strichpunkten können Sie im *command window* oder in Dateien auch eine neue Zeile beginnen.

Rechenoperationen

```
>> z=[2 1 0];
>> a=x+z

a =

     3     3     3
```

Sie können zwei Zeilen- oder zwei Spalten-Vektoren addieren oder subtrahieren – das entspricht den Standardregeln der Vektorrechnung

```
>> b=2*a

b =

     4     4     0
```

Skalar mal Vektor, das ist eine Standard-Regel

```
>> x/2
ans =
    0.5000    1.0000    1.5000
```

Division Vektor durch Skalar, ebenfalls Standard

```
>> 2/x
Error using  /
Matrix dimensions must agree.
```

in dieser Reihenfolge nicht möglich

```
>> 2./x
ans =
    2.0000    1.0000    0.6667
```

Das geht: Elementweise Division

```
>> b=x+y
b =
     3     4     5
     2     3     4
     6     7     8
```

Addition von Zeilen- plus Spaltenvektor ergibt eine Matrix – Das ist nicht gerade Standard, sondern MATLABs kreative Erweiterung des Plus-Operators.

Achtung bei älteren MATLAB-Versionen! Versionen vor R2016b erlauben nicht, Zeilen- zu Spaltenvektoren zu addieren. Obiger Befehl ergibt eine Fehlermeldung

```
» b=x+y
Error using +
Matrix dimensions must agree.
```

Die kreative Interpretation der Standard-Regeln in den neueren Versionen gilt für viele arithmetische und logische Operatoren. Regel: Wenn es irgendwie sinnvoll erscheint, erweitert MAT-

LAB die Operanden so, dass eine elementweise Operation möglich wird[18].

MATLAB führte dieses automatische Erweitern 2016 mit der Begründung: [19] ein: *"MATLAB has a long history of inventing notation that became widely accepted"* – die Kommentare im Blog sind eher kontroversiell. Nicht alle Anwender freut so einen lockerer und origineller Umgang mit den Rechenregeln.

> Vorsicht bei Rechenoperationen mit Vektoren! Wenn die Formate von Zeilen- und Spaltenvektoren nicht den Regeln der Standard-Vektor- und Matrizenrechnung entsprechen, findet MATLAB oft eine kreative, aber meist ungewollte Interpretation der Anweisungen.

```
>> a=x.*z
a =
     2     2     0
```

Sie können zwei Vektoren derselben Größe elementweise multiplizieren (oder dividieren); array operator `.*` (oder `./`)

```
>> x*y
ans =
    19
```

Skalares (oder inneres) Produkt. Standardregel für Zeilen- mal Spaltenvektor

```
>> y*x
ans =
     2     4     6
     1     2     3
     5    10    15
```

Spalten- mal Zeilenvektor ergibt Matrix. Heißt äusseres, Tensor- oder Kronecker-Produkt; auch das ist eine Standard-Rechenoperation.

```
>> x.*y
ans =
     2     4     6
     1     2     3
     5    10    15
```

Elementweise Multiplikation von Zeilen- mit Spaltenvektor ergibt dieselbe Matrix wie oben – MATLABs kreative Interpretation.

Vektoren transponieren

```
>> x
x =
     1     2     3
>> x'
ans =
     1
     2
     3
>> y
y =
     2
     1
     5
>> y'
ans =
     2     1     5
```

Höchste Zeit, dass Sie den ' Operator kennenlernen. Damit transponieren Sie Vektoren: Zeilen- wird Spaltenvektor, und umgekehrt.

[18] Für eine genauere Erklärung suchen Sie in der MATLAB Dokumentation nach *Compatible Array Sizes for Basic Operations*

[19] `https://blogs.mathworks.com/loren/2016/10/24/matlab-arithmetic-expands-in-r2016b/`

Im Matlab-Desktop sehen Sie übrigens die Registerkarte „Workspace“ links in der Mitte, und wenn Sie draufklicken, darüber ein Fenster. (Möglicherweise ist das Fenster schon offen, ohne dass Sie extra angeklickt haben.) Alle bisher verwendeten Variablen sind dort aufgelistet. Das Matrix-Symbol neben dem Variablennamen erinnert daran, dass MATLAB alle Variablen als Matrizen interpretiert - Skalare als 1×1-Matrizen, Zeilenvektoren mit n Elementen als $1 \times n$- und Spaltenvektoren mit m Einträgen als $m \times 1$-Matrizen. Doppelklick auf eine Zeile dieser Liste öffnet ein Fenster, den „Array Editor“, das die Werte der Variablen in Tabellenform anzeigt. Die Werte lassen sich (Doppelklick auf die entsprechende Zelle im Tabellenblatt) auch ändern.

Zeilenvektoren mit regelmäßigen Einträgen

Zum Erzeugen von Zeilenvektoren noch einige Beispiele:

```
>> x=linspace(0,10,5)

x =

         0    2.5000
5.0000    7.5000   10.0000
```

Dieser Befehl erzeugt einen Zeilenvektor der Länge 5, dessen Elemte äquidistant im Intervall [0,10] liegen. Ohne drittes Argument entsteht immer ein Vektor der Länge 100.

```
>> x=0:2.5:10
x =
         0    2.5000
5.0000    7.5000   10.0000
```

Auch mit dem Doppelpunkt-Operator lässt sich derselbe Zeilenvektor erzeugen nach dem Muster `Anfangswert:Schrittweite:Endwert`
Bei gewünschter Vektor-Länge ist `linspace` einfacher; wenn die Schrittweite vorgegeben ist, bietet sich die Doppelpunkt-Variante an.

```
>> x=1:6
x =
     1     2     3     4
5     6
```

Bei Schrittweite 1 ist die Doppelpunkt-Anweisung besonders einfach

Ü 2.2 Matrizen

Der Name MATLAB steht für *matrix laboratory*. Das signalisiert Ihnen: Das Arbeiten mit Matrizen ist in dieser Rechenumgebung ein ganz zentrales Thema. Hier eine kurze Einführung ins Erstellen von und in die Grundrechnungsarten für Matrizen.

Hoffentlich sind Ihnen die Rechenregeln der Matrizenrechnung noch in Erinnerung. Bevor Sie weiterarbeiten – ist Ihnen klar, wir die Multiplikation von Matrizen abläuft? Sie sollten mit Stift auf Papier nachrechnen können:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \cdot \begin{bmatrix} 5 & 6 & 7 \\ 8 & 9 & 0 \end{bmatrix} = \begin{bmatrix} 21 & 24 & \dots \\ 47 & \dots & ? \end{bmatrix}$$

Matrizen erstellen

```
>> A=[1 2; 3 4]
A =
     1     2
     3     4
>> B = [ 5, 6, 7
8, 9, 0]
B =
     5     6     7
     8     9     0
```

Eingabe: Leerzeichen oder Komma trennt Komponenten in einer Zeile; Strichpunkt oder neue Zeile trennt Zeilen.

Rechenoperationen, Zugriff auf Elemente

MATLAB bietet ähnlich wie für Vektoren folgende Operationen an:

`+ , -`	elementweise Addition, Subtraktion
`*`	multipliziert nach den Regeln der linearen Algebra
`.* , ./`	elementweise Multiplikation, Division
`A'`	Matrix `A` transponieren

Bei Rechenoperationen zwischen Matrizen ist wichtig, dass die Matrix-Dimensionen zueinander passen. Versuchen Sie für die gerade erstellten Matrizen `A` und `B`

```
>> C=A*B
C =
    21    24     7
    47    54    21
```

Matrix-Multiplikation nach den Regeln der linearen Algebra.

```
>> C=B*A
Error using  *
Incorrect dimensions ...
```

Innere Dimensionen müssen übereinstimmen.

Es is A eine 2×2-, B eine 2×3-Matrix. Bei $A \cdot B$ ist in $(2 \times 2) \cdot (2 \times 3)$ die innere Dimension 2. Bei $B \cdot A$ mit $(2 \times 3) \cdot (2 \times 2)$ passen die inneren Dimensionen nicht: $2 \neq 3$.

Elementweise Operationen lassen sich nur auf Matrizen gleicher Größe anwenden.

```
>> C=A+B
Arrays have incompatible sizes for this operation.
```

```
>> A(1,2)
ans =
     2
```

Zugriff auf Element a_{12}

```
>> B(1:2, 2:3)
ans =
     6     7
     9     0
```

Zugriff auf Elemente in Zeilen 1 bis 2, Spalten 2 bis 3

Ü 2.3 Der Doppelpunkt-Operator

Der Doppelpunkt (englisch: *colon*) ist einer der nützlichsten Operatoren im Umgang mit Matrizen und Vektoren. Er erzeugt Vektoren, Index-Bereiche und Iterationsindizes. Als Index-Ausdruck kann er aus Matrizen einzelne Zeilen oder Spalten herausgreifen.

In der MATLAB-Hilfe finden Sie Informationen dazu unter dem Stichwort *colon*.

Syntax-Beispiele

Erzeugen einfacher Zahlenfolgen

`x=3:8` erzeugt den Zeilenvektor `[3, 4, 5, 6, 7, 8]`

`x=0:2:6` erzeugt den Zeilenvektor `[0, 2, 4, 6]`

`x=10:-1:0` erzeugt den Countdown-Vektor `[10, 9, ..., 3, 2, 1, 0]`

Diese Syntax werden wir für Zählschleifen im Kapitel Ü 2.6 verwenden.

Index-Ausdrücke, Zugriff auf Vektor- und Matrixelemente

Als Matrix- oder Vektorindex bedeutet der Doppelpunkt soviel wie „alle Zeilen“ oder „alle Spalten“.

`A(:,4)` alle Zeilen, Spalte 4; die gesamte vierte Spalte von A

`A(3,:)` Zeile 3, alle Spalten; die gesamte dritte Zeile von A

`x(3:7)` verwendet den Vektor `3:7 = [3, 4, 5, 6, 7]` als Zugriffs-Index; greift den Teil-Vektor bestehend aus den x-Komponenten 3 bis 7 heraus

`A(2:4,3:5)` die Untermatrix bestehend aus den Zeilen 2 bis 4 und Spalten 3 bis 5 von A

Achtung, verwechseln Sie nicht `x(3:7)`, `x(3,7)` und `x([3,7])` ! Der erster Ausdruck ist ein Teilvektor von `x` (siehe oben); der zweite Ausdruck bezeichnet das Matrixelement x_{37}; der dritte verwendet den Vektor `[3, 7]` als Zugriffs-Index, greift also die 3- und 7- Komponente des Vektors x heraus. Das ergibt einen Vektor der Länge 2 mit den Komponenten `[x(3), x(7)]`.

Ü 2.4 Matrizen und Abbildungen

Die Matrix ist dazu gedacht,
dass sie aus einem Vektor einen and'ren macht.

Die linearen Abbildungen zwischen Vektorräumen $\mathbb{R}^m \to \mathbb{R}^n$ entsprechen genau den Matrix-Vektor-Multiplikationen $\mathbf{y} = A \cdot \mathbf{x}$ mit $n \times m$-Matrizen A.

Lineare Beziehungen zwischen Ein- und Ausgangsgrößen sind Grundbausteine für jede Art von Datenanalyse und -modellierung. Spannend und herausfordernd kann das für hochdimensionale Vektorräume werden. Im 2- und 3-dimensionalen Raum hingegen lassen sich Abbildungen noch gut visualisieren. Wenn Sie die Beispiele hier durchgearbeitet haben, kann die Verallgemeinerung auf mehrdimensionale Räume auch kein großer Schritt mehr sein. Laden Sie dazu die Datei `cat.dat` von der Übungs-Homepage in Ihr Arbeitsverzeichnis.

```
>> X=readmatrix('cat.dat');
>> size(X)
ans =
   119     2
>> X=X';
>> size(X)
ans =
     2   119
```

Sie lesen aus einer Textdatei einen Datensatz als Matrix ein. X hat 119 Zeilen und 2 Spalten. Wir vertauschen fürs weitere Arbeiten Zeilen und Spalten. X hat also nun 2 Zeilen und 119 Spalten.

> Für MATLAB-Versionen vor R2019a: Den Befehl `X=readmatrix('cat.dat')` gibt es noch nicht. Laden Sie die Skript-Datei datenX.m herunter und führen Sie diese aus. Danach ist die Matrix `X` im workspace geladen und Sie können mit den Befehlen `size(X)` und `X=X';` weiterarbeiten.

Die Spaltenvektoren in `X` entsprechen 119 Punkten im 2-dimensionalen Raum. Zeichnen Sie diese Punkte, dann können Sie sich die Daten besser vorstellen.

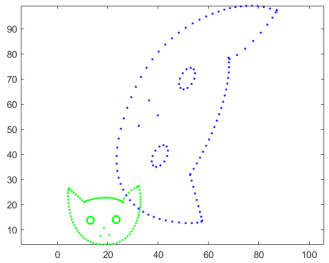
```
>> plot(X(1,:),X(2,:),'b.')
>> axis equal
```

Arnolds Katze[20] blickt Sie an. Gegeben sei nun die Matrix

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 0 \end{bmatrix} .$$

Wie wirkt diese Matrix auf Punkte des $\mathbb{R}^2$? Sie können die 2×2-Matrix A auf die 119 Spaltenvektoren, die in der Matrix X stehen, einfach durch Multiplikation anwenden: $Y = A \cdot X$ ist eine 2×119-Matrix, deren Spaltenvektoren jeweils die mit A multiplizierten Spaltenvektoren von X sind. Lassen Sie sich Ausgangs- und Bildpunkte zeichnen:

```
A=[ 1 2; 3 0];
Y=A*X;
plot(X(1,:),X(2,:),'g.',Y(1,:),Y(2,:),'b.')
axis equal
```



Sie sehen: A verdreht, vergrößert und verzerrt die Katze. Sie transformiert Ausgangsvektoren in Bildvektoren, die (außer in Sonderfällen) andere Länge und Richtung haben. (Spätestens jetzt sollten Sie das Merksprüchlein vom Anfang des Abschnittes verstehen!)

In diesem Beispiel werden alle Vektoren mehr oder weniger verlängert, aber nicht beliebig groß. Der linke Ohrzipfel, Punkt $\approx$ [32; 27], wird auf [86; 96] abgebildet; das entspricht einer Verlängerung um den Faktor $\approx 3{,}1$.

[20] Der Mathematiker Wladimir Igorewitsch Arnold (1937–2010) illustrierte damit in Lehrbüchern der klassischen Mechanik die Eigenschaften von Transformationen.

```
>> x=[32; 27];
>> y=A*x
y =
    86
    96
>> norm(y)/norm(x)
ans =
    3.0784
```

So können Sie Verlängerungs-Faktoren berechnen (in der 2-Norm).

Versuchen Sie andere Werte für x. Für welchen Vektor finden Sie den größten Verlängerungsfaktor? Umgekehrt: finden Sie auch einen Vektor mit besonders kleinem Verlängerungsfaktor?

Die Norm einer Matrix liefert den maximalen Verlängerungsfaktor

(Das gilt für 1-, 2- oder ∞-Norm; je nachdem, in welcher Norm gemessen wird, können die Werte leicht unterschiedlich sein. Für die Matrix A aus diesem Beispiel ist $\|A\|_2 = 3{,}5266$. Weil sich dieser nicht so einfach aus den Matrixelementen berechnen lässt, arbeitet man auch mit der 1-Norm (maximale Spaltenbetragssumme: 4, der Vektor [1; 0] verlängert sich um diesen Wert in der 1-Norm) oder der ∞-Norm (maximale Zeilenbetragssumme: 3, der Vektor [1;1] verlängert sich um diesen Wert in der ∞-Norm)).

```
>> [norm(A) norm(A,1) norm(A,'inf')]
ans =
    3.2566    4.0000    3.0000
```

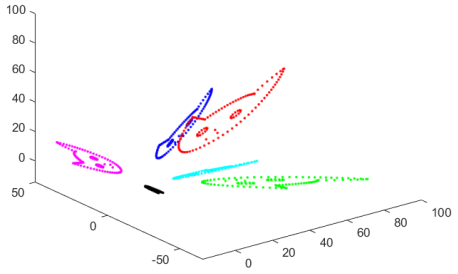
Das sind drei Matrix-Normen von A

Matrizen können aber auch lineare Abbildungen zwischen Vektorräumen unterschiedlicher Dimension vermitteln. Gegeben seien nun die Matrizen

$$B = \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \end{bmatrix} \quad , \quad C = \frac{1}{15} \begin{bmatrix} -2 & 2 & 14 \\ -2 & -13 & -16 \\ 14 & 16 & 7 \end{bmatrix} .$$

Wenden Sie auf die Daten-Matrix X zuerst B und dann einige Male C an und sehen Sie Arnolds Katze durch den Raum wandern...

```
Z=B*X;
plot3(Z(1,:),Z(2,:),Z(3,:),'b.')
hold on
Z=C*Z;
plot3(Z(1,:),Z(2,:),Z(3,:),'r.')
Z=C*Z;
...
plot3(Z(1,:),Z(2,:),Z(3,:),'k.')
hold off
```



Aufgabe 11:

Gegeben sind die 2×119 Datenmatrix X wie oben und Matrizen

$$D = \begin{bmatrix} 1 & 1 \\ -\frac{1}{4} & \frac{3}{4} \end{bmatrix} \quad , \quad E = \begin{bmatrix} 1 & 2 \\ -\frac{1}{4} & \frac{3}{4} \end{bmatrix} \quad , \quad F = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ -1 & \frac{1}{2} \end{bmatrix} \quad , \quad G = \begin{bmatrix} \frac{1}{2} & 0 \\ -\frac{1}{2} & \frac{1}{2} \end{bmatrix}$$

Wenden Sie die entsprechenden Abbildungen jeweils mehrere Male auf die Datenmatrix an und zeichnen Sie die Bildpunkte. Für welche Matrizen

- konvergiert die Fixpunkt-Iteration?

- liegt eine kontrahierende Abbildung vor?
  Eine kontrahierende Abbildung garantiert automatisch Konvergenz der Fixpunkt-Iteration. Aber auch für manche nicht-kontrahierende Abbildungen kann Fixpunktiteration konvergieren.
- liegt eine symplektische Abbildung vor?
  Das sind Abbildungen, die zwar Flächen verzerren, aber den Flächeninhalt unverändert lassen. Das können Sie aus den graphischen Darstellungen natürlich nur annähernd schätzen. Vielleicht finden Sie aber auch heraus: neben der Norm, die den Verlängerungsfaktor misst, gibt es eine weitere wichtige Matrix-Größe, die den Flächen- (oder allgemeiner: Volums-) Vergrößerungs-Faktor determiniert.

  Symplektische Abbildungen sind ein wichtiges Thema in der theoretischen Mechanik und für österreichische Radrennfahrerinnen[21].

Ü 2.5 Funktionen

Sie können in MATLAB eigene Funktionen definieren und in sogenannten Funktions-M-Dateien (*function M-files*) speichern. Bei einfache Funktionstermen ist es auch möglich, Funktions-Einzeiler, sogenannte *anonymous functions* zu verwenden.

Dieser Abschnitt erklärt folgende Punkte:

- Eine einfache Funktionsdatei
- Nullstellen mit `fzero` und `roots`
- Funktions-Einzeiler (*anonymous functions*)

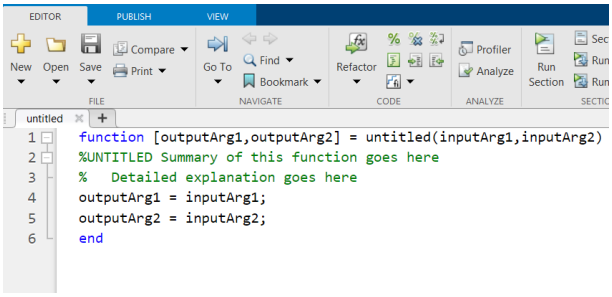
Ü 2.5.1 Eine einfache Funktionsdatei

Gegeben sei die Funktion

$$f : x \mapsto y = 3\cos x - \log x \ .$$

Sie ist ihnen in der Vorlesung und im Skriptum bereits begegnet.

Wählen Sie im MATLAB-Fenster links oben „New–Function“. Sie öffnen damit ein Fenster des MATLAB-Editors, in dem auch schon eine Muster-Funktion drinsteht.



Ergänzen Sie und erstellen Sie einen Quelltext. Orientieren Sie sich am folgenden Muster:

```
function [ y ] = meinf(x)
%MEINF Mein erster Versuch, eine Funktion zu programmieren
%   Funktion aus dem Skript, Abbildung 2
%
% Uebungen NM1, C.B. Feb.2014
y=3*cos(x)-log(x);
end
```

Die erste Zeile einer Funktionsdatei legt die Ein- und Ausgabeparameter sowie den Namen

[21] Anna Kiesenhofer, Integrable systems on b-symplectic manifolds. (Thesis, 2016)

einer Funktion fest. In diesem Fall heißt die Funktion `meinf`, hat eine Variable x als Argument (Input) und liefert einen Funktionswert y (Output). Die Argumente x und y können Skalare, aber auch Felder (Vektoren, Matrizen) sein. Sie können eigene Funktionsnamen verwenden, aber bitte nur zulässige Namen (keine Umlaute und Sonderzeichen, keine Ziffern zu Beginn)

Speichern Sie Ihre Funktion ab. Achten Sie darauf, dass die Datei unter dem Namen `meinf.m` gespeichert wird. Am besten legen Sie sich für die Matlab-Dateien, die Sie in diesen Übungen erstellen, ein eigenes Unterverzeichnis an.

Je nachdem, in welchem Verzeichnis sie gespeichert haben, kann für Sie eine kleine Zusatzarbeit anfallen:

> Skript- und Funktionsdateien lassen sich im *command window* nur aufrufen, wenn sie sich im aktuellen Ordner befinden.
> Wechseln Sie in der MATLAB-Oberfläche, linkes Fenster „Current Folder“ in das Verzeichnis, in dem Sie Ihre Funktion abgespeichert haben

Tipp: Schneller geht es, wenn Sie im Editor-Fenster, nachdem Sie die Funktion gespeichert haben, auf das grüne „Run“-Pfeil-Symbol klicken. Wenn eine Dialog-Box erscheint mit einem Text der Art *File D:/work/meinf.m is not found in the current folder blabla blabla*, dann klicken Sie einfach auf „Change Folder“ und ignorieren Sie weitere Meldungen (*mein.f requires more input...*) wurscht–das Verzeichnis ist gewechselt, mehr wollten wir nicht.

> Die Option *add to path* ist nicht zu empfehlen. Bei Namensgleichheit von Dateien verlieren Sie sonst die Übersicht, welche Datei nun tatsächlich ausgeführt wird.

Wechseln Sie wieder in die MATLAB *Command Window*. Stellen Sie zuerst gleich einmal die Anzeige von mehr Dezimalstellen ein und zusätzliche Leerzeilen ab:

```
>> format long g
>> format compact
>>
```

Andere Möglichkeiten:
short g gibt weniger Stellen.
short oder long e gibt Exponentialschreibweise.
format loose schiebt Leerzeilen ein.

Testen Sie nun, ob sich die Funktion aufrufen lässt:

```
>> y=meinf(1)
y =
   1.62090691760442
```

Sie haben die Funktion an der Stelle 1 ausgewertet.

Wenn Sie brav Kommentare eingegeben haben, gibt es nun eine Belohnung: Geben Sie ein:

```
>> help meinf
 meinf Mein erster Versuch, eine Funktion zu programmieren
    Funktion aus dem Skript, Abbildung 2

  Uebungen NM1, C.B Feb.2014
```

Und noch etwas ist toll an dieser Funktion: sie liefert nicht nur für einen skalaren Wert ein Ergebnis, sondern automatisch auch für eine ganze Werteliste. Erinnern Sie sich: Die Anweisung *Startwert*:*Schrittweite*:*Endwert* ist (alternativ zu `linspace`) eine zweite Möglichkeit, Zeilenvektoren zu erzeugen.

```
>> x=0:1:10

x =

     0     1     2     3
4     5     6     7     8
9    10
```

Werte von 0 bis zehn. Auch x=0:10 hätte funktioniert (Wenn Schrittweite fehlt, nimmt MATLAB automatisch 1)

Die Funktion `meinf` lässt sich elementweise auf den Zeilenvektor x anwenden.

```
>> meinf(x)
ans =   Inf    1.62090691760442
-1.94158769020137 ...
```

Kleiner Schönheitsfehler: $f(x) = 3\cos x - \log x$ ist für $x = 0$ nicht definiert. Der Funktionswert $f(0)$ wird als `Inf` gespeichert.

Ü 2.5.2 Nullstellen mit dem Newton-Verfahren

Dazu müssen Sie auch die Ableitung der Funktion f als Funktions-Datei programmieren. Ausnahmsweise helfen Ihnen die Unterlagen hier noch beim Differenzieren:

$$f'(x) = -3\sin x - \frac{1}{x}$$

```
function y = meindf(x)
%MEINDF Ableitung der Funktion meinf
% es ist so fad, Kommentare
% zu schreiben, aber es zahlt
% sich aus, wenn man eine Woche
% spaeter versucht, zu verstehen,
% was das hier sein soll
y=-3*sin(x)-1./x;
end
```

Öffnen Sie ein neues M-file, programmieren Sie die Ableitung, und speichern Sie unter dem Namen `meindf.m` ab.

```
>> meindf(1)
ans =
  -3.52441295442369
>> x=1:3
x =
     1     2     3
>> meindf(x)
ans =
  -3.52441295442369
-3.22789228047704
-0.75669335751293
```

Prüfen Sie, ob sich die Ableitung korrekt auswerten lässt

Ein kleines, aber wichtiges Detail: Wenn Sie die Division als `1/x` programmieren, funktioniert die Auswertung für *skalare* x problemlos, nicht aber, wenn x ein Vektor ist! Anfangs ist es oft schwer zu durchblicken, wann MATLAB Multiplikationen und Divisionen von sich aus elementweise interpretiert, und wann man explizit den Punkt setzen muss.

> **Regel:** (War schon in der vorigen Einheit da[22])
> In Funktionsdateien immer die elementweisen Operatoren `.* ./ .^` verwenden.

[22] *repetitio est mater studiorum*, auf Deutsch „Wiederholung ist die Mutter der Langeweile“, oder so ähnlich...

Dann hätten wir allerdings oben gemäß der Regel auch die erste Multiplikation mit `.*` schreiben sollen:

```
y=-3.*sin(x)-1./x;
```

In diesem Fall ist es wurscht, ob man `.*` oder `*` schreibt: wenn einer der beiden Operanden ein Skalar ist, wird automatisch elementweise gerechnet.

Zurück zum Newton-Verfahren. Informieren Sie sich notfalls im Skriptum, Kapitel 1.10, über die Rechenvorschrift.

```
>> x=1
x =
     1
>> x=x-meinf(x)/meindf(x)
x =
   1.45990834177644
```

Beginnen Sie mit Startwert 1 und rechnen Sie einen Schritt des Newton-Verfahrens.

Wiederholen Sie den Newton-Schritt, bis sich der Wert der Nullstelle nicht mehr ändert!

Vergleichen Sie ihre Werte mit folgender Tabelle, und beachten Sie das Konvergenzverhalten:

	korrekte Stellen:
1	1
1.45990834177644	1.4
1.44725583798192	1.44725
1.44725861727779	1.447258617277
1.44725861727790	alle

Die Anzahl der richtigen Stellen ist in jedem Schritt mindestens doppelt so groß wie im vorhergehenden. Diese außerordentlich rasche Konvergenz (quadratische Konvergenz) ist charakteristisch für das Newton-Verfahren.

Ü 2.5.3 Nullstellen mit der Sekantenmethode

```
>> xalt = 2
xalt =
     2
>> x = 1
x =
     1
>> xneu = x - meinf(x)*(x-xalt)/(meinf(x)-meinf(xalt))
xneu =
   1.45499210414322
```

Wenn Sie nicht wissen, was das hier soll, ziehen Sie das Skriptum, Kapitel 1.9 oder Vorlesungsunterlagen zu Rate.

```
>> xalt = x; x = xneu;
```

Mit diesen Befehlen setzen Sie die beiden Startwerte der Sekantenmethode neu auf die beiden zuletzt berechneten Werte. Jetzt können Sie die Formel erneut auswerten (aber nicht neu eintippen, Pfeiltaste verwenden!)

```
>> xneu = x - meinf(x)*(x-xalt)/(meinf(x)-meinf(xalt))
xneu =
   1.44716725175157
```

Wiederholen Sie die Auswertung der Sekantenmethode, bis sich der Wert der Nullstelle nicht mehr ändert!

Vergleichen Sie ihre Werte mit folgender Tabelle, und beachten Sie das Konvergenzverhalten:

	korrekte Stellen:
1	1
1.45499210414322	1.4
1.44716725175157	1.447
1.44725862822699	1.4472586
1.44725861727792	1.4472586172779
1.44725861727790	alle

Die Anzahl richtiger Stellen, sagt die Theorie, ist die Summe der richtigen Stellen der beiden vorherigen Näherungen. Das würde typischer Weise 0, 1, 1, 2, 3, 5, 8, 13, ... genaue Stellen bedeuten (kommt ihnen diese Folge bekannt vor?) Dem entsprechend sollte sich pro Schritt die Anzahl der richtigen Stellen um etwa 60% erhöhen.

Tatsächlich verdoppelt sich die Anzahl bei den ersten Schritten (1-2-4-8), und erhöht sich schliesslich immer noch um 75% (von 8 auf 14 Stellen). Die Sekantenmethode konvergiert hier schneller, als sie es den Regeln der Theorie entsprechend müsste.

Ü 2.5.4 Nullstellen mit fzero

MATLAB hat *numerische* Verfahren zur Nullstellensuche eingebaut, eine Kombination aus Intervallhalbierung, Sekantenmethode und inverser quadratischer Interpolation. Die Funktion `fzero` ruft diese Verfahren auf.

Für unser Beispiel $f(x) = 3\cos(x) - \log(x)$ lautet der Aufruf:

```
>> fzero(@meinf,1)
ans =
   1.44725861727790
>>
```

fzero steht für „find zero“

Wichtig: Dem Funktionsnamen (hier: `meinf`) müssen Sie den „Funktionshenkel“ `@` voranstellen (MATLAB nennt `@` *function handle*). Funktionsnamen mit Henkel `@` davor sind ein eigener Datentyp, damit lassen sich Funktionen an andere Funktionen übergeben. Stellen Sie sich vor, `@` sei der Henkel, mit dem Sie ein Kaffeehäferl weitergeben.

Aber `fzero` hat seine Tücken: es findet einen Punkt so nah wie möglich bei einem Vorzeichenwechsel der Funktion. Für stetige Funktionen (Zwischenwertsatz!) ist das zugleich ein Wert nahe einer Nullstelle. Für unstetige Funktionen kann `fzero` Werte liefern, die zu Singularitäten der Funktion gehören. Beispiel: der Tangens bei $\pi/2$

```
>> fzero(@tan,1)
ans =
   1.57079632679490
>>
```

Das ist keine Nullstelle der Tangensfunktion

Mehrfache Nullstellen (gerader Ordnung), bei denen die Funktion die x-Achse berührt, aber nicht schneidet, kann `fzero` auch nicht finden.

Funktions-Henkel können noch mehr. Damit lässt sich ohne Skript, als Einzeiler, eine Funktion definieren. Das erklärt der nächste Abschnitt.

Ü 2.5.5 Funktions-Einzeiler, *Anonymous Functions*

Bei einem einzeiligen Funktionsterm ist es nicht nötig, eigene Funktionsdateien zu schreiben. Unsere Beispiel-Funktion f mit Funktionsterm $f(x) = 3\cos(x) - \log(x)$ und ihre Ableitung $f'(x) = -3\sin x - \frac{1}{x}$ lassen sich auch als sogenannte *anonymous functions* definieren. Das sind Funktionen, die nicht in einer Programmdatei gespeichert sind, sondern mit einer Variablen vom Typ *function handle* (erkennbar am „Funktionshenkel“ `@` mit Funktionsargument in Klammer) verknüpft sind. Für unsere Beispiele:

```
>> f = @(x) 3*cos(x) -log(x)
f =
  function_handle with value:
    @(x)3*cos(x)-log(x)

>> df = @(x) -3*sin(x)-1./x
df =
  function_handle with value:
    @(x)-3*sin(x)-1./x
```

Sie können diese Einzeiler-Funktionen auswerten

```
>> f(1)
ans =
    1.6209
```

oder direkt `fzero` anwenden. Unbedingt ausprobieren! Achtung, weil `f` bereits vom Typ „Funktionshenkel“ ist, ist beim Argument von `fzero` *kein* Henkel `@` mehr erlaubt.

```
>> fzero(f,1)
ans =
    1.4473
>> fzero(f,10)
ans =
   11.9702
```

Die Schritte eines Newton-Verfahrens im *command window* könnten dann beispielsweise so aussehen:

```
>> x=1;
>> x=x-f(x)/df(x)
x =
   1.459908341776445
>> x=x-f(x)/df(x)
x =
   1.447255837981919
>> x=x-f(x)/df(x)
x =
   1.447258617277790
>> x=x-f(x)/df(x)
x =
   1.447258617277903
```

> Funktions-Einzeiler (*anonymous functions*) sind praktisch, wenn der Funktionsterm aus einem einzigen ausführbaren Befehl besteht.
> Die Bezeichnung eines Funktions-Einzeiler ist eine Variable vom Typ „Funktions-Henkel“, sie bezieht sich *nicht* auf eine Funktionsdatei

Ü 2.5.6 Nullstellen mit roots

Lösungen (man sagt auch Wurzeln) *polynomialer* Gleichungen sind für MATLAB leichter zu finden. Betrachten wir als Beispiel

$$p(x) = x^3 - 2x - 5$$

Ein Polynom ist durch die Angabe seiner Koeffizienten bestimmt. In diesem Beispiel lauten sie

$$1; 0; -2; -5$$

weil $p(x) = 1 \cdot x^3 + 0 \cdot x^2 - 2 \cdot x - 5 \cdot x^0$. Sie übergeben `roots` die Liste der Koeffizienten als Vektor `[1 0 -2 -5]`, und roots liefert *alle* (auch die komplexen) Nullstellen des Polynoms.

```
>> roots([1 0 -2 -5])
ans =
   2.09455148154233
  -1.04727574077116 + 1.13593988908893i
  -1.04727574077116 - 1.13593988908893i
>>
```

Übungsbeispiele

Aufgabe 12:

Wie groß ist das Molvolumen von Stickstoff bei 20 C und 1 bar nach der Van der Waals-Gleichung?

Die Zustandsgleichung

$$\left(p + \frac{a}{V_{mol}^2}\right)(V_{mol} - b) = RT$$

beschreibt den Zusammenhang zwischen Druck p, Molvolumen V_{mol} und Temperatur T. Die Konstanten a und b haben für Stickstoff die Werte

$$a = 0{,}129\,\mathrm{Pa\,m^6/mol^2}, \quad b = 38{,}6 \times 10^{-6}\,\mathrm{m^3/mol}.$$

Die molare Gaskonstante ist $R = 8{,}3145\,\mathrm{J/molK}$. Nach Einsetzen der Zahlenwerte verbleibt als Gleichung für V_{mol}:

$$\left(100000 + \frac{0,129}{V_{mol}^2}\right)(V_{mol} - 0,0000386) = 2437,4$$

Lösen Sie diese Aufgabe mit der Sekanten- und der Newtonmethode und mittels `fzero`. Die Gleichung lässt sich auch auf polynomiale Form umformen. Verwenden Sie `roots` zur Lösung. Das Skriptum beschreibt in Kapitel 1.6 eine Umformung als Fixpunkt-Gleichung. Lösen Sie die Aufgabe auch mit Fixpunkt-Iteration!

Aufgabe 13:

Gegeben sei das Polynom

$$p(x) = -64 + 176\,x - 188\,x^2 + 101\,x^3 - 29\,x^4 + \frac{17\,x^5}{4} - \frac{x^6}{4}$$

Schreiben Sie dazu eine Funktions-M-Datei. Denken Sie dabei daran, die elementweisen Operatoren `.^` zu verwenden. Vergessen Sie nicht den Strichpunkt am Ende der Zeile (sonst liefert das M-file mächtig viel unnötige Ausgabezeilen im *Command Window*).

Zeichnen Sie das Polynom für $0 < x < 5$. Suchen Sie Nullstellen mit dem Newton-Verfahren, mit `fzero` (Sie müssen dazu geeignete Startwerte setzen) und mit `roots` .

Warum findet `fzero` nicht alle Nullstellen?

Überprüfen Sie, wie vorher in den Übungsunterlagen, die Konvergenzgeschwindigkeit des Newton-Verfahrens für alle drei Nullstellen. Verhält sich die Konvergenzgeschwindigkeit den Regeln entsprechend?

Ändern Sie im Polynom den Koeffizienten von x^3 um 1%; 0,1%; 0,01%. Bestimmen Sie Nullstellen mit `roots` und geben sie für jede ursprüngliche reelle Nullstelle an: Gibt es diese reelle Nullstelle noch? Wenn Ja, um wieviel hat sich ihr Wert relativ geändert? Oder hat sich eine mehrfache Nullstelle in mehrere verschiedene reelle Nullstellen aufgesplittet?

Diese Aufgabe illustriert: Die numerische Berechnung mehrfacher Nullstellen ist anfällig gegenüber kleinen Änderungen des Polynoms und Rundungsfehlern. Nullstellen verschwinden, verschieben oder vermehren sich. Man spricht von einem *schlecht konditionierten Problem*.

> **Schlecht konditioniertes Problem:** Kleine Änderungen in den Daten und/oder Rundungsfehler bewirken starke Änderungen im Ergebnis.

Ü 2.6 Kontrollstrukturen

Sie haben in den Aufgaben zur vorigen Übungseinheit Fixpunkt-Iterationen oder Iterationen des Newtonverfahrens oder der Sekantenmethode gleichsam „im Handbetrieb“ im *Command Window* eingegeben und, wenn die Ergebnisse sich nicht mehr geändert haben, das Verfahren beendet. Sie können diesen Ablauf auch als Programm formulieren. MATLAB bietet die üblichen Kontrollstrukturen (ähnlich wie in Java oder C++). Für den Anfang reichen `for`-Schleifen und `if`-Verzweigungen, wie sie dieser Abschnitt vorstellt. Mehr dazu, auch über `while`-Schleifen, erzählt Ihnen die MATLAB-Hilfe.

Schleifen

Eine for-Schleife hat zumeist die Form

```
for index = startwert:endwert
    anweisung
    anweisung
    ...
end
```

Der Index durchläuft dann die Werte `startwert, startwert+1, startwert+2...endwert`; er muss nicht (wie in Java oder C++) durch `index++` erhöht werden. Allgemeinere Form des Schleifenkopfes:

```
for index = startwert:schrittweite:endwert
```

Beispiel: ein Schleifenzähler s, der mit Schrittweite $-0,1$ die Werte $1; 0,9; 0,8; \ldots 0$ durchläuft:

```
for s = 1:-0.1:0
```

Verzweigungen

Eine bedingte Verzweigung mit `if` hat die Form

```
if ausdruck
    anweisung
    anweisung
    ...
end
```

MATLAB wertet „ausdruck“ aus, und wenn das Resultat logisch `true` oder (Unterschied zu Java!) ungleich 0 ist, führt es die folgenden Anweisungen bis zum `end` aus. Geschachtelte `if` sind möglich, jede Ebene muss mit dem entsprechenden `end` abgeschlossen werden.

Während Java sehr streng darauf achtet, dass in einer `if`-Anweisung nur ein logischer Ausdruck stehen darf, erlaubt MATLAB sogar Vektoren oder Matrizen. Wenn „ausdruck“ kein Skalar ist, muss jede einzelne Komponente `true` oder $\neq 0$ sein.

Die allgemeinere Form mit `elseif` und/oder `else` hat die Form

```
if ausdruck1
    anweisungen1
elseif ausdruck2
    anweisungen2
else
    anweisungen3
end
```

Musterprogramm

Das folgende Musterprogramm zur Fixpunkt-Iteration stellt Schleifen und Verzweigungen vor. Es führt eine Fixpunktiteration gemäß der Vorschrift

$$x^{(k+1)} = \phi(x^{(k)})$$

für den Startwert $x^{(0)}$ durch. Das Programm können Sie von der Übungs-Homepage herunterladen:

```
function x = fixpunkt(phi,x0)
%FIXPUNKT: Demo-Programm zur Fixpunkt-Iteration
% Das Verfahren iteriert gemaess der Vorschrift
%       x_i+1 = phi(x_i)
% bis Aenderungen unter eine Toleranzschwelle sinken
% oder maximale Iterationszahl ueberschritten wird.
%
% Eingabe   phi... Funktion, deren Fixpunkt gesucht ist
%                  (mit Funktions-Henkel @)
%           x0 ... Startwert oder -vektor
% Ausgabe   x .... Bei Konvergenz: Fixpunkt,
%                  sonst: NaN
% Beispiel  fixpunkt(@cos,1)
%
%-----------------------------------------NMI,SS09-18 C.Brand
itmax = 100;                % maximale Iterationszahl
errlim = 1.e-9;             % Fehlerschranke
for i=1:itmax
    x = phi(x0);            % Funktionsauswertung
    if norm(x-x0)<errlim    % Abbruchkriterium: 2-Norm des Fehlers
        return
    end
    x0 = x;
end
x = NaN;
end
```

Ü 2.7 Fixpunkt-Iteration ein- und mehrdimensional

Eindimensional

In Aufgabe 5 haben Sie die Qudratwurzel aus a als Fixpunkt der Funktion

$$y = \frac{1}{2}\left(x + \frac{a}{x}\right)$$

berechnet. Wir wiederholen, verwenden Schleifen und das Fixpunkt-Musterprogramm und verallgemeinern auf den mehrdimensionalen Fall.

```
>> heron = @(x)(x+13/x)/2;
```

Definieren Sie die Iterations-Funktion als *anonymous function*. Hier soll $\sqrt{13}$ berechnet werden. Alternativ können Sie auch eine Funktions-Datei dafür schreiben.

```
>> heron(2)
ans =
    4.2500
```

Testen Sie unbedingt, ob sich die Funktion aufrufen lässt und ein passendes Ergebnis liefert.

```
>> x=1;
>> x=heron(x)
x =
     7
```

So sieht eine Fixpunkt-Iteration im „Handbetrieb“ aus: wiederholter Aufruf der Funktion in der Befehlszeile.

```
>> x=heron(x)
x =
    4.4286
>> x=heron(x)
x =
    3.6820
>> x=heron(x)
x =
    3.6063
```

```
>> fixpunkt(heron,1)
ans =
    3.6056
```

Und so berechnet das Fixpunkt-Musterprogramm dasselbe Ergebnis.

Mehrdimensionale Fixpunkt-Iteration, vektorwertige Funktionen

Dieses Programm kann aber ebenso mehrdimensionale Fixpunkt-Iteration durchführen. In der Vorlesung wurde der Fixpunkt einer vektorwertigen Funktion $\mathbf{\Phi} : \mathbb{R}^2 \to \mathbb{R}^2$ als Beispiel behandelt:

$$\begin{aligned} x_1 &= \frac{1}{4}(x_2 - x_1 x_2 + 1) \\ x_2 &= \frac{1}{6}(x_1 - \log(x_1 x_2) + 2) \end{aligned}$$

Die Funktion $\mathbf{\Phi}(\mathbf{x})$ kann als Funktions-Datei so implementiert werden:

```
function y = phi(x)
x1=x(1);
x2=x(2);
y = [ (x2 - x1.*x2 + 1)/4
      (x1-log(x1.*x2)+2)/6 ];
end
```

(Bemerkung: Eine einzeilige *anonymous function* wäre schon auch möglich, aber bei einem etwas umfangreicheren Funktionsterm ist die Funktions-Datei die übersichtlichere Implementierung!)

Aufruf der Fixpunkt-Funktion für die Funktionsdatei (Henkel `@`+Dateiname!)

```
>> fixpunkt(@phi,[1;1])

ans =

    0.3534
    0.6400
```

Ü 2.8 Weitere Aufgaben bis zur nächsten Übungseinheit

Aufgabe 14:

Formulieren Sie die Iterationsvorschrift des Newton-Verfahrens zur Lösung von

$$\sin x = x/2 \ .$$

Programmieren Sie die entsprechende Iterations-Funktion und rufen Sie das Fixpunkt-Musterprogramm auf. Finden Sie damit alle positiven Lösungen der obigen Gleichung.

Aufgabe 15:

Suchen Sie die Nullstellen der Funktion

$$f(x) = e^x - 3x^2$$

mit Hilfe des Newton-Verfahrens und einfacher Fixpunkt-Iteration (mehrere Umformungen möglich).

Vergleichen Sie (bei denselben Startwerten) die Anzahl der Iterationen. Schätzen Sie bei den Verfahren mit linearer Konvergenz den Reduktionsfaktor C. Findet Ihre Fixpunkt-Formulierung alle Nullstellen?

Aufgabe 16:

Das Vorlesungsskriptum diskutiert die Gleichung

$$r = K\frac{q-1}{1-q^{-n}} \quad \text{für } r = 900, K = 100\,000, n = 180.$$

Schreiben Sie ein MATLAB-Programm, das allgemein für Eingabewerte r, K und n die Lösung q findet. (Annahme: $n \cdot r > K$ und daher $q > 1$) Welches Verfahren Sie verwenden, bleibt Ihnen überlassen.

Hinweise dazu:

- Der Aufzinsungsfaktor q liegt realistischer Weise im Bereich $1 < q < 1.05$, entsprechend einer monatlichen Verzinsung über 0% und unter 5%. Für $q = 1$ ist die Gleichung nicht definiert (Nenner wird 0), ein Startwert $q = 1$ ist nicht sinnvoll.
- Die Fixpunkt-Form (Auflösen nach q im Zähler) konvergiert langsam, ist aber rasch implementiert und vergleichsweise unempfindlich bezüglich der Wahl des Startwertes (sofern $q > 1$).

- Newton-Verfahren und MATLABs `fzero` brauchen gute Startwerte.

Aufgabe 17:

Formulieren Sie analog zum Fixpunkt-Musterprogramm eine Funktions-M-Datei zur Intervallhalbierung. Der Aufruf

```
IntervHalb(@igendeineFunktion, x0, x1)
```

soll, ausgehend von den Startwerten $x^{(0)}$ und $x^{(1)}$ eine Nullstelle der Funktion finden. Testen Sie das Verfahren, indem Sie Nullstellen folgender Funktion suchen:

$$f(x) = x^2 - 3\tan x + 1$$

Anfangsintervalle mit Vorzeichenwechsel sind: $[0, 1]$, $[1, 2]$, $[4.5, 4.7]$. Findet Intervallhalbierung für alle drei Intervalle eine Nullstelle? Was findet `fzero`, wenn Sie obige Intervallgrenzen als Startwerte geben?

Aufgabe 18:

Die Vorlesungsfolien zur 1. Vorlesung beschreiben die Illinois-Variante der Regula Falsi. Dieses Verfahren zeigt im Regelfall die guten Konvergenzeigenschaften der Sekantenmethode und bietet gleichzeitig den sicheren Einschluss der Nullstelle im aktuellen Intervall. Wenn Sie dieses Verfahren implementieren, haben Sie ein sehr brauchbares Allzweck-Nullstellen-Programm.

Orientieren Sie sich am Fixpunkt-Musterprogramm und schreiben Sie eine Funktions-M-Datei zur Regula Falsi in der Illinois-Variante. Der Aufruf

```
Illinois(@igendeineFunktion, x0, x1)
```

soll, ausgehend von den Startwerten $x^{(0)}$ und $x^{(1)}$ eine Nullstelle der Funktion finden. Testen Sie das Verfahren, indem Sie alle positiven Lösungen der folgenden Gleichung suchen:

$$\sin x = x/2$$

Rechnen Sie zum Vergleich auch mit `fzero` die Lösung.

Aufgabe 19:

Gegeben sei das Gleichungssystem

$$\begin{array}{rcrcrcr} 8x_1 & + & x_2 & - & x_3 & = & 8 \;, \\ 2x_1 & + & x_2 & + & 9x_3 & = & 12 \;, \\ x_1 & - & 7x_2 & + & 2x_3 & = & -4 \;. \end{array}$$

Formulieren Sie ein Fixpunktverfahren. Überlegen Sie, aus welcher Gleichung sie x_1 ausdrücken sollen, aus welcher x_2 und x_3. Hinweis: Unbekannte möglichst aus jener Gleichung ausdrücken, in der sie den stärksten Einfluss (den größten Koeffizienten) haben. Testen Sie das Verfahren.

Aufgabe 20:

(Wenn Sie die Unterlagen durchgearbeitet haben, ist diese Aufgabe eigentlich schon gelöst)

Gegeben sei ein System von zwei Gleichungen in zwei Unbekannten:

$$\begin{aligned} f(x, y) = 4x - y + xy - 1 &= 0 \\ g(x, y) = -x + 6y + \log xy - 2 &= 0 \end{aligned}$$

Formulieren Sie dafür ein Fixpunktverfahren und testen Sie!

Ü 3 Dritte Übungseinheit

Inhalt der dritten Übungseinheit:

- Script- und Funktionsdateien, Live Scripts
- Rechenoperationen bei Eingabe von Matrizen und Vektoren
- Bergabstrich löst Gleichungssysteme
- Inverses Problem, Regularisierung
- Newton-Verfahren für Systeme
- Lokale Funktionen
- `fsolve`
- Isolinien- und Oberflächengrafiken

Ü 3.1 Skript- und Funktionsdateien: Zusammenfassung

Wir haben schon mit beiden gearbeitet: Siehe Abschnitte Ü 1.3 und Ü 2.5 . Wichtige Unterschiede bestehen in der Verwendung und dem Gültigkeitsbereich von Variablen. Hier eine kurze Wiederholung und Zusammenfassung.

- Skripts. Sie haben weder Eingabe-Argumente noch Rückgabewerte. Sie greifen auf Daten und Variable des aktuellen Workspace zu. Befehlszeilen in einer Skript-Datei wirken so, als ob sie direkt im *Command Window* eingegeben würden.

  Ein häufiger Fehler ist, dass das Skript Variable aus dem aktuellen Workspace verwendet, aber nicht selber definiert. Sie merken während der aktuellen MATLAB-Sitzung nichts. Erst beim nächsten Neustart meldet MATLAB: `Undefined function or variable`. Geben Sie deswegen ganz zu Beginn des Skripts den Befehl `clear variables`. Damit löschen Sie alle Variablen im Workspace und merken sofort, ob sie Definitionen im Skript vergessen haben. Siehe Abschnitt Ü 1.3.

- Funktions-Dateien. Sie übernehmen Eingabewerte (Argumente)und liefern Resultate. Innerhalb der Funktion deklarierte Variable gelten nur lokal, also innerhalb der Funktion.

- Lokale Funktionen: In einer Funktionsdatei können weitere Funktionen („Unter-Funktionen“) vorkommen. Die erste Funktion (die „Hauptfunktion“) lässt sich von der Befehlszeile im *Command Window* aus oder über Befehlszeilen in einem Skript starten. Alle weiteren Funktionen (Reihenfolge egal) sind nur lokal – also nur für andere Funktionen aus derselben Datei – verfügbar. Es kann ganz praktisch sein, ein Programm mit Unterfunktionen zu gliedern. Weitere Infos: MATLAB-Hilfe, Stichwort *“local functions”*

- Auch Skript-Dateien können (ab MATLAB R2016b) lokale Funktionen enthalten; die dürfen aber erst nach der letzten Zeile des Skript-Codes beginnen. Hilfe dazu suchen Sie unter *“Add Functions to Scripts”*

- Verschachtelte Funktionen (für Fortgeschrittene): das sind Funktionen, die innerhalb einer übergeordneten Funktion deklariert sind. Damit können Sie Zugriff auf Variable noch differenzierter steuern. Details finden Sie in der MATLAB-Hilfe, Stichwort *“nested functions”*.

- Funktions-Einzeiler (*anonymous functions*) sind praktisch, wenn der Funktionsterm aus einem einzigen ausführbaren Befehl besteht. Wurden in Abschnitt Ü 2.5.5 vorgestellt.

- Live-Scripts und -Funktionen (Für Fortgeschrittene): Diese Art von Programm-Dateien kann Quellcode und Output gemeinsam darstellen, Begleittext und Kommentare übersichtlich formatieren, Sie können Schalt- und Steuerflächen für interaktives Arbeiten einbauen und mehr. Solche Dateien haben Endung `.mlx` im Unterschied zu Endung `.m` bei „gewöhnlichen“ Skripts.

  Wenn Sie Ihre Dateien besonders ansprechend gestalten wollen, können Sie ja versuchsweise im Editor auf *Save/Save as/Save as type: Matlab Live Code files (*.mlx)* klicken und so Ihre Datei in ein Live-Script umwandeln. Sie finden mehr Informationen und Beispiele in der MATLAB-Hilfe, Stichwort *Live Scripts and Functions*. Eine nette Ausarbeitung[23] von Aufgabe 8 finden Sie hier (klick!)

Ü 3.2 Rechenausdrücke bei der Eingabe von Vektoren und Matrizen

Eine Erinnerung an die letzte Einheit: Bei der Eingabe von Vektoren und Matrizen wirken Komma, Strichpunkt, Leerzeichen und Zeilenumbruch als Trennzeichen.

Sie können nicht nur Zahlenwerte, sondern auch Rechenausdrücke angeben:

```
>> x=[1+2, 2+3 3+4]
x =
     3     5     7
```

Zeilenvektoren: Leerzeichen oder Komma trennt Komponenten. Hier (nicht zu empfehlen) einmal Komma, einmal Leerzeichen.

```
>> y=[1-2 3 + 4 5 +6 7+ 8]
y =
    -1     7     5     6    15
```

Vorsicht mit Leerzeichen bei + und −: kann als Vorzeichen, als Rechenoperation oder als Trennzeichen interpretiert werden.

> Typischer Fehler bei Rechenausdrücken in einem Vektor: Leerzeichen vor Operator wirkt ungewollt als Trennzeichen. Empfehlung: Bei Rechenoperationen entweder keine Leerzeichen oder beidseitig Leerzeichen um Operatoren.

Regeln bei Plus oder Minus in Termen:

- beidseitig kein Abstand: Rechenoperation wird ausgeführt (`+` und `-` wirken als *binäre Operatoren*)
- beidseitig Leerzeichen: ebenfalls als binäre Operation interpretiert.
- links Leerzeichen, rechts nicht: Leerzeichen links wirkt als Trennzeichen zwischen Matrix-Elementen, Operatoren `+` und `-` wirken als *unäre Operatoren* (als *Vorzeichen* des folgenden Terms).

  Passiert oft unabsichtlich, erzeugt Fehlermeldung oder falsches Ergebnis.
- rechts Leerzeichen, links nicht: binäre Operation (aber wenn Sie das absichtlich machen, ist das ziemlich verhaltensoriginell).

[23] von Chr. Tuschl, danke schön!

Ü 3.3 Gleichungssysteme: MATLABs schräge Schreibung

MATLAB verwendet – völlig normal – den Schrägstrich als Divisionsoperator: `x = 3/4` liefert wenig überraschend `x = 0.75000`.

Eher unüblich ist der andersrum gekippte „Bergabstrich“ `\` oder Backslash:

```
>> x=3\4
x =
    1.3333
```

Auch `\` dividiert, aber in der Form `x = Nenner\Zähler`. Es liegt ja auch wirklich, wenn Sie sich `\` als abwärts geneigten Bruchstrich vorstellen (oder den Bildschirm kurz mal nach links kippen), der linke Term *unter* und der rechte *über* dem Bruchstrich.

Grenzwertig originell ist jedoch MATLABs Interpretation des Bergabstrichs bei Gleichungssystemen. Das Wichtigste in Kürze:

`x = A\b` löst (oder „löst“) das Gleichungssystem $A\mathbf{x} = \mathbf{b}$

Nicht jedes von MATLAB so berechnete Ergebnis ist die Lösung eines Gleichungssystems im eigentlichen Sinn – deswegen die Anführungszeichen bei „löst“.

Niemand will sich im Detail merken, was genau MATLAB bei den verschiedenen Sonderfällen tut. Nur damit Sie sehen und gewarnt sind, was alles passieren kann, hier eine Aufzählung.

Der Befehl `x = A\b` liefert für ein Gleichungssystem $A\mathbf{x} = \mathbf{b}$

- bei nicht singulärer $n \times n$- Matrix die eindeutige Lösung;
- bei singulärer $n \times n$- Matrix eine Warnmeldung und, falls es Lösungen gibt, eine Lösung mit möglichst vielen Null-Komponenten. Falls es keine Lösung gibt, liefert MATLAB meistens unsinnige Zahlenwerte.
- bei einer $n \times m$- Matrix mit $n < m$ (*unterbestimmtes* Gleichungssystem), falls es Lösungen gibt, eine Lösung mit möglichst vielen Null-Komponenten.
- bei einer $n \times m$- Matrix mit $n > m$ (*überbestimmtes* Gleichungssystem), die „am wenigsten falsche Lösung“. Das ist jener Vektor $\mathbf{x}$, für den $A\mathbf{x} - \mathbf{b}$ die kleinste 2-Norm hat. Man spricht von der *Kleinste-Quadrate-Lösung*.
- Sonderfälle sind $n \times m$- Matrizen mit $n \neq m$ und Rang $< \min(m, n)$. Matlab warnt: „Warning: Rank deficient“ und liefert eine Kleinste-Quadrate-Lösung.

Gleichungssysteme mit mehreren rechten Seiten

Bei gleicher Matrix A und mehreren rechten Seiten $\mathbf{b}, \mathbf{c}, \mathbf{d}, \ldots$ lassen sich die Gleichungssysteme $A\mathbf{x} = \mathbf{b}, A\mathbf{y} = \mathbf{c}, A\mathbf{z} = \mathbf{d}, \ldots$ gemeinsam lösen. Stellen Sie alle rechten Seiten als Spaltenvektoren in einer Matrix B zusammen: `B = [b,c,d]`. MATLABs `\` liefert eine Matrix X, deren Spalten die Lösungsvektoren enthält: $X = [\mathbf{x}, \mathbf{y}, \mathbf{z}]$.

`X = A\B` löst (oder „löst“) die Gleichungssysteme $A \cdot X = B$

Schrägstriche zwischen Matrizen, allgemeiner Fall

Die Kurzfassung:

> Verwenden Sie den „Bergaufstrich“ `/` zwischen Skalaren als Divisionsoperator und den „Bergabstrich“ `\` zwischen Matrizen und Vektoren zum Lösen linearer Gleichungssysteme.

Wenn Sie mehr über MATLABs Umgang mit `/` und `\` wissen wollen, lesen Sie weiter...

Sie könnten $x = 3/4$ auch in der Form $x = 3 \cdot 4^{-1}$ schreiben, oder $x = 4^{-1} \cdot 3$. Niemand tut das bei skalaren Termen. Bei Matrizen ist die Schreibweise mit Inversen jedoch Standard, zum Beispiel $A \cdot B^{-1}$ oder $A^{-1} \cdot B$. MATLAB erlaubt dafür die Bruchstrich-Schreibung:

$$A \cdot B^{-1} = A/B \quad \text{und } A^{-1} \cdot B = A\backslash B$$

Stellen Sie sich `/B` als B^{-1} vor, weil B „unter“ dem Bruchstrich liegt, und entsprechend `A\` als A^{-1}. Die Schrägstriche stehen absichtlich *links* von B und *rechts* von A – Matrixmultiplikation ist nicht kommutativ! Je nachdem, von welcher Seite Sie mit einer Inversen multiplizieren wollen, verwenden Sie `/` oder `\`.

MATLAB berechnet nicht wirklich die inversen Matrizen und multipliziert damit. Das explizite Ausrechnen einer Inversen ist rechenaufwändig und mit Rundungsfehlern behaftet. In Wirklichkeit löst MATLAB mit der schrägen Bruchstrichschreibweise lineare Gleichungssysteme. Das ist nur bei nichtsingulären quadratischen Matrizen algebraisch äquivalent zur Multiplikation mit der Inversen.

Gleichungssysteme statt inverser Matrix

Wenn in mathematischen Ausdrücken eine Multiplikation mit der inversen Matrix auftritt, brauchen Sie für die rechnerische Auswertung die Inverse in expliziter Form nicht wirklich[24].

Für das numerische Rechnen besteht ein gewaltiger Unterschied im Hinblick auf Rechenaufwand und -genauigkeit, ob Sie eine Inverse berechnen und damit multiplizieren, oder so umformen, dass Sie Gleichungssysteme lösen.

> Vermeiden Sie explizite Multiplikation mit einer inversen Matrix! Es verursacht unnötigen Rechenaufwand und unerwünschte Rundungsfehler.
> `X = A\B` berechnet $A^{-1} \cdot B$ als Lösung des Gleichungssystems $A \cdot X = B$
> `X = A/B` berechnet $A \cdot B^{-1}$ als Lösung des Gleichungssystems $X \cdot B = A$

Zur Erklärung, warum sich Multiplikation mit Inverser auf Lösung eines Gleichungssystems zurückführen lässt, hier die entsprechenden Umformungen. Achtung, man muss „auf der richtigen Seite“ multiplizieren, damit die Inversen verschwinden: im ersten Fall multipliziert man A von links, das andere mal mit B von rechts. (Matrixmultiplikation ist nicht kommutativ!)

$$\begin{aligned} X &= A^{-1} \cdot B && | \; A\cdot \\ A \cdot X &= A \cdot A^{-1} \cdot B \\ A \cdot X &= B \end{aligned} \qquad\qquad \begin{aligned} X &= A \cdot B^{-1} && | \; \cdot B \\ X \cdot B &= A \cdot B^{-1} \cdot B \\ X \cdot B &= A \end{aligned}$$

[24] Damit Sie erst gar nicht in Versuchung kommen, eine Inverse zu berechnen, verschweigen diese Übungsunterlagen (vorerst einmal) absichtlich den MATLAB-Befehl zur Berechnung der Inversen.

Ü 3.4 Inverse Probleme, Regularisierung

Was eine Matrix tut,
macht die Inverse wieder gut.

Abschnitt Ü 2.4 hat erklärt: Die Matrix-Vektor-Multiplikation $\mathbf{y} = A \cdot \mathbf{x}$ definiert eine Transformation *Eingabedaten* $\rightarrow$ *Bilddaten*. Oft liegt das Problem in umgekehrter Form vor: Gegeben ist der Ergebnisvektor $\mathbf{y}$, welcher Vektor $\mathbf{x}$ führt zu diesem Ergebnis?

Wenn die inverse Matrix existiert, ist die theoretische Antwort einfach:

$$\mathbf{y} = A \cdot \mathbf{x} \qquad \Leftrightarrow \qquad \mathbf{x} = A^{-1} \cdot \mathbf{y} \quad ,$$

Es kann aber sein, dass mehrere $\mathbf{x}$-Vektoren denselben Ergebnisvektor $\mathbf{y}$ haben, oder es überhaupt keinen $\mathbf{x}$-Vektor gibt, der exakt zum Ergebnisvektor $\mathbf{y}$ führt. Dann wird es schwierig. Aber selbst wenn eine Inverse theoretisch existiert, kann es sein, dass die Rücktransformation damit praktisch unmöglich ist.

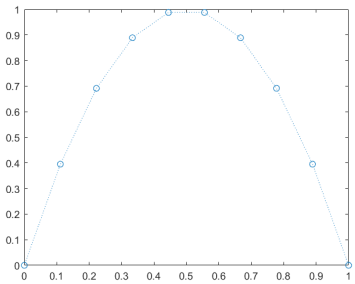
Inverses Problem: Ergebnis bekannt, Ursache gesucht. Oft schwierig.

Hier stellen wir Ihnen eine solche Situation vor, und zeigen, wie man damit umgehen kann.

Aufgabe 21: Schlecht konditioniertes inverses Problem

Erzeugen Sie eine Zeitreihe $\mathbf{x}(t)$ mit n Datenpunkten; hier mit parabelförmigen Verlauf; zeichnen Sie die Daten.

```
n=10;
t = linspace(0,1,n)';  % t Spaltenvektor!
x = 4*t.*(1-t);
plot(t,x,'o:')
```



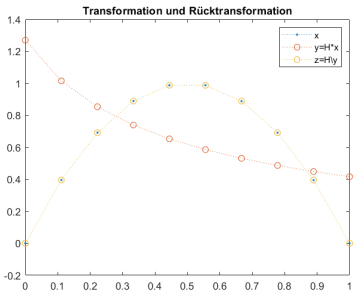
Wenden Sie auf $\mathbf{x}$ eine Transformation, die sogenannte Hilbert-Matrix, an: $\mathbf{y} = H \cdot \mathbf{x}$. Die Matrix ist einfach aufgebaut, ihre Inverse (sie enthält nur ganzzahlige Elemente) lässt sich explizit angeben. In MATLAB erzeuht sie der Befehl `hilb(n)` . Zum Beispiel für $n = 4$

$$H = \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \end{bmatrix} \qquad H^{-1} = \begin{bmatrix} 16 & -120 & 240 & -140 \\ -120 & 1200 & -2700 & 1680 \\ 240 & -2700 & 6480 & -4200 \\ -140 & 1680 & -4200 & 2800 \end{bmatrix}$$

Auf den Ergebnisvektor $\mathbf{y}$ wenden Sie gleich wieder die Rücktransformation an. Das heißt, Sie lösen das Gleichungssystem $H \cdot \mathbf{z} = \mathbf{y}$. Theoretisch[25] muss dann matürlich $\mathbf{z}$ ident mit dem Ausgangsvektor $\mathbf{x}$ sein. Zeichnen Sie daher $\mathbf{x}, \mathbf{y}$ und $\mathbf{z}$. In MATLAB führen Sie das so aus:

[25] Alte Lebensweisheit: *Theoretisch ist kein Unterschied zwischen Theorie und Praxis. Praktisch schon.*

```
H = hilb(n);
y = H*x;
z = H\y;
plot(t,x,'.:',t,y,'o:',t,z,'o:')
legend('x','y=H*x','z=H\y')
title('Transformation und Rücktransformation')
```



In der Abbildung hier, für $n = 10$ Datenpunkte, liegen die **x**- und **z**-Datenpunkte perfekt übereinander.

Wiederholen Sie die Aufgabe für größere n. Ab welchem n liefert die Rücktransformation deutlich unterschiedliche Werte, ab wann katastrophal unbrauchbare?

Die Hilbert-Matrix ist nur ein Beispiel für Transformationen, die auf extrem schlecht konditionierte inverse Probleme führen.

Sie sollen nun trotzdem für $n = 50$ die Rücktransformation durchführen. In solchen Fällen wenden Sie Regularisierungsmethoden an. Hier das Kochrezept für Tichonov-Regularisierung.

Statt

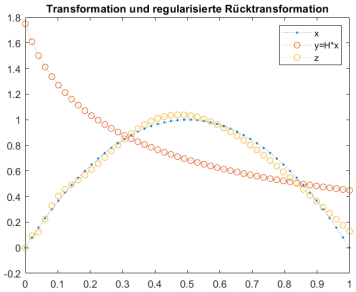
```
z = H\y;
```

verwenden Sie für die Rücktransformation

```
alpha= ...
z = (H'*H + alpha*eye(n))\H'*y;
```

für ein *sehr* kleines $\alpha > 0$.

Für welchen Wert α können Sie $n = 50$ Datenpunkte transformieren und rücktransformieren, so dass in der Zeichnung **x** und **z** möglichst gut übereinstimmen? Hier rechts ist die Übereinstimmung noch nicht besonders gut. Bei günstig gewähltem α sollte sie deutlich besser sein.



Regularisierung für inverse Probleme tritt in vielen Anwendungsgebieten auf. Das ist ein hochaktuelles Thema, obwohl es dazu noch nicht einmal deutsche Wikipedia-Einträge gibt[26].

Die Übungsunterlagen gehen auf die Theorie nicht ein[27], sie wollen hier nur zeigen: selbst theoretisch anspruchsvollen Methoden lassen sich mit einfachen MATLAB-Befehlen ausführen.

[26] siehe die Stichworte *Regularization (mathematics)* und *Tikhonov regularization* in der englischen Wikipedia

[27] Erklärung der Bedeutung von α (nur für Interessierte): Der Ergebnisvektor **y** ist in der Praxis nicht völlig exakt bestimmbar. Wenn man relative Fehler der Größenordnung α im Ergebnis **y** zulässt, dann gibt es unendlich viele Ausgangsvektoren **x**, die im Rahmen der Genauigkeit den Ergebnisvektor **y** hätten erzeugen können. Der obige Kochrezept-Befehl findet unter all diesen Vektoren jenen mit kleinster 2-Norm.

Das ist zwar nicht notwendig der „richtige“ Ausgangsvektor, aber unter bestimmten Zusatzannahmen der „plausibelste“.

Ü 3.5 Newton-Verfahren für Systeme: Anleitung

Lösen Sie das folgende nichtlineare Gleichungssystem mit dem Newton-Verfahren.

$$\begin{aligned} 2x_1 - x_2 &= e^{-x_1} \\ x_1 + 2\sin x_2 &= \cos x_2 \end{aligned}$$

Es folgt eine Schritt-für-Schritt-Anleitung. Am Ende dieses Abschnittes gibt es auch ein Link zu einer fertigen Lösung. Dringender Rat: arbeiten Sie zuerst diese Anleitung durch (insbesondere, wenn Sie der Meinung sind, dass fertige Musterlösungen Zeit und Mühe sparen. . .)

Gleichung auf Nullstellen-Problem umformen

Bringen Sie zuerst die Gleichungen in die Form $\mathbf{f}(\mathbf{x}) = \mathbf{0}$.

$$\begin{aligned} 2x_1 - x_2 - e^{-x_1} &= 0 \\ x_1 + 2\sin x_2 - \cos x_2 &= 0 \end{aligned}$$

Jacobi-Matrix berechnen

Berechnen Sie die Jacobi-Matrix: $D_{\mathbf{f}}(\mathbf{x}) = \begin{bmatrix} 2 + e^{-x_1} & -1 \\ 1 & 2\cos x_2 + \sin x_2 \end{bmatrix}$

Startvektor wählen

Beginnen Sie mit dem Startvektor $\mathbf{x}^{(0)} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$. Werten Sie $\mathbf{f}$ und $D_{\mathbf{f}}$ für $\mathbf{x}^{(0)}$ aus – besser noch: lassen Sie MATLAB das tun (wird gleich erklärt).

Korrektur-Vektor berechnen

Sie berechnen einen Korrektur-Vektor $\Delta\mathbf{x}^{(0)}$ als Lösung eines linearen Gleichungssystems. Dessen Matrix ist die Jacobi-Matrix, auf der rechten Seite steht $-\mathbf{f}(\mathbf{x}^{(0)})$.

Zur Theorie lesen Sie bitte im Skriptum nach. Dort finden Sie:

> Iterationsvorschrift
>
> $$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \Delta\mathbf{x}^{(k)}$$
>
> mit $\Delta\mathbf{x}^{(k)}$ als Lösung von $D_f(\mathbf{x}^{(k)})\Delta\mathbf{x}^{(k)} = -\mathbf{f}(\mathbf{x}^{(k)})$

MATLAB löst lineare Gleichungssysteme mit dem „Bergab-Strich“-Operator `\`. Ein System $A\mathbf{x} = \mathbf{b}$ würden Sie in der Form `x = A\b` lösen. Für das Newton-Verfahren können Sie Gleichungslösen `Deltax = Df\(-f)` und Korrekturschritt `xneu = x + Deltax` als Einzeiler zusammenfassen: `xneu = x - Df\f`

Programmierung

Programmieren Sie $\mathbf{f}(\mathbf{x})$ und $D_{\mathbf{f}}(\mathbf{x})$ als MATLAB-Funktionen und speichern Sie die beiden als Dateien `f.m` und `Df.m`.

Abschnitte Ü 2.1 und Ü 2.2 haben die Eingabe von Vektoren und Matrizen schon kurz erklärt. Die Matrixelemente in einer Zeile können Sie durch Beistriche oder Leerzeichen trennen. Strichpunkte oder der Beginn einer neuen Zeile im Quelltext trennen die Zeilen in Matrizen und Spaltenvektoren.

Vervollständigen Sie die beiden folgenden Funktionsdateien!

```
function y = f(x)
y= [ ... %erste Komponente
     ... %zweite Komponente
   ];
end
```

Vektorwertige Funktion: Vektor als Eingabe, Vektor als Ergebnis!

```
function dy = Df(x)
dy= [ ... %erste Zeile
      ... %zweite Zeile
   ];
end
```

Matrixwertige Funktion: Vektor als Eingabe, Matrix als Ergebnis!

Prüfen Sie im *Command Window*, ob sich die beiden Funktionsdateien korrekt aufrufen lassen. Das sollte herauskommen:

```
>> x=[1; 1];
>> f(x)
ans =
    0.6321
    2.1426
>> Df(x)
ans =
    2.3679   -1.0000
    1.0000    1.9221
```

Achtung, setzen Sie Leerzeichen korrekt, wenn Sie die Jacobi-Matrix programmieren! Darauf weist schon Abschnitt Ü 3.2 hin; hier nochmal:

Leerzeichen um + und − bei Termen in Matrizen:
RICHTIG

- keine Lerzeichen. Beispiel `[ a+b, x+y ]`
- beiderseits Leerzeichen. Beispiel `[ a + b, x + y ]`

FALSCH:

- einseitig Lerzeichen. Beispiel `[ a +b, x +y ]`

Handbetriebene Iteration im Command Window

Wenn Funktion und Jacobi-Matrix korrekt programmiert sind, testen Sie im *Command Window* die Newton-Iteration. Beginnen Sie die Suche mit $\mathbf{x} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$.

```
>> x=[1; 1];
>> x = x - Df(x)\f(x)
x =
          0.395158347619808
          0.199928444993336
>> x = x - Df(x)\f(x)
x =
          0.449399668569469
          0.261761474532574
>> x = x - Df(x)\f(x)
x =
          0.449562385013556
          0.261217530152151
>> x = x - Df(x)\f(x)
x =
          0.449562377948136
          0.261217503068493
```

Der Startwert ist mäßig genau. Die zweite Iteration liefert aber immerhin schon drei signifikante Stellen; ab dann lässt sich quadratisches Konvergenzverhalten beobachten.

Ü 3.6 Lokale Funktionen

Die beiden Funktionsdateien zusammen mit der Newton-Iteration sind in eine Datei zum Herunterladen gepackt. Dieses Programm zeigt Ihnen die Verwendung lokaler Funktionen.

> Ob Sie Aufgaben in einer Datei mit lokalen Funktionen oder mit anonymous functions programmieren, oder ob Sie Ihre Lösung lieber in Skripts und Funktionen in getrennten Dateien aufteilen, bleibt Ihnen überlassen! Alle Varianten haben Vor- und Nachteile.

Holen Sie sich die Datei und führen Sie sie aus. Für eine sauber formatierte Version können Sie die Datei auch in ein Live-Script umwandeln oder die *publish*-Möglichkeit von MATLAB nützen. Siehe MATLAB-Hilfe, Stichwort *Publish and Share MATLAB Code*. So sollte die Ausgabe aussehen:

```
>> NewtonMusterAufgabe
Konvergenz nach 4 Iterationen
Nullstelle:
          0.449562377948136
          0.261217503068493
```

Damit sollte es leicht fallen, die folgende Aufgabe zu lösen:

Aufgabe 22:

Aufgabe 20 fragt nach einer Lösung des Gleichungssystems

$$\begin{aligned} f(x,y) = 4x - y + xy - 1 &= 0 \\ g(x,y) = -x + 6y + \log xy - 2 &= 0 \end{aligned}$$

durch Fixpunkt-Iteration. Nun sollen Sie das Newton-Verfahren anwenden. Orientieren Sie sich am Beispiel des vorigen Abschnittes und am Musterprogramm `NewtonMusterAufgabe.m`. Im Skriptum (Seite 27) ist dieses Beispiel auch durchgerechnet.

Aufgabe 23:

Lösen Sie das folgende nichtlineare Gleichungssystem mit dem Newton-Verfahren. Startvektor $[1; -1; 0]$. Iterieren Sie, bis aufeinanderfolgende Lösungen in der ∞-Norm, das ist in MATLAB `norm(...,inf)`, auf 10^{-12} übereinstimmen.

$$\begin{aligned} x^2 + \sin^2 y + z &= 1 \\ e^x + e^{-x} - yz &= 2 \\ x + y + z^2 &= 0 \end{aligned}$$

Aufgabe 24:

Gegeben sind die Funktionsgleichungen

$$\begin{aligned} y &= 2x^2 - 1 \\ y &= \sin(3x) \quad \text{für } -1 \le x \le 1 \quad . \end{aligned}$$

Zeichnen Sie beide Kurven und lesen Sie aus der Graphik die Koordinaten der Schnittpunkte auf eine Nachkommastelle genau ab.

Formen Sie beide Gleichungen auf $\cdots = 0$ um und berechnen Sie die Lösung des nichtlinearen 2×2-Systems mit dem Newton-Verfahren.

Verwenden Sie die Näherungen aus der graphischen Darstellung als Startwerte. Konvergenzkriterium: sollen sich in keiner Komponente um mehr als 10^{-9} unterscheiden.

Aufgabe 25:

In einer wässrigen Natriumacetat-Lösung bestimmt ein nichtlineares System von vier Gleichungen die Konzentration der Ionen H^+, OH^-, Ac^- und der undissoziierten Säure HAc. Gegeben sind $K_w = 10^{-14}, K_s = 10^{-4,75}$ und eine variable Konzentration C.

$H^+ \cdot OH^-$	=	K_w	Ionenprodukt des Wassers
$H^+ \cdot Ac^-$	=	$K_s \cdot HAc$	Säure-Dissoziationsgleichgewicht
$HAc + Ac^-$	=	C	Acetat-Gesamtkonzentration
$Ac^- + OH^-$	=	$C + H^+$	Elektroneutralität

Wenn Sie die unbekannten Konzentrationen H^+, OH^-, Ac^-, HAc mit Variablen x_1, x_2, x_3, x_4 bezeichnen, lässt sich die Aufgabe umformulieren zu

$$\mathbf{f}(\mathbf{x}) = 0 \quad \text{mit} \quad \mathbf{f}(\mathbf{x}) = \begin{bmatrix} -K_w + x_1 x_2 \\ x_1 x_3 - K_s x_4 \\ -C + x_3 + x_4 \\ C + x_1 - x_2 - x_3 \end{bmatrix}$$

Schreiben Sie eine MATLAB-Funktion `f(x,C)`, die für einen Eingabevektor $\mathbf{x} = (x_1, x_2, x_3, x_4)$ und gegebene Konzentration C den Vektor $\mathbf{f}(\mathbf{x})$ berechnet. ($K_w = 10^{-14}, K_s = 10^{-4,75}$ sind fix gegeben.) Schreiben Sie auch die entsprechende Funktion für die Jacobi-Matrix und ein Newton-Verfahren zur Nullstellenberechnung. Haben Sie eine Lösung gefunden, dann ist $-\log_{10} x_1$ der pH-Wert.

Bei diesem Beispiel sind gute Startwerte wichtig. Empfehlung:

$$\mathbf{x}^{(0)} = [10^{-7}; 10^{-7}; C; 0]$$

Zum Vergleich Testwerte für $C = 0.01$

```
>> x0=[1.e-7;1.e-7;0.01;0];
>> f(x0,0.01)
ans =
  1.0e-008 *

  -0.00000000000000
   0.10000000000000
                  0
                  0

>> df(x0)
ans =

   0.00000010000000   0.00000010000000                   0                   0
   0.01000000000000                   0   0.00000010000000  -0.00001778279410
                  0                   0   1.00000000000000   1.00000000000000
   1.00000000000000  -1.00000000000000  -1.00000000000000                   0

>> x0=x0-df(x0)\f(x0,0.01)
x0 =

   0.00000000035638
   0.00000019964362
   0.00999980071276
   0.00000019928724
```

Ü 3.7 Lösen von Systemen mit *fsolve*

Gleichungssysteme in der Form $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ kann MATLABs *Optimization Toolbox* mit dem Befehl `fsolve` lösen. Der Aufruf funktioniert ähnlich wie `fzero`.

Angenommen, Sie haben die Funktion aus Abschnitt Ü 3.5 als Funktionsdatei `f.m` programmiert. Dann lauten Aufruf und Ergebnis

```
>> fsolve(@f,[1,1])

Equation solved.

fsolve completed because the vector of function values is near zero
as measured by the default value of the function tolerance, and
the problem appears regular as measured by the gradient.

<stopping criteria details>

ans =
 0.449562378440135 0.261217525788617
```

Sie ahnen vielleicht aufgrund der umfangreichen Ausgabe: so einfach kann das nicht gewesen sein. Ist es auch nicht – die zugrunde liegenden Methoden können wir im Rahmen dieser Übung nicht behandeln.

Ab der achten Stelle unterscheiden sich die Resultate von denen des Newton-Verfahrens. Wenn Sie auf hohe Genauigkeit Wert legen, müssten Sie die Standard-Einstellungen von `fsolve` anpassen[28].

Aufgabe 25 lässt sich mit Standardeinstellungen gar nicht lösen. (`fsolve` behauptet zwar auch „Equation solved“, aber das Resultat liegt total daneben.)

[28] Challenge: wer findet Einstellungen, so dass `fsolve` auf alle 16 double-Stellen genau dieselben Werte liefert wie das Newton-Verfahren? Ich hab probiert und es nicht geschafft. C.B.

Aufgabe 26:

Lösen Sie Aufgaben 22, 23 und 24 mit `fsolve`. Stellen Sie das Anzeigeformat im *command window* auf `format long g`, und vergleichen Sie die Genauigkeit mit den Resultaten des Newton-Verfahrens.

Ü 3.8 Isolinien- und Flächen-Diagramme

Aufgabe 27: Darstellung einer Funktion $z = f(x, y)$ durch Isolinien

Stellen Sie die Funktion

$$f : \mathbb{R}^2 \to \mathbb{R}, \quad z = xe^{-x^2-y^2}$$

als Isolinien-Diagramm dar. Am einfachsten geht das mit dem Befehl `ezcontour`: (amerikanisch-englisch ausgesprochen klingt das wie *ea-sy-contour*)

```
>> ezcontour('x*exp(-x^2 -y^2)')
```

Achtung, Hochkommas ' nicht vergessen!

Wenn Sie aber genauer festlegen wollen, wie und was sie darstellen wollen, verwenden Sie den „richtigen“ Befehl `contour`. Die MATLAB-Hilfe zum Stichwort `contour` erklärt, wie das geht. Sie können nach einem dort angegebenen Musterbeispiel (je nach Matlab-Version scrollen Sie drei, vier Abbildungen nach unten) arbeiten:

Für ein Isolinien-Diagramm (engl: *contour plot*) der Funktion $z = xe^{-x^2-y^2}$ im Bereich $-2 \leq x \leq 2, -2 \leq y \leq 3$ erzeugen Sie zuerst x- und y-Werte auf einem Gitter in der xy-Ebene.

```
>> x = -2:0.2:2;
>> y = -2:0.2:3;
>> [X,Y] = meshgrid(x,y);
```

Beachte: in englischen Texten steht oft `.2` (ohne 0 vor dem Dezimalpunkt), wo bei uns 0,2 stehen würde. Auch der MATLAB-Ausdruck `-2:.2:2` ist korrekt.

Für die (x, y)-Wertepaare berechnen Sie eine Matrix Z mit dem Befehl

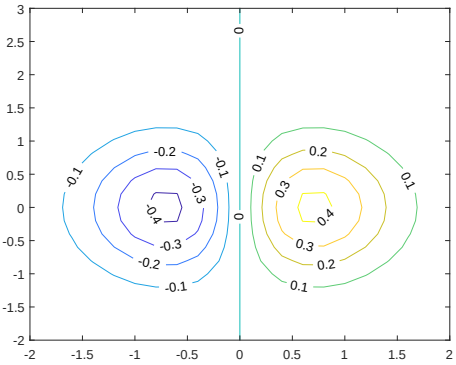
```
>> Z = X.*exp(-X.^2-Y.^2);
```

Anschließend erzeugen Sie den Isolinien-Plot:

```
>> contour(X,Y,Z,'ShowText','on')
>> colormap cool
```

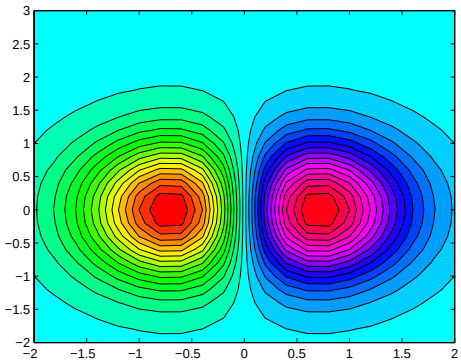
Es gibt verschiedene Varianten des contour-Befehls. In der MATLAB-Hilfe finden Sie weitere Beispiele (Befehle contourf, contour3)

So sollte das Isolinien-Diagramm aussehen, das Sie mit den obigen Befehlen erzeugt haben.



Weitere Varianten des `contour`-Befehls: `contour(X,Y,Z,30)` erzeugt 30 Isolinien; `contour(X,Y,Z,-1:0.1:1)` erzeugt Isolinien für Werte von -1 bis 1 in 0,1-Schritten.

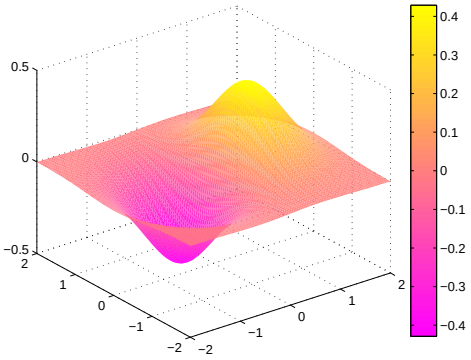
Erforschen Sie die Optionen des `contour`-Befehles. Ihre Aufgabe ist, herauszufinden, mit welchen Befehlen sie den Plot möglichst genau so aussehen lassen, wie hier gezeigt:



Aufgabe 28: Darstellung einer Funktion $z = f(x, y)$ als Fläche im Raum

Ganz ähnlich wie `contour` funktioniert der Befehl `surf`. Informieren Sie sich in der MATLAB-Hilfe und zeichnen Sie die Funktion aus Aufgabe 27 als Fläche im Raum.

Ihre Aufgabe ist, herauszufinden, mit welchen Befehlen sie den Plot möglichst genau so aussehen lassen, wie hier gezeigt (Datenbereich, Auflösung, Farbgebung, Farbbalken...).



Ü 3.9 Graphische Suche nach Nullstellen für nichtlineare Gleichungssysteme in zwei Variablen

Für eine Funktion $f : \mathbb{R} \to \mathbb{R}$ in der Form $y = f(x)$ lassen sich Nullstellen aus dem Funktionsgraph ablesen. Das kennen Sie schon aus der Mittelschule und der ersten Vorlesung.

Eine Funktion $f : \mathbb{R}^2 \to \mathbb{R}$ können Sie in der Form $z = f(x, y)$ als Fläche im Raum oder als Isoliniengrafik darstellen und so ebenfalls graphisch die Lage von Nullstellen finden. Eine Nullstelle ist in diesem Fall ein Wertepaar (x, y), das die Gleichung $f(x, y) = 0$ erfüllt.

Für ein System zweier nichtlinearer Gleichungen in der Form

$$f_1(x, y) = 0$$
$$f_2(x, y) = 0$$

kann man die Funktion

$$f(x, y) = [f_1(x, y)]^2 + [f_2(x, y)]^2$$

definieren. Die ist genau dann gleich 0, wenn f_1 und f_2 gleich Null sind. Kleiner als 0 kann f (aufgrund ihrer Definition als Summe zweier Quadrate) auch nicht werden. Die Nullstellen von f sind also zugleich auch globale Minima von f und Lösungen des nichtlinearen Gleichungssystems $f_1 = 0$, $f_2 = 0$.

Die nächsten beiden Aufgaben illustrieren diesen Zusammenhang.

Einleitung zu den Aufgaben 29 und 30

Die Funktion $f : \mathbb{R}^2 \to \mathbb{R}$, gegeben durch

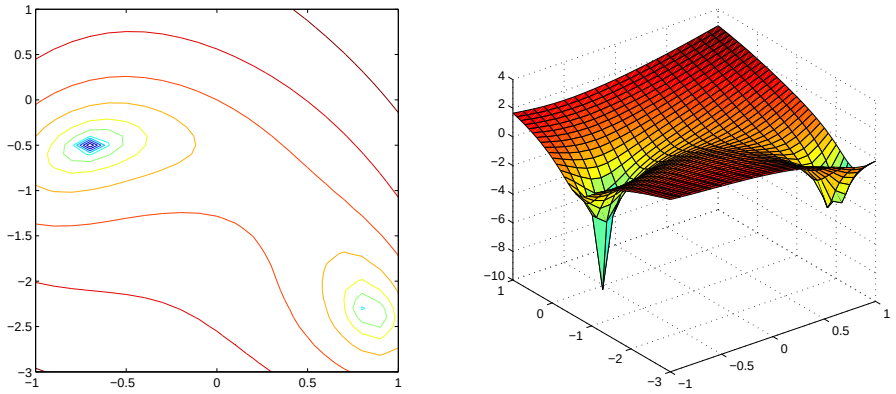
$$f(x, y) = \left[x^2 + \sin y\right]^2 + \left[e^x + y\right]^2$$

hat Minimalwert 0 für genau jene Wertepaare $(x; y)$, die Lösung des folgenden nichtlinearen Gleichungssystems sind:

$$\begin{aligned} x^2 + \sin y &= 0 \\ e^x + y &= 0 \end{aligned}$$

Aufgabe 29: Nullstellen einer Funktion in zwei Variablen (Teil 1)

Zeichnen Sie für $\log f$, den Logarithmus der oben gegebenen Funktion, im Bereich $-1 < x < 1$ und $-3 < y < 1$ eine `contour`- und eine `surf`-Grafik in der Art der Abbildung. (In der logarithmischen Darstellung erkennt man die Lage des Minimums deutlicher als in direkter Darstellung von f.) Verwenden Sie ≈ 20 Gitterpunkte in x- und ≈ 40 Gitterpunkte in y-Richtung.



Aufgabe 30: Nullstellen einer Funktion in zwei Variablen (Teil 2)

Lösen Sie das oben gegebene nichtlineare Gleichungssystem mit dem Newton-Verfahren. Finden Sie beide Lösungen im Bereich $-1 < x < 1$ und $-3 < y < 1$. Passende Startwerte können Sie aus der grafischen Darstellung (Aufgabe 29) entnehmen. Abbruchkriterium: Aufeinanderfolgende Iterationen unterscheiden sich, gemessen als Summe der Beträge aller Komponenten-Differenzen, weniger als 10^{-8}.

Ü 4 Vierte Übungseinheit

Inhalt der vierten Übungseinheit:

- Eindeutig, nicht eindeutig und gar nicht lösbare Systeme
- Fehlerempfindlichkeit, Matrixnormen, Konditionszahl
- Determinante, Komplexität
- Modelle an Daten anpassen
- Lineare Datenmodelle

Eine Anleitung für diese Übungseinheit

Im aktuellenVorlesungsskript behandelt Kapitel 3 lineare Gleichungssysteme mit quadratischer Matrix (mit genausoviel Gleichungen wie Unbekannten). In den Folien zur 4. Vorlesung können sie dazu die Abschnitte

1. Lösbarkeit, Fehlerempfindlichkeit und
4. Direkte Verfahren

zur Wiederholung durchklicken.

Insbesondere die Skriptum-Kapitel 3.2, 3.4, 3.6, 3.7.1 und 3.8 bringen Material, das direkt mit den Übungaufgaben 31 – 35 zusammenhängt.

Die weiteren Aufgaben in dieser Übungseinheit befassen sich damit, wie sich aus Datensätzen die Parameter eines Modells schätzen lassen; zum Beispiel: aus der Anzahl von infizierten Personen die Parameter eines exponentiellen Wachstumsmodells, Anleitung dazu in den Übungsunterlagen Ü 4.4.2.

Dazu sollten Sie in den Vorlesungsfolien das Thema

5. Überbestimmte Systeme

durchklicken und Kapitel 5 im Vorlesungsskript durchblättern. Wir werden die Theorie noch ausführlicher behandeln, aber zum praktischen Einsatz reicht vorläufig, zu wissen:

- Ein System mit mehr Gleichungen als Unbekannten ist im Regelfall nicht exakt lösbar
- Eine Näherungslösung (die „am wenigsten falsche Lösung“) lässt sich mit der Methode der kleinsten Fehlerquadrate bestimmen.
- MATLABS Operator \ findet bei überbestimmten Gleichungssystemen die kleinste-Quadrate-Näherungslösung.

Ü 4.1 Gleichungssysteme, Lösungsmannigfaltigkeiten

Können Sie ein lineares Gleichungssystem mit zwei Gleichungen und zwei Unbekannten immer lösen?

> Die Faustregel „Bei genausoviel Gleichungen wie Unbekannten gibt es immer eine Lösung“ ist falsch. FALSCH! **FALSCH!!**

Bei den drei Gleichungssystemen

$$\begin{aligned} x + y &= 2 \\ 2x + 2y &= 4 \end{aligned} \qquad \begin{aligned} x + y &= 2 \\ 2x + 2y &= 3 \end{aligned} \qquad \begin{aligned} x + y &= 2 \\ x + 2y &= 3 \end{aligned}$$

sehen Sie hoffentlich mit freiem Auge: Beim ersten und beim dritten sind $x = 1,\ y = 1$ Lösungen. Das mittlere System ist unlösbar. Beim ersten System gibt es allerdings noch unendlich viele weitere Lösungen.

Wiederholen Sie, was Sie dazu in Mathematik 1 gelernt haben. Im Skriptum, Kapitel 3.4 finden Sie weitere Informationen. Sie sollten mit folgenden Themen vertraut sein:

- Matrix-Rang, Rang der erweiterten Matrix, MATLAB-Befehle `rank(A), rank([A,b])`, Fallunterscheidungen zur Lösbarkeit.
- Stufenform eines Gleichungssystems, MATLAB-Befehl `rref([A,b])`
- Allgemeine Lösung des homogenen Systems, Nullraum, MATLAB-Befehl `null(A)`
- Spezielle Lösung des inhomogenen Systems, MATLAB-Befehle `A\b` bei vollem Rang, `pinv(A)*b` bei lösbaren Systemen mit Rang kleiner Zeilenzahl.
- Bei nicht lösbaren Systemen liefert `pinv(A)*b` die „am wenigsten falsche“ Antwort (mit kleinstem Fehler in der 2-Norm)

Aufgabe 31: Gleichungssysteme, Lösungsmannigfaltigkeiten

Welche der folgenden Gleichungssysteme $A\mathbf{x} = \mathbf{b}$ sind eindeutig, mehrdeutig oder gar nicht lösbar? Geben Sie, wenn möglich, die (oder eine) Lösung an. Bei mehrdeutigen Lösungen: wie viele freie Parameter hat die Lösungsschar?

Weitere Erläuterungen, Arbeitsschritte
Die möglichen Fälle zur Lösbarkeit entscheiden Sie über Rang der Matrix und Rang der erweiterten Matrix, Befehle `rank(A), rank([A,b])`. Als Alternative können Sie auch `rref([A,b])` verwenden.

Wenn eine eindeutige Lösung existiert, ist `A\b` der richtige Befehl.

Wenn eine Lösungsschar existiert, liefert `pinv(A)*b` eine mögliche Lösung, und zwar jene mit kleinstem Absolutbetrag. Der Standard-Befehl `A\b` kann auch funktionieren; er liefert einen Lösungsvektor mit möglichst vielen Komponenten gleich 0.

Die vollständige Darstellung einer Lösungsmannigfaltigkeit können Sie aus dem Ergebnis von `rref([A,b])` ablesen. Das erfordert aber noch etwas Umformen.

Alternative: Die allgemeine Lösung besteht aus einer speziellen Lösung (bestimmt mit `pinv(A)*b`) plus einer Linearkombination der Spaltenvektoren des Nullraumes (bestimmt mit `null(A)`).

1. $$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}, \mathbf{b} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

2. $$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}, \mathbf{b} = \begin{bmatrix} 2 \\ 0 \\ 1 \end{bmatrix}$$

3.
$$A = \begin{bmatrix} 10 & 9 & 8 & 7 \\ 6 & 5 & 4 & 3 \\ 2 & 1 & 0 & -1 \\ -2 & -3 & -4 & -5 \end{bmatrix}, \mathbf{b} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

4.
$$A = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \\ 1 & 4 & 9 & 16 \\ 1 & 8 & 27 & 64 \end{bmatrix}, \mathbf{b} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

Aufgabe 32: LR-Zerlegung

Das Skriptum behandelt in Kapitel 3 Gauß-Elimination, LR-Zerlegung, Vorwärts-Elimination, Rücksubstitution. Dazu listet es auch JAVA-Codes. Hier eine Portierung in MATLAB. Wenn Sie diese Aufgabe durcharbeiten, haben Sie Ihren ersten eigenen Gleichungslöser programmiert.

> (Warnung: Die Programme dienen zur Illustration der Grundform der Verfahren. Für den praktischen Einsatz sind sieso noch nicht geeignet, es wäre zu ergänzen: Pivotierung, Absicherungen gegen singuläre Matrizen, Division durch Null...

Verwenden Sie die Angabe des Rechenbeispiels im Skript, Abschnitt3.6: Gegeben sei das Gleichungssystem $A \cdot \mathbf{x} = \mathbf{b}$ mit

$$A = \begin{bmatrix} 5 & 6 & 7 \\ 10 & 20 & 23 \\ 15 & 50 & 67 \end{bmatrix} \quad , \quad \mathbf{b} = \begin{bmatrix} 6 \\ 6 \\ 14 \end{bmatrix}$$

Sie können damit alle Zwischenergebnisse nachvollziehen.

Tippen Sie nicht ab, das ist zu fehleranfällig. Verwenden Sie Copy-Paste aus der PDF-Datei!

Schritt 1: LR-Zerlegung. Die beiden Matrizen L und R werden aus Effizienz-Gründen nicht extra gespeichert, sondern ersetzen die A-Matrix. Die L-Matrix ohne die Eins-Diagonale besetzt das Dreieck unterhalb der Hauptdiagonale; Die R-Matrix befüllt das obere Dreieck.

```
%% Zerlegung A = LR
% geht schief, wenn Null in der Diagonale auftritt!
for k=1:n
    for i=k+1:n
        A(i,k)=A(i,k)/A(k,k);
        for j=k+1:n
            A(i,j) = A(i,j)-A(i,k)*A(k,j);
        end
    end
end
```

Die Zerlegung ist damit abgeschlossen, vergleichen Sie (und beachten Sie: die beiden Ausgabe-Matrizen überschreiben die entsprechenden Bereiche in A):

$$A \text{ (Eingabe)} \begin{bmatrix} 5 & 6 & 7 \\ 10 & 20 & 23 \\ 15 & 50 & 67 \end{bmatrix} \quad , \quad L \cdot R \text{ (Ausgabe)} \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 4 & 1 \end{bmatrix} \cdot \begin{bmatrix} 5 & 6 & 7 \\ 0 & 8 & 9 \\ 0 & 0 & 10 \end{bmatrix}$$

Erst jetzt kommt die rechte Seite b ins Spiel: Die Vorwärts-Elimination löst $Ly = b$ und berechnet so das Zwischenresultat y. Im Code wird y gleich als x gespeichert.

```
%% Vorwärts-Elimination Ly=b
x = b;
for i=1:n
    for j=1:i-1
        x(i) = x(i) - A(i,j)*x(j);
    end
end
```

Das Zwischenresultat y bzw. x entspricht der letzten Spalte nach Abschluss des Eliminationsverfahrens, vor der Rücksubstitution, vergleichen Sie im Skript:

$$[A\,\mathbf{b}]^{(2)} = \begin{bmatrix} 5 & 6 & 7 & 6 \\ 0 & 8 & 9 & -6 \\ 0 & 0 & 10 & 20 \end{bmatrix}$$

```
%% Rücksubstitution Rx=y
for i=n:-1:1
    for j=i+1:n
        x(i) = x(i)-A(i,j)*x(j);
    end
    x(i) = x(i)/A(i,i);
end
```

Nun ist x die Lösung des Systems.

Wie gut ist diese ganz naive Implementierung im Vergleich zu „richtigen“ Gleichungslöse-Programmen? Testen Sie dazu mit zufällig erzeugten Systemen

```
n=4;
A=floor(10*rand(n));
b=floor(10*rand(n,1));
```

und vergleichen Sie MATLABs Ergebnis `x=A\b` mit unserem Spielzeug-Programm. Meistens sollten die Ergebnisse gleich sein. Testen Sie viele Systeme und geben Sie an: in wieviel Prozent der Fälle versagt unser naives Programm? Warum?

Ü 4.2 Fehlerempfindlichkeit im Eliminationsverfahren

Aufgabe 33: Fehlerempfindlichkeit

Lösen Sie mit Matlabs Bergabstrich das Gleichungssystem $A\mathbf{x} = \mathbf{b}$ (es ist das Gleichungssystem Nr. 4 aus Aufgabe 31),

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \\ 1 & 4 & 9 & 16 \\ 1 & 8 & 27 & 64 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

Addieren Sie nun zur Matrix künstliche Fehler-Matrizen der Form $\delta A =$ `0.01*rand(4)` und vergleichen Sie die Lösung mit jener des ungestörten Systems.

Berechnen Sie die relativen Fehler der Matrixdaten[29] $||\delta A||/||A||$ und der Lösung $||\delta \mathbf{x}||/||\mathbf{x}||$. Testen Sie einige Fälle, auch mit kleineren oder größeren Störtermen, und schätzen Sie daraus das *maximale Verhältnis* von rel. Fehler der Lösungs zu rel. Fehler in den Daten ab.

Was bedeutet in diesem Zusammenhang der Wert `cond(A)`?

Ändern Sie nun die Matrix des Gleichungssystems zu

$$A = \begin{bmatrix} 1 & 1/2 & 1/3 & 1/4 \\ 1/2 & 1/3 & 1/4 & 1/5 \\ 1/3 & 1/4 & 1/5 & 1/6 \\ 1/4 & 1/5 & 1/6 & 1/7 \end{bmatrix}$$

(Hinweis: `hilb(4)` liefert genau diese Matrix!) und wiederholen Sie Ihre Untersuchungen.

Ü 4.3 Entwicklung der Determinante, Komplexität

Aufgabe 34: Determinante

Gegeben sind

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, B = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}, C = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \\ 1 & 4 & 9 & 16 \\ 1 & 8 & 27 & 64 \end{bmatrix}, D = \begin{bmatrix} 10 & 9 & 8 & 7 \\ 6 & 5 & 4 & 3 \\ 2 & 1 & 0 & -1 \\ -2 & -3 & -4 & -5 \end{bmatrix}$$

Berechnen Sie von diesen Matrizen die Determinante auf verschiedene Arten:

1. Mit der Matlab-Funktion `det( )`
2. durch *LR*-Zerlegung, Matlab-Funktion `lu( )`, aus dem Produkt der Diagonalelemente von R[30]. Der Befehl `diag(R)` extrahiert übrigens die Hauptdiagonale einer Matrix als einen Vektor, der Befehl `prod` multipliziert alle Elemente.
3. Auch das primitive *LR*-Programm aus Aufgabe 32 liefert die Determinante als Produkt der Diagonalelemente (sogar vorzeichenrichtig, weil es keine Pivot-Suche durchführt). Testen Sie!
4. Die folgende Funktion implementiert das Standardverfahren zur Berechnung der Determinante durch Entwicklung nach Unterdeterminanten. (Copy-Paste aus PDF, lässt sich auch von der Übungs-Homepage herunterladen)

```
function d = mydet(A)
n = length(A);
if n==1
    d = A(1,1);
else
    d = 0;
```

[29] Sie brauchen Matrixnormen dazu! Informieren Sie sich in Skript und Vorlesungsfolien!

[30] Vorsicht, es kann sein, dass Matlabs `lu()` während der Rechnung Zeilen der Matrix vertauscht (Pivotierung). Mit jedem Zeilentausch wechselt das Vorzeichen der Determinante. Hat also `lu()` eine ungerade Anzahl von Zeilen vertauscht, hat das Produkt der Diagonalelemente das falsche Vorzeichen. Im Rahmen dieser Aufgabe sind Vorzeichen Glückssache.

```
        sig = 1;
        for i=1:n
            d = d + sig*A(1,i)*mydet(A(2:n,[1:i-1,i+1:n]));
            sig = -sig;
        end
    end
```

Wiederholen Sie auch die bekannten Standardverfahren (Regel von Sarrus, Entwicklung nach Unterdeterminanten) und rechnen Sie für A, B, C von Hand nach.

> Achtung: die Regel von Sarrus (Gitterzaunregel) gilt nur für 2×2- und 3×3-Matrizen. Wer 4×4-Determinanten auf diese Weise berechnen will, rechnet falsch. FALSCH! **FALSCH!!**

Aufgabe 35:

Ein Rechenverfahren, das zwar korrekt rechnet, aber ewig dafür braucht, hat nur theoretischen Wert. Die Berechnung der Determinante durch Entwicklung nach Unterdeterminanten ist ein Beispiel dafür.

Die Funktion `mydet` aus Aufgabe 34 implementiert das Standardverfahren zur rekursiven Berechnung der Determinante durch Entwicklung nach Unterdeterminanten.

Testen Sie die Rechenzeit`mydet` im Vergleich zum MATLAB-Standardbefeht `det`. Einfache Testmatrizen liefert z. B. die Funktion `magic(n)`.

MATLAB bietet mit den Befehlen `tic` und `toc` eine Art Stoppuhr an, mit der sie die Rechenzeit messen können. Sie könnten dazu Code der folgenden Art verwenden:

```
>> A = magic(4)
A =
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
>> tic; mydet(A); toc
Elapsed time is 0.000203 seconds.
```

Versuchen Sie nun auch etwas größere Matrizen als in Aufgabe 34. Bis zu welchem n lässt sich die Determinante

1. in weniger als zehn Sekunden,
2. in weniger als einer Minute
3. in weniger als zehn Minuten

berechnen? Das hängt natürlich von der Leistungsfähigkeit Ihres Rechners ab. Schätzen Sie aufgrund Ihrer Zeitmessungen und der Tabelle im Skriptum, Kapitel 3.7.1, wie lange die Berechnung für $n = 15$ und $n = 20$ dauern würde.

Ü 4.4 Modell-Funktionen an Daten anpassen

Gegeben sind Datenpunkte, gesucht sind Parameter für ein Modell, das diese Daten beschreibt. Im einfachsten Fall: eine „schöne Kurve“, die sich dem Verlauf der Datenpunkte gut anpasst.

Statistisch aussagekräftig ist das nur, wenn (deutlich) mehr Datenpunkte gegeben als Modellparameter gesucht sind. Das führt auf überbestimmte Gleichungssysteme.

Material und Links dazu siehe Einleitung. Noch einmal vorweg die wichtige Aussage:

> MATLABs Bergabstrich \ findet für überbestimmte Gleichungssystemen (die in der Regel nicht exakt lösbar sind) ein „am wenigsten falsches “ Ergebnis.

Etwas exakter gesagt: MATLABs Bergabstrich \ findet für überbestimmte Gleichungssystemen die bestmögliche Näherung im Sinn der kleinsten Fehlerquadrate.

MATLAB bietet in seiner *curve fitting toolbox* viele Hilfsmittel, um Kurven oder Flächen an gegebene Daten anzupassen. Bevor Sie sich darin vertiefen, sollten Sie die grundlegenden Ideen und Methoden verstehen. Darum geht es in dieser Übungseinheit.

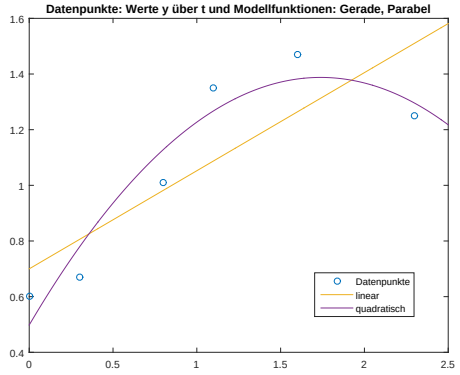
Ü 4.4.1 Eine Variable: Beispiel aus der MATLAB-Hilfe

Die MATLAB-Hilfe (R2019–R2021) diskutiert Beispiele unter MATLAB › Data Import and Analysis › Descriptive Statistics › Programmatic Fitting › MATLAB Functions for Polynomial Models (geben Sie *“Programmatic Fitting”* im Suchfenster der Hilfe ein, dann finden Sie dieses Beispiel rasch.)

Hier ist das Beispiel im Abschnitt *MATLAB Functions for Polynomial Models* für Sie aufbereitet, so dass Sie aus den Angaben der MATLAB-Hilfe und den Zusatzinforrmationen folgende Aufgabe lösen können:[31]

Gegeben sind Messwerte y zu verschiedenen Zeiten t als Spaltenvektoren:

```
t = [0 0.3 0.8 1.1 1.6 2.3]';
y = [0.6 0.67 1.01 1.35 1.47 1.25]';
```



Sie sollen eine Funktion finden, die den Zusammenhang $y = y(t)$ gut modelliert. Einfache Ansätze sind: eine lineare Funktion, oder ein allgemeineres Polynom, zum Beispiel:

$$y = a_0 + a_1 t \qquad \text{oder}$$
$$y = a_0 + a_1 t + a_2 t^2$$

Dafür gibt es in MATLAB die Befehle `polyfit` und `polyval` und das *Basic Fitting User Interface*. Tipp: lassen Sie mit `plot` die Punkte zeichnen. Klicken Sie im *figure*-Fenster auf Tools › Basic Fitting und probieren Sie aus!

[31] Eigentlich müssen Sie nur die Beispiel-Befehle aus der Matlab-Hilfe in Ihre eigene Skript-Datei kopieren. Aber bitte schalten Sie ihr Hirn nicht aus, während Sie Strg + C und Strg + V verwenden!

Die Polynomfunktionen passen sich in diesem Beispiel nicht besonders gut an die Daten an. Versuchen Sie stattdessen ein lineares Modell[32] mit allgemeineren Funktionen, nämlich

$$y(t) = a_0 + a_1 e^{-t} + a_2 t e^{-t}$$

Vorgangsweise: Setzen Sie die gegebenen Datenpunkte in die Ansatzfunktion ein. Sie erhalten pro Wertepaar eine (lineare!) Gleichung in den drei Unbekannten a_0, a_1, a_2.

$$\begin{aligned} 0{,}60 &= a_0 + a_1 e^{0{,}0} + a_2 \cdot 0{,}0 \cdot e^{0{,}0} \\ 0{,}67 &= a_0 + a_1 e^{-0{,}3} + a_2 \cdot 0{,}3 \cdot e^{-0{,}3} \\ 1{,}01 &= a_0 + a_1 e^{-0{,}8} + a_2 \cdot 0{,}8 \cdot e^{-0{,}8} \\ &\dots \quad \dots \end{aligned}$$

Im MATLAB-Skript erstellen Sie die Koeffizientenmatrix X (die MATLAB-Hilfe sagt: *form the design matrix*). Achtung: `t` und `y` müssen Spaltenvektoren sein!

```
X = [ones(size(t))  exp(-t)  t.*exp(-t)];
```

Der Aufbau dieser Matrix ist der entscheidende Punkt, den Sie verstehen müssen. Die Koeffizienten von a_0 im obigen Gleichungssystem sind immer eins, daher enthält die erste Spalte von X lauter Einser, wie ein Volksschulzeugnis. Die Koeffizienten von a_1 sind e^{-t}-Werte, daher steht in der zweiten Spalte von X ein Vektor von e^{-t}-Werten, und so weiter.

Bitte mitdenken: Natürlich ist nicht immer automatisch die erste Spalte von X ein Volksschulzeugnis – das hängt von den gewählten Ansatzfunktionen ab, und von deren Reihenfolge. Aber als einfache Regel gilt: „Die Spalten von X enthalten die Werte der Ansatzfunktionen."

Die rechte Seite sind einfach nur die y-Werte. Die Modellkoeffizienten a_0, a_1, a_2 berechnet der Bergabstrich-Operator \ als bestmögliche Näherung aus dem überbestimmten Gleichungssystem:

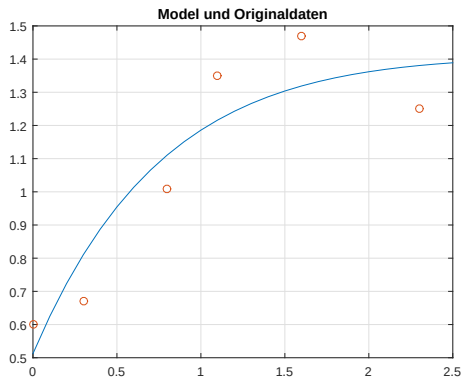
```
a = X\y
a =

    1.3983
   -0.8860
    0.3085
```

Also ist das bestmögliche Datenmodell gegeben durch

$$y = 1{,}3983 - 0{,}8860\, e^{-t} + 0{,}3085\, t e^{-t}.$$

Jetzt können Sie Ihre Modellfunktion auswerten (in genügend hoher Auflösung, also an vielen t-Punkten in engem Abstand) und zusammen mit den Originaldaten zeichnen.

```
T = (0:0.1:2.5)';
Y = [ones(size(T))  exp(-T)  T.*exp(-T)]*a;
plot(T,Y,'-',t,y,'o'), grid on
title('Modell und Originaldaten')
```



[32] *linear* sind hier nicht die verwendeten Ansatzfunktionen e^{-t} und te^{-t}. *Lineares Modell* bedeutet, dass diese Funktionen *linear kombiniert* werden: also ein Ansatz der Form „Koeffizient 1 mal Funktion 1 plus Koeffizient 2 mal Funktion 2…"

Auch hier bitte mitdenken: das Einsetzen von t-Daten in das Modell geschieht elegant wieder in Matrix-Vektor-Form: Die Matrix Y ist ähnlich aufgebaut wie die vorher verwendete X-Matrix. Multiplikation von Y mit Koeffizientenvektor $\mathbf{a}$ liefert Modellwerte.

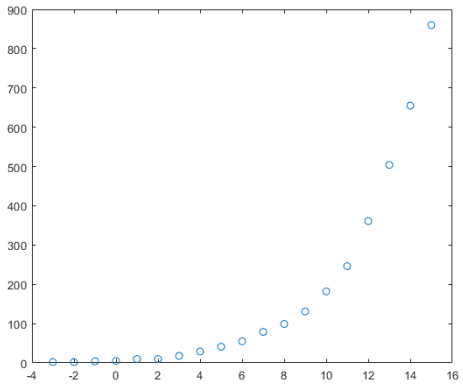
Alternative Auswertung: (in Wirklichkeit dieselbe Rechnung wie oben, nur für die a-Vektorkomponenten einzeln angeschrieben – vielleicht ist es so leichter verständlich)

```
Y = [ones(size(T))  exp(-T)  T.*exp(-T)]*a;  % Auswertung in Matrix-Vektor-Form
%
Y = a(1) + exp(-T)*a(2) + T.*exp(-T)*a(3); % Alternativ: komponentenweise Schreibung
```

Ü 4.4.2 Anpassen eines exponentiellen Wachstumsmodells

Gegeben sind (t, y)-Wertepaare. (MATLAB-Quelltext, copy-paste-fähig, nur die einfachen Anführungszeichen bei `'o'` müssen Sie ausbessern!)

```
t = -3:15;
y = [2 2 4 5 10 10 18 29 41 55 79 ...
99 131 182 246 361 504 655 860];
plot(t,y,'o')
```



Gesucht sind die Parameter a und b eines exponentiellen Wachstumsmodells

$$y = a \exp(bt) \ .$$

Es handelt sich bei y um die COVID-19-Fälle in Österreich (aufsummiert), mit Zeitachse t in Tagen vom 26. Februar bis 15. März 2020.

Wenn Sie hier, so wie vorher im Abschnitt Ü 4.4.1, die y- und t-Werte in die Modellgleichung einsetzen, erhalten Sie pro Wertepaar eine Gleichung für die unbekannten Parameter a und b.

$$\begin{aligned} 2 &= a \exp(-3b) \\ 2 &= a \exp(-2b) \\ 4 &= a \exp(-b) \\ \vdots &\qquad \vdots \\ 860 &= a \exp(15b) \end{aligned}$$

Im Unterschied zu vorher sind die Gleichungen jedoch nichtlinear in a und b. Wir behandeln echt nichtlineare überbestimmte Systeme erst später. Dieses System lässt sich aber zu einem linearen Problem vereinfachen.

Dieser einfache Trick zeigt dir, wie du ein nichtlineares System loswirst. . .

Logarithmieren Sie die Gleichungen:

$$\begin{aligned} \log 2 &= \log a - 3b \\ \log 2 &= \log a - 2b \\ \log 4 &= \log a - b \\ &\vdots \qquad\quad \vdots \\ \log 860 &= \log a - 15b \end{aligned}$$

(Natürlich steht log hier für den natürlichen Logarithmus!) Nennen Sie noch $\log a = \bar{a}$, dann steht hier ein überbestimmtes lineares Gleichungssystem in den Unbekannten $\bar{a}$ und b.

In MATLAB transformieren Sie die Datenvektoren von Zeilen- zu Spaltenvektoren. Die Koeffizientenmatrix X und rechte Seite dieses Systems erzeugen Sie dann so

```
t = t';
y = y';
X = [ones(size(t)) t];
rhs = log(y);
```

Der Bergabschrägstrich liefert die kleinste-Quadrate-Schätzung für die unbekannten Parameter $\bar{a} = \log a$ und b.

```
ab=X\rhs
```

Für die hier gegebenen Daten ist `ab(1)` $= \bar{a} = \log a = 1.7602$ und damit der Parameter $a = \exp(\bar{a}) = 5.8135$. Das ist der (vom Modell geschätzte) Anfangswert für $y(0)$, wobei hier $t = 0$ der 29. Februar ist.

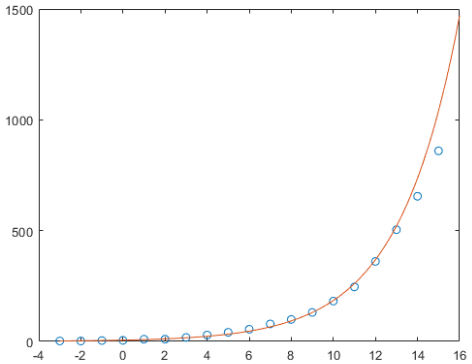
Der Wert `ab(2)` $= b = 0.3458$ ist die Wachstumskonstante im exponentiellen Modell. Um den Faktor $\exp(b) = 1.4132$ multiplizieren sich pro Tag die y-Werte, das entspricht einem täglichen Zuwachs um 41%. Die Verdopplungszeit T_2 in Tagen erhalten Sie aus

$$T_2 = \frac{\log 2}{b} = 2.0043 \quad .$$

Erinnern Sie sich noch, das war die beunruhigende Datenlage vor drei Jahren. Schauen wir uns die Daten und das geschätzte Wachstumsmodell genauer an.

Wir brauchen für die Zeitachse Zeitpunkte in feinerer Auflösung und gleich mal einen Tag in die Zukunft extrapoliert. `linspace` liefert hundert Zeitpunkte, das reicht. Für diesen `T`-Vektor werten wir das Modell aus.

```
a = exp(ab(1));
b = ab(2);
T = linspace(-3,16)';
Y = a*exp(b*T);
plot(t,y,'o',T,Y)
```



Zum Glück liegen die letzten beiden Datenpunkte schon unter der Modellkurve.

Vorsicht! Daraus allein hätte sich (mit Datenstand von März 2020) keine fundierte Vorhersage ableiten lassen. Dazu sind das Modell und unsere Datenanpassung zu stark vereinfacht. Was sich damals aussagen ließ: Ein exponentielles Wachstumsmodell mit den hier berechneten Parametern sieht in der grafischen Darstellung – bis auf die letzten beiden Datenpunkte – plausibel aus; **wenn sonst nichts das Wachstum bremst**, sagt dieses Modell in zwei Wochen katastrophale Fallzahlen voraus.

Im Nachhinein lassen sich die Modell-Voraussagen überprüfen. Am Ende dieses Abschnittes ist dazu eine grafische Darstellung bis Anfang April ergänzt.

Vorher versuchen wir aber noch eine etwas verfeinerte Modellrechnung. Denn unser logarithmierender Linearisierungs-Trick verursacht einen statistischen Fehler: die Parameter-Schätzung minimiert die Fehler in den logarithmierten Daten. Dadurch werden die (weniger aussagekräftigen) Datenpunkte zu Beginn stärker gewichtet als die (aktuelleren) Daten gegen Ende des Zeitintervalls.

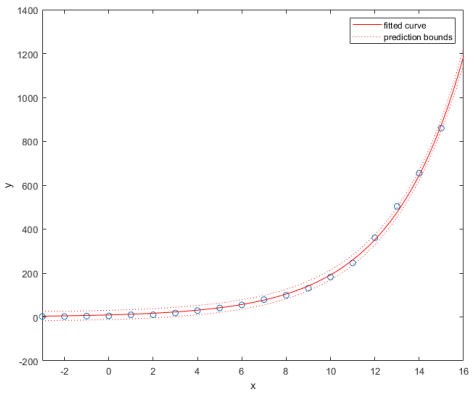
Wir werden uns noch eingehender mit der direkten Anpassung des nichtlinearen Datenmodells $y = a\exp(bt)$ befassen, aber jetzt lassen wir MATLAB arbeiten. Wenn bei Ihnen die *Curve Fitting Toolbox* installiert ist, funktioniert der Befehl
`modell = fit(t,y,'exp1')` Achtung, da steht 1, eins, nicht klein-ell bei `'exp1'` !)
und liefert als Ergebnis

```
modell =
     General model Exp1:
     modell(x) = a*exp(b*x)
     Coefficients (with 95% confidence bounds):
       a =       9.162  (7.889, 10.44)
       b =       0.304  (0.294, 0.314)
```

Zu den geschätzten Parametern wird auch ein 95%-Konfidenzintervall angegeben. **Vorsicht!** Diese Angaben gehen davon aus, dass die Daten tatsächlich einem exponentiellen Gesetz gehorchen und die einzelnen Datenpunkte nur zufallsbedingt mit einer gewissen Wahrscheinlichkeitsverteilung von der exponentiellen Kurve abweichen. Die Aussage ist also: *Wenn* ein exponentielles Wachstum vorläge, *dann wären das* die geschätzten Parameter.

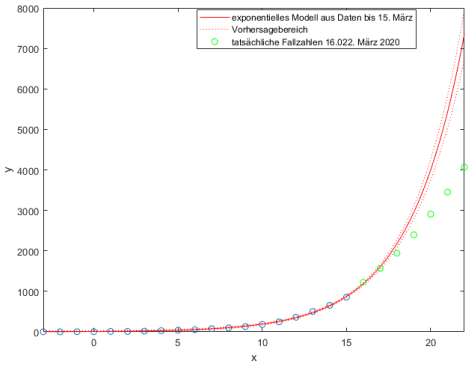
Wenn das Modell über MATLABs `fit`-Befehl berechnet ist, gibt es zusätzliche Optionen für die graphische Darstellung: Das Schlüsselwort `'predobs'` zeichnet auch Unsicherheits-Grenzen ein. Der `xlim([-3,16])`-Befehl stellt den x- (hier t-) Achsenbereich ein; ein größerer Endwert würde das Modell weiter in die Zukunft hin extrapolieren.

```
plot(t,y,'o')
hold on
xlim([-3,16])
plot(modell,'predobs')
hold off
```



Nochmal **Vorsicht!** Die engen Unsicherheitsgrenzen sollen nicht den Eindruck hoher Genauigkeit oder Verlässlichkeit entstehen lassen. Die Kurven fußen auf der Annahme, dass der Verlauf wirklich einem exponentiellen Modell folgt.

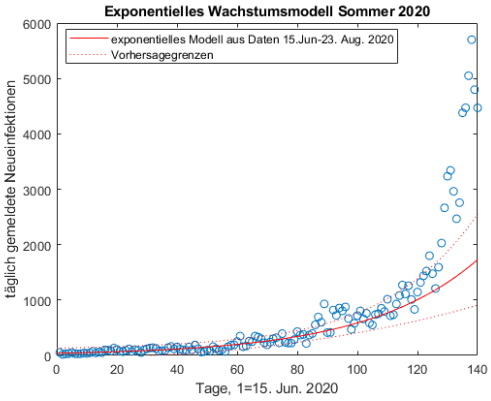
Wir haben das Modell in der obigen Graphik nicht weiter in die Zukunft extrapoliert, weil die Modellannahmen stark vereinfacht und unsicher waren. Tatsächlich konnte der erste Lockdown das exponentielle Wachstum bremsen. Die mit mehr Datenpunkten ergänzte Grafik rechts zeigt, dass einige Tage nach Beginn des Lockdowns die gemeldeten Fallzahlen von der exponentiellen Modellkurve abweichen. Es stellt sich ein anderes Wachstumsmodell ein.



Aufgabe 36: COVID-19 im Sommer und Herbst 2020

Sie können hier Daten zu den täglichen Neuinfektionen in Österreich, Zeitraum 140 Tage (15. Juni bis 1. Nov 2020) herunterladen.

Berechnen Sie aus den ersten 70 Datenpunkten die Parameter eines exponentiellen Wachstumsmodells, sowohl in der einfachen Form (linearer Fit an die logarithmierten Daten), als auch mit MATLABS *curve fitting toolbox*, Befehl `fit`. Vergleichen Sie das angepasste exponentielle Modell mit den tatsächlichen Daten. Stellen Sie Ihre Modellrechnung etwa so dar, wie in der Grafik rechts.



Interpretation: Bis etwa Tag 120 liegen die tatsächlichen Fallzahlen einigermaßen innerhalb der Vorhersagegrenzen, aber dann brechen die Daten nach oben aus. Offensichtlich hatte sich zu diesem Zeitpunkt (Mitte Oktober) der Ausbreitungsmodus signifikant verändert. Konsequenter Weise mussten ab Anfang November wieder Ausgangsbeschränkungen verordnet werden.

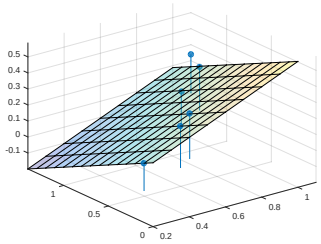
Ü 4.4.3 Mehrere Variable

Die MATLAB-Hilfe zeigt *Multiple Regression* als nächstes Beispiel unter
MATLAB » Data Import and Analysis » Descriptive Statistics » Programmatic Fitting » Multiple Regression .

Beschreibung und Illustration siehe auch 4. Vorlesung, vorletzte Folie. Auch das Skriptum erklärt in Kapitel 5.4: Anpassen eines linearen Modells (einer Ausgleichsebene).

Hier eine Musterlösung zur Aufgabe der vorletzten Vorlesungsfolie. Im Vergleich zur Lösung in der MATLAB-Hilfe zeigt sie auch das Zeichnen der Ebene und der Originaldaten.

```
%% Datenpunkte: Variable x1, x2 und Messwert y
x1 = [.2 .5 .6 .8 1.0 1.1]';
x2 = [.1 .3 .4 .9 1.1 1.4]';
y  = [.17 .26 .28 .23 .27 .24]';

%% Matrix des ueberbestimmten Systems
% fuer  Ansatz y = a0 + a1*x1 + a2*x2
X = [ones(size(x1))  x1  x2];

%% Loesung: Koeffizientenvektor
a = X\y
%% Auswertung und Zeichnung auf feinem Gitter
[XX,YY]=meshgrid(0.2:0.1:1,0:0.1:1.4);
ZZ= a(1) + a(2)*XX + a(3)*YY;

surf(XX,YY,ZZ,'FaceAlpha',0.3),
axis tight, hold on
% Originaldaten
stem3(x1,x2,y,'.','MarkerSize',25)
hold off
```



Ü 4.5 Weitere Aufgaben zu Linearen Datenmodellen

Aufgabe 37: Hochwasser

Für die Blies (einen Nebenfluss der Saar) sollen die Hochwasserstände am Pegel Neunkirchen aus den Wasserständen des Pegels Ottweiler und des Pegels Hangard vorhergesagt werden. Es liegen die Daten der Scheitelwasserstände von 12 Winterhochwässern aus den Jahren 1963–1971 vor: (Daten können Sie von der Übungs-Homepage runterladen)

Wasserstand in cm												
Neunkirchen y	172	309	302	283	443	298	319	419	361	267	337	230
Ottweiler x_1	93	193	187	174	291	184	205	260	212	169	216	144
Hangard x_2	120	258	255	238	317	246	265	304	292	242	272	191

Quelle: U. Maniak, Hydrologie und Wasserwirtschaft, Springer, 1988

Wir unterstellen den Daten das lineare Modell $a_0 + a_1 x_1 + a_2 x_2 = y$. In diesem Ansatz sind a_0, a_1 und a_2 unbekannte Koeffizienten, die aus den zwölf gegebenen Werte-Tripeln möglichst gut bestimmt werden sollen.

Berechnen Sie die Koeffizienten des linearen Modells, und geben Sie auch den Differenzenvektor zwischen Modellvorhersage und Messwerten an. (Dieses Beispiel ist im Skriptum Kapitel 5.4 ausführlich durchgerechnet)

Aufgabe 38: Jack Sparrows Kompass

Egal, ob Sie eine Geophysik-Vorlesung besucht oder Johnny Depp in *Pirates of the Caribbean* gesehen haben, Sie haben gelernt, dass eine Kompassnadel nicht immer nach Norden zeigt.

Die Abweichung heißt magnetische Deklination, ihre Kenntnis ist nach wie vor von großer Wichtigkeit für die Seefahrt. Deswegen gibt die Zentralanstalt für Meteorologie und Geodynamik regelmäßig für alle Landeshauptstädte aktuelle Werte der magnetischen Deklination an:[a] (Daten von Übungs-Hompage herunterladbar!)

[a] Update 2017: Diese Tabelle wird von der ZAMG inzwischen nicht mehr bereitgestellt, vermutlich, weil die Bedeutung der Seefahrt in den Landeshauptstädten abgenommen hat)

Deklinationswerte der einzelnen Landeshauptstädte
(östliche Deklination bezogen auf Jahresmitte 2008)

Stadt	Länge x	Breite y	Deklination D
Wien (WIK)	16.37	48.20	3.0000
Eisenstadt	16.52	47.85	3.0333
St.Pölten	15.63	48.20	2.9000
Graz	15.45	47.07	2.7833
Linz	14.30	48.30	2.5500
Klagenfurt	14.31	46.62	2.5000
Salzburg	13.03	47.80	2.2500
Innsbruck	11.40	47.27	1.8833
Bregenz	9.77	47.50	1.4000

Finden Sie für diese Daten eine Anpassung der Form

$$z(x, y) = a_1 + a_2 x + a_3 x^2 + a_4 y$$

Geben Sie die Differenz $D - z(x, y)$ zwischen den tatsächlichen Deklinationswerten und den angepassten Werten an. Welcher Wert wird am schlechtesten approximiert?

Moderne Modelle für das Erdmagnetfeld (International Geomagnetic Reference Field IGRF, World Magnetic Model WMM) verwenden im Prinzip denselben Ansatz: Sie passen ein Modell an Messdaten an. Aber im Unterschied zu unserem Polynom-Ansatz für 9 Datenpunkte verarbeiten Sie riesige Datenmengen und verwenden als Ansatz Kugelflächenfunktionen.

Aufgabe 39: Für Weicheier

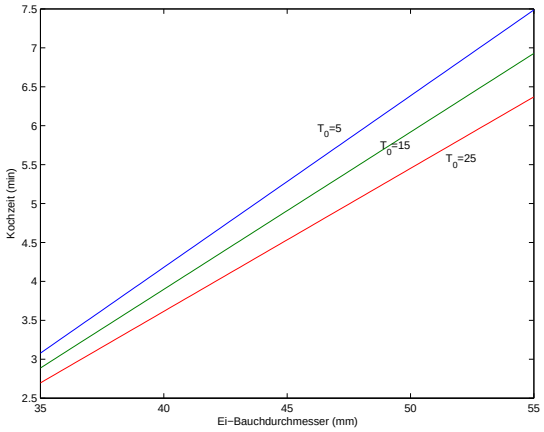
Der Wiener Physiker Werner Gruber beschäftigt sich mit der Thermodynamik des Eierkochens und gibt nebenstehende Werte für die Kochzeit eines perfekt weichen Frühstücks-Eis an, abhängig von Bauchdurchmesser d (nicht seiner, der des Eis) und Ausgangstemperatur T_0 (Daten können Sie von der Übungs-Homepage runterladen)

Durchmesser d (mm)	Ausgangstemperatur T_0 (C)	Kochzeit (min:sek)
35.0	4	3:10
40.0	4	4:10
45.0	4	5:20
50.0	4	6:30
35.0	20	3:00
40.0	20	3:40
45.0	20	4:40
50.0	20	5:50

Finden Sie für diese Daten eine Anpassung der Form

$$t(d, T_0) = a_1 + a_2 d + a_3 T_0 + a_4 d T_0$$

Zeichnen Sie ein Diagramm ähnlich dem hier gezeigten (x-Achse: Eidurchmesser, y-Achse: Kochzeit), in dem Sie für Anfangs/-temperaturen 5,15,25 Grad die Kochzeiten eintragen.

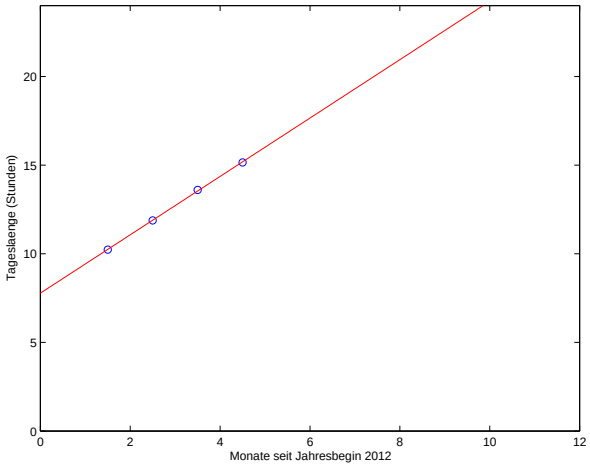


Aufgabe 40: Ein Reich, in dem die Sonne nicht untergeht

Eine Aufgabe zu Datenmodellen mit polynomialen und allgemeineren Ansatzfunktionen.

Die folgende Tabelle gibt für Leoben die Tageslänge (von Sonnenauf- bis -untergang) in Stunden für vier Tage von Februar bis Mai an. Klar ersichtlich ist ein eindeutiger Trend zu immer längeren Tagen.

Datum kalendarisch	Monate seit Jahresbeg.	Tageslänge Stunden
15. Feb. 2019	1,5	10,23
15. Mär. 2019	2,5	11,88
15. Apr. 2019	3,5	13,60
15. Mai. 2019	4,5	15,15



Anlageberater und andere begabte Verkäufer könnten Ihnen anhand der Grafik zu diesen Daten überzeugend erklären, dass spätestens Ende Oktober die Sonne nicht mehr untergeht.

Bevor Sie nun in Solar-Aktien investieren, erstellen Sie selber Modelle für diesen Datensatz

1. Lineare Regression: $y = a_0 + a_1 x$
2. Polynom-Ansatz $y = a_0 + a_1 x + a_2 x^2 + a_3 x^3$
3. Datenmodell $y = a_1 + a_2 \cos\left(x\frac{\pi}{6}\right) + a_3 \sin\left(x\frac{\pi}{6}\right)$

Stellen Sie die Datenpunkte und die drei Modelle in einer Grafik dar (Achsen-Bereich wie oben). Beantworten Sie (durch Ablesen aus der grafischen Darstellung) folgende Fragen: Ab wann geht laut Modell 1 die Sonne in Leoben nicht mehr unter (Tageslänge 24 h)? Für wann sagt Modell 2 ewige Nacht voraus (Tageslänge 0 h)? Wann ist laut Modell 3 Sommersonnenwende (Tageslänge maximal)?

Nur Modell 3 kann verlässliche Voraussagen treffen, weil es einen problemgerechten Ansatz verwendet: die periodischen Schwankungen des Sonnenlaufes werden durch Sinus- und Cosinusterme genähert.

Ü 5 Fünfte Übungseinheit

Inhalt der fünften Übungseinheit:

- Nichtlineare Datenmodelle
- Kurze Wiederholung: lineare und nichtlineare Datenmodelle
- Kenntnisnachweis-Musteraufgaben
- Bonus-Material: MATLAB-Werkzeuge zum Anpassen von Funktionen an Daten

Ü 5.1 Überbestimmte nichtlineare Systeme, Gauß-Newton-Verfahren

Das Gauß-Newton-Verfahren findet Näherungslösungen (im Sinn der kleinsten Fehlerquadrate) für überbestimmte nichtlineare Systeme. Die Grundidee ist, ähnlich wie bei der Lösung nichtlinearer Gleichungssysteme mit dem Newton-Verfahren, mit der *Jacobimatrix* des nichtlinearen Systems iterativ Korrekturterme zu berechnen. Die Aufgaben 41 und 42 zeigen ausführlich den Rechenweg.

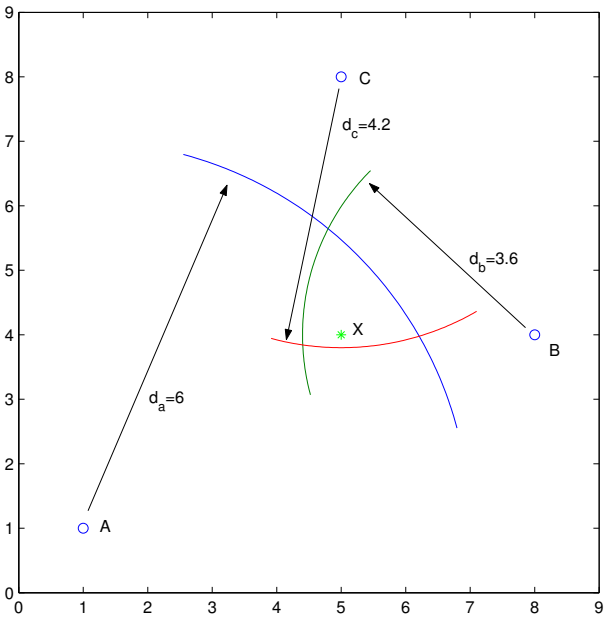
Aufgabe 41: Standortbestimmung durch Trilateration

Die Abstände von drei festen Punkten A, B, C in der xy-Ebene zu einem unbekannten Punkt X sind (etwas ungenau) bekannt. Gesucht ist eine möglichst gute Positionsbestimmung.

Punkt	x	y	Entfernung
	1	1	6
	8	4	3,6
	5	8	4,2

Die entsprechenden Gleichungen lauten:

$$\begin{aligned}\sqrt{(x_1-1)^2+(x_2-1)^2} &= 6\\ \sqrt{(x_1-8)^2+(x_2-4)^2} &= 3.6\\ \sqrt{(x_1-5)^2+(x_2-8)^2} &= 4.2\end{aligned}$$



Den drei Gleichungen entsprechen drei Kreise im $\mathbb{R}^2$. Sie haben keinen gemeinsamen Schnittpunkt. Dementsprechend gibt es keine exakte Lösung des überbestimmten Gleichungssystems.

Schreiben Sie ein MATLAB-Programm, das die kleinste-Quadrate-Anpassung für den Standort findet.

Bemerkung: Diese Aufgabe ist die 2-dimensionale Vereinfachung einer Positionsbestimmung im Raum. Die GPS-Technik beruht auf diesen geometrischen Grundlagen. Ein GPS-Empfänger misst die Abstände (genauer: Signallaufzeiten) zu mehreren Satelliten. Die Signale sind stark verrauscht, erst Ausgleichsrechnung und zusätzliche Datenfilterung gewährleisten eine auf einige Meter genaue Position.

Kurzfassung: Aufgabenstellung und Lösungsweg

Gegeben: überbestimmtes nichtlineares System

$$\mathbf{f}(\mathbf{x}) = \mathbf{0}\ , \quad \mathbf{x} \in \mathbb{R}^n\ , \quad \mathbf{f}(\mathbf{x}) \in \mathbb{R}^m, \quad m > n$$

Iterative Lösung des linearisierten Systems: Ausgehend von Startvektor $\mathbf{x}^{(0)}$ bestimmt man eine Korrektur $\Delta\mathbf{x}$.

Die Rechenvorschrift des Newton-Verfahrens für $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ ergibt ein überbestimmtes lineares System mit der Jacobimatrix D_f

$$D_f \cdot \Delta\mathbf{x} = -\mathbf{f}(\mathbf{x})$$

Verbesserte Lösung $\mathbf{x}^{(1)} = \mathbf{x}^{(0)} + \Delta\mathbf{x}$.

Konkret für die vorliegende Aufgabe:

$$\mathbf{f}(\mathbf{x}) = \begin{bmatrix} \sqrt{(x_1-1)^2+(x_2-1)^2} - 6 \\ \sqrt{(x_1-8)^2+(x_2-4)^2} - 3.6 \\ \sqrt{(x_1-5)^2+(x_2-8)^2} - 4.2 \end{bmatrix}, \qquad D_f = \begin{bmatrix} \frac{x_1-1}{\sqrt{(x_1-1)^2+(x_2-1)^2}} & \frac{x_2-1}{\sqrt{(x_1-1)^2+(x_2-1)^2}} \\ \frac{x_1-8}{\sqrt{(x_1-8)^2+(x_2-4)^2}} & \frac{x_2-4}{\sqrt{(x_1-8)^2+(x_2-4)^2}} \\ \frac{x_1-5}{\sqrt{(x_1-5)^2+(x_2-8)^2}} & \frac{x_2-8}{\sqrt{(x_1-5)^2+(x_2-8)^2}} \end{bmatrix}$$

Mit Startvektor $\mathbf{x} = \begin{bmatrix} 5 \\ 4 \end{bmatrix}$ erhält man

$$\mathbf{f}\left(\begin{bmatrix} 5 \\ 4 \end{bmatrix}\right) = \begin{bmatrix} -1 \\ -3/5 \\ -1/5 \end{bmatrix}, \ D_f = \begin{bmatrix} \frac{4}{5} & \frac{3}{5} \\ -1 & 0 \\ 0 & -1 \end{bmatrix}, \ \text{lin. Syst.} \begin{bmatrix} \frac{4}{5} & \frac{3}{5} \\ -1 & 0 \\ 0 & -1 \end{bmatrix} \cdot \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 3/5 \\ 1/5 \end{bmatrix}$$

Ergibt $\Delta x_1 = 1/25, \Delta x_2 = 7/25 \quad \longrightarrow$ verbesserte Position $[5.04; 4.28]$.

Sie wiederholen diese Rechenschritte und iterieren, bis sich die Position im Rahmen einer vernünftig gewählten Fehlerschranke nicht mehr ändert. Vergleichen Sie dazu den Übungs-Abschnitt Ü 3.5: der Rechengang und vor allem auch die Implementierung in MATLAB sind nahezu gleich. Unterschied: die Matrix des Gleichungssystems ist nun nicht mehr quadratisch und auch nicht konstant; sie hängt von den jeweils aktuellen Näherungswerten für $\mathbf{x}$ ab.

Natürlich sagt dieses Kochrezept nichts zur Theorie überbestimmter linearer Systeme oder zu den Eigenschaften der so berechneten Ausgleichslösung. Die Idee, ein nichtlineares System mittels Jacobi-Matrix linearisiert anzunähern, wird Ihnen aber in der Praxis ständig begegnen.

Aufgabe 42: Gauß-Newton-Verfahren in Wikipedia

Die Angabe zum folgenden Beispiel stammt aus der englischen bzw. französische Wikipedia (Stichworte *Gauss-Newton algorithm* bzw. *Algorithme de Gauss-Newton*). Lösen Sie die Aufgabe mit folgender Anleitung in MATLAB.[33]

[33] Wer dieses Beispiel in der englischen oder französischen Wikipedia-Seite durchliest und bei der Abbildung der Modellkurve auf "more details" klickt, findet fertigen MATLAB-Code zur Lösung des Beispiels und Zeichnen der Kurve. Aber das Verstehen von fremdem Code ist auch nicht einfach.

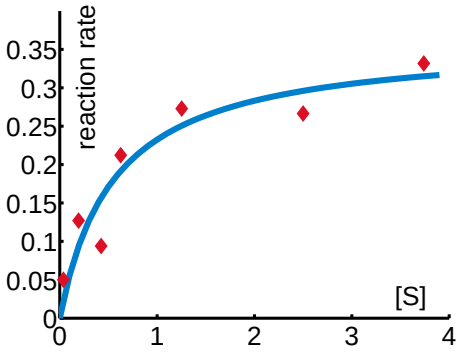
Biologie-Experimente zur Beziehung zwischen Substanz-Konzentration x und Reaktionsrate y in einer durch Enzyme vermittelten Reaktion haben die Daten in folgender Tabelle ergeben:

x	0.038	0.194	0.425	0.626	1.253	2.500	3.740
y	0.05	0.127	0.094	0.2122	0.2729	0.2665	0.3317

Gesucht ist eine Kurve (Modellfunktion) der Form

$$y = a\frac{x}{b+x}$$

die im Sinn der kleinsten Quadrate die Daten am besten approximiert. Die Parameter a und b sind zu bestimmen.
Einsetzen der Daten ergibt sieben nichtlineare Gleichungen in den beiden Unbekannten a, b.



$$a\frac{0.038}{b+0.038} - 0.05 = 0$$
$$a\frac{0.194}{b+0.194} - 0.127 = 0$$
$$\vdots$$
$$a\frac{3.740}{b+3.740} - 0.3317 = 0$$

Das Gleichungssystem in Vektor-Schreibweise:

$$\mathbf{f}(\mathbf{x}) = 0 \text{ mit } \mathbf{f} : \mathbb{R}^2 \to \mathbb{R}^7$$

Die Jacobi-Matrix D_f dieses Systems ist eine 7×2-Matrix. Zeile i enthält die partiellen Ableitungen der i-ten Gleichung nach den Unbekannten a und b.

$$(D_f)_{i1} = \frac{x_i}{b+x_i}, \quad (D_f)_{i2} = -\frac{ax_i}{(b+x_i)^2}$$

Der weitere Rechenweg verläuft völlig analog zur Anleitung im Übungs-Abschnitt Ü 3.5 und der Musterlösung auf Seite 27 im Skriptum: Wählen Sie als Startwert `[a;b]=[0.9;0.2]`, werten Sie $\mathbf{f}$ und D_f aus. Die kleinste-Quadrate-Näherung aus dem überbestimmten Gleichungssystem $D_f\mathbf{\Delta x} = -\mathbf{f}$ liefert den Korrekturvektor $\mathbf{\Delta x}$.

Beachten Sie: bei einer quadratischen Jacobi-Matrix liefert der MATLAB-Befehl `Df(x0)\f(x0)` die Lösung des Gleichungssystems; hat die Jacobi-Matrix mehr Zeilen als Spalten (überbestimmtes System), liefert derselbe Befehl *nicht* die Lösung (es gibt keine exakte Lösung!), sondern eine Anpassung mit geringstmöglichem Restfehler (die „am wenigsten falsche Antwort“)!

Bonus-Frage: Unser Newton-Musterprogramm findet die verbesserte Näherung mit dem MATLAB-Befehl

```
x = x0 -  Df(x0)\f(x0);     % Newton-Schritt
```

Das Wikipedia-Beispiel nennt die Jacobi-Matrix J_f und verwendet für denselben Schritt (in MATLAB-Code geschrieben) den Befehl

```
x = x0 - (J'*J)\(J'*f(x0));
```

Warum einerseits `Df(x0)\f(x0)` und andererseits `(J'*J)\(J'*f(x0))`? Was wird da jeweils gerechnet? Vor- und Nachteile der beiden Varianten im Vergleich?

Aufgabe 43: Kalorimeterversuch

Der Datensatz `Kalorimeter.dat` (klick!) (auch auf der Übungsseite zum Runterladen) enthält Wertepaare (Zeit t in Minuten, Temperatur T in Celsius) zu einem Kalorimeterversuch, gemessen ab der Einbringung eines Versuchsobjektes. Die Temperatur sinkt zuerst rasch, steigt dann aber wieder und gleicht sich langsam der Umgebungstemperatur an.

Der Verlauf der Temperatur T als Funktion der Zeit t soll durch ein Modell der Form

$$T(t) = T_0 + C_1 \exp(-\lambda_1 t) + C_2 \exp(-\lambda_2 t)$$

beschrieben werden. Bestimmen Sie die Parameter dieses nichtlinearen Modells ausgehend von den Startwerten

$$T_0 = 6 \qquad C_1 = -1 \qquad \lambda_1 = 1/10 \qquad C_2 = 20 \qquad \lambda_2 = 2$$

(drei Iterationen des Newton-Verfahrens reichen aus.) Zeichnen Sie Messpunkte und Modellfunktion.

Ü 5.2 Kurze Wiederholung: lineare und nichtlineare Datenmodelle

Die Aufgaben zur Anpassung von Funktionen an Daten beschränkten sich in der vorigen Einheit auf *lineare Modelle*. Dabei bedeutet „linear“: die gesuchte Anpassung f ist eine *Linearkombination* von n Basisfunktionen $\phi_1, \phi_2, \ldots, \phi_n$ in der Form

$$f = a_1\phi_1 + a_2\phi_2 + \cdots + a_n\phi_n \quad .$$

Wichtig: Die Basisfunktionen $\phi_1, \ldots, \phi_n$ können beliebige, auch nichtlineare Funktionen sein – es sind die gesuchten Koeffizienten $a_1, \ldots, a_n$, die nur linear im Ansatz auftreten!

> Datenanpassung mit linearen Modellen führt auf ein überbestimmtes Gleichungssystem für die gesuchten Koeffizienten $a_1, \ldots, a_n$. In den Spalten der Matrix stehen die Werte der Basisfunktionen $\phi_1, \ldots, \phi_n$ an den gegebenen Datenpunkten.

Verwirrender Weise wird für die Anpassung einer Ausgleichs-Geraden auch der Begriff „lineare Regression“ verwendet. Unter „polynomiale Regression“ versteht man die Anpassung eines Polynoms an gegebene Daten, wobei es sich aber um ein lineares Datenmodell handelt.

> Unterscheiden Sie:
>
> - Typ der Anpassungsfunktionen: Gerade ("lineare Regression"), Polynom ("polynomiale Regression"), Winkel- oder Exponentialfunktionen...
> - Verknüpfung der Ansatzfunktionen: linear, nichtlinear. Dazu gehören die Begriffe lineares/nichtlineares Datenmodell.

Vergleichen Sie: Lineare Modelle

Aufgabe 38: Ansatz $z(x,y) = a_1 + a_2 x + a_3 x^2 + a_4 y$.
Hier sind die Basisfunktionen $\phi_1(x,y) = 1, \phi_2(x,y) = x, \phi_3(x,y) = x^2, \phi_4(x,y) = y$.

Aufgabe 39: Ansatz $t(d, T_0) = a_1 + a_2 d + a_3 T_0 + a_4 d T_0$.
Basisfunktionen sind $\phi_1(d,T_0) = 1, \phi_2(d,T_0) = d, \phi_3(d,T_0) = T_0, \phi_4(d,T_0) = d \cdot T_0$.

Aufgabe 40: Ansatz $y = a_1 + a_2 \cos\left(x\frac{\pi}{6}\right) + a_3 \sin\left(x\frac{\pi}{6}\right)$
Basisfunktionen sind $\phi_1(x) = 1, \phi_2(x) = \cos\left(x\frac{\pi}{6}\right), \phi_3(x) = \sin\left(x\frac{\pi}{6}\right)$.

Nichtlineare Modelle

Aufgabe 42: Ansatz $y = a\frac{x}{b+x}$. Hier sind die gesuchten Koeffizienten a und b nichtlinear verknüpft!

Aufgabe 43: Ansatz $T(t) = T_0 + C_1 \exp(-\lambda_1 t) + C_2 \exp(-\lambda_2 t)$.
Wären λ_1 und λ_2 bekannt, dann wäre es ein linearen Modell mit Parametern T_0, C_1 und C_2. Weil auch die beiden λ_i gesucht sind, wird die Aufgabe nichtlinear.

Aufgabe 41: Ansatz für Distanz d zwischen Punkt P und X: $d = \sqrt{(x_1 - p_1)^2 + (x_2 - p_2)^2}$; mehrere Datensätze für Punkt-Koordinaten $P = (p_1, p_2)$ und zugehörige Distanzen d sind gegeben. Gesucht: Koordinaten $X = (x_1, x_2)$.

Datenanpassung mit nichtlinearen Modellen führt auf ein überbestimmtes nichtlineares Gleichungssystem. Das Gauss-Newton-Verfahren löst dieses System iterativ; anders als beim Standard-Newton-Verfahren ist die Jacobi-Matrix hier nicht quadratisch; jeder Iterationsschritt löst ein überbestimmtes lineares Gleichungssystem.

Kapitel Ü 5.1 in diesen Unterlagen zeigt einen Standard-Lösungsweg zur nichtlinearen Anpassung. Der folgende Abschnitt Ü 5.4 zeigt weitere Möglichkeiten und stellt MATLAB-Werkzeuge vor. Vielleicht finden Sie diese Werkzeuge einfacher und intuitiver zu verwenden als den Lösungsweg aus Kapitel Ü 5.1.

Ü 5.3 Kenntnisnachweis

Für den ersten Kenntnisnachweis werden zwei Aufgaben zu folgenden Themenkreisen gestellt:

- Lineare Gleichungssysteme: Lösbarkeit, Lösungsmenge
- Nichtlineare Gleichungssysteme: Newton-Raphson-Verfahren, `fsolve` oder einfache Fixpunkt-Iteration.
- Zeichnen von Kurven, graphische Lösung von Gleichungen.
- Darstellen von Funktionen $z = z(x,y)$ als Isolinien oder Flächen, graphisches Finden von Minima/Maxima.
- Lineare Datenmodelle: Anpassen von Funktionen an Daten

Hier folgen drei Muster-Angaben.

Muster-Angabe A

Die Funktion

$$f(x,y) = \left[x^2 + \sin y\right]^2 + \left[e^x + y\right]^2$$

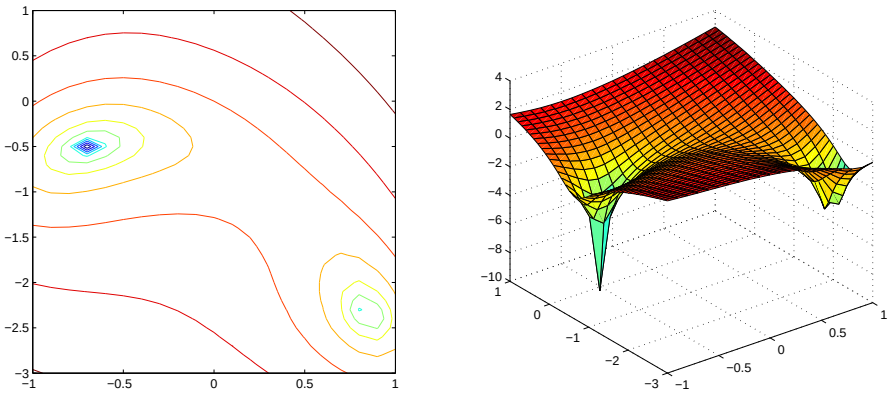
hat Minimalwert 0 für genau jene Wertepaare $(x;y)$, die Lösung des folgenden nichtlinearen Gleichungssystems sind:

$$\begin{aligned} x^2 + \sin y &= 0 \\ e^x + y &= 0 \end{aligned}$$

Aufgabe 1

8 Punkte

Zeichnen Sie für $\log f$, den Logarithmus der gegebenen Funktion, im Bereich $-1 < x < 1$ und $-3 < y < 1$ eine `contour`- und eine `surf`-Grafik in der Art der Abbildung. (In der logarithmischen Darstellung erkennt man die Lage des Minimums deutlicher als in direkter Darstellung von f.) Verwenden Sie ≈ 20 Gitterpunkte in x- und ≈ 40 Gitterpunkte in y-Richtung.



Aufgabe 2

12 Punkte

Lösen Sie das nichtlineare Gleichungssystem mit dem Newton-Verfahren. Finden Sie beide Lösungen im Bereich $-1 < x < 1$ und $-3 < y < 1$. Passende Startwerte können Sie aus der grafischen Darstellung entnehmen. Abbruchkriterium: Aufeinanderfolgende Iterationen unterscheiden sich, gemessen in der 1-Norm, weniger als 10^{-8}.

Muster-Angabe B

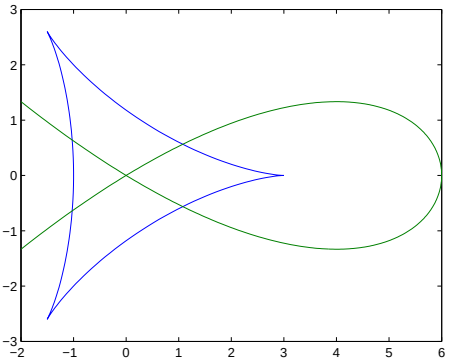
Aufgabe 1

8Punkte

Zwei Kurven in Parameterdarstellung sind gegeben, eine *dreispitzige Hypozykloide* und und *Tschirnhausens Kubische*.

Kurve 1	Kurve 2
$x = 2\cos(s) + \cos(2s)$	$x = 2(3 - t^2)$
$y = 2\sin(s) - \sin(2s)$	$y = 2t(1 - \frac{1}{3}t^2)$

Stellen Sie die beiden Kurven für $s \in [0, \quad 2\pi]$ und $t \in [-2, \quad 2]$ in der xy-Ebene dar und bestimmen Sie graphisch die Lage der beiden Schnittpunkte in der oberen Halbebene.



Aufgabe 2

12 Punkte

Die Schnittpunkte der beiden Kurven von Aufgabe 1 sind auch als Lösung des folgenden Gleichungssystems bestimmt.

$$(x^2 + y^2)^2 + 18(x^2 + y^2) - 27 = 8(x^3 - 3xy^2)$$
$$x^3 = 6\left(x^2 - 3y^2\right)$$

Finden Sie die Lösungen des nichtlinearen Gleichungssystems in der oberen Halbebene mit der MATLAB-Funktion `fsolve`. Passende Startwerte können Sie aus der grafischen Darstellung entnehmen.

Machen Sie die Probe: Setzen Sie die von `fsolve` berechneten Werte in die Funktion ein. Idealer Weise sollte das Ergebnis der Nullvektor sein. Geben Sie die 2-Norm des Restvektors an.

Muster-Angabe C

Aufgabe 1

12 Punkte

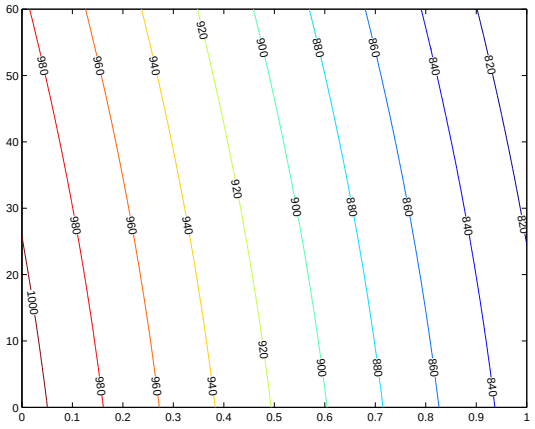
Für die Dichte ρ eines Alkohol-Wasser-Gemisches in Abhängigkeit von Temperatur T (in Celsius) und Alkoholgehalt c (Massenanteil) liegen folgende Tabellenwerte vor:

T	20	40	60	80	20	40	60	80
c	0.1	0.1	0.1	0.1	0.2	0.2	0.2	0.2
rho	981.8	974.8	965.2	954.6	968.6	958.5	945.8	931.3

Finden Sie für diese Daten eine Anpassung der Form

$$\rho(T, c) = a_1 + a_2 T + a_3 T^2 + a_4 c$$

und zeichnen Sie ein Diagramm in der Art der nebenstehenden Abbildung, (x-Achse: Konzentration $0 \leq c \leq 1$, y-Achse: Temperatur $0 \leq T \leq 60$,)



Antwort Koeffizienten a_1, a_2, a_3, a_4 :

Aufgabe 2

8 Punkte

Lineare Gleichungssysteme: Lösbarkeit, Lösungsmenge

Gegeben ist ein lineares Gleichungssystem $Ax = b$ mit Matrix A und rechter Seite b.

$$A = \begin{bmatrix} 1 & -3 & 5 & 1 & -1 \\ 3 & -8 & 17 & 5 & -1 \\ 2 & -4 & 15 & 9 & 5 \\ 7 & -21 & 37 & 14 & 3 \\ 9 & -29 & 44 & 18 & 13 \end{bmatrix}, \quad b = \begin{bmatrix} 15 \\ 59 \\ 70 \\ 129 \\ 142 \end{bmatrix}.$$

Verwenden Sie entsprechende MATLAB-Befehle und geben Sie im Skript als Kommentarzeilen die Anworten auf folgende Fragen

- Ist das System eindeutig lösbar? Falls ja, geben Sie den Lösungsvektor an.
- Ist das System nicht lösbar? Falls ja, geben Sie die kleinste-Quadrate-Näherungslösung (die „am wenigsten falsche Lösung“) an.
- Ist das System mehrdeutig lösbar? Geben Sie einen Lösungsvektor mit möglichst kleiner 2-Norm an.

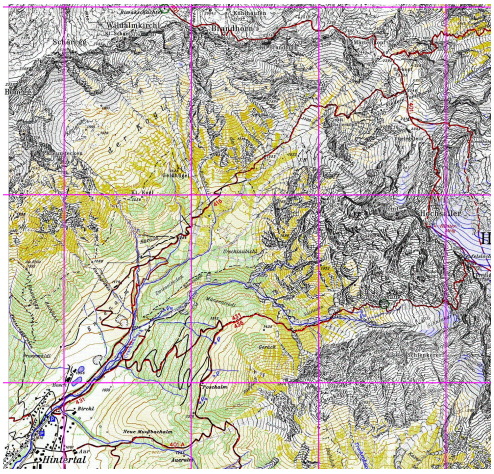
Aufgabe 44: Lineares Datenmodelle anpassen: Georeferenzierung

Das ist noch eine Aufgabe zum Thema „Lineare Datenmodelle“ zur Vorbereitung auf den ersten Kenntnisnachweis. Eine Musterlösung dazu steht zur Verfügung.

Laden Sie dazu hier (klick!) oder von der Übungs-Homepage die Daten und ein Musterprogramm herunter. Die MATLAB-Befehle

```
imdata = imread('HohTs.jpg');
image(imdata)
```

lesen eine Bilddatei (Ausschnitt aus der Alpenvereinskarte Hochkönig, Hagengebirge 1:25000) ein und stellen sie im *figure*-Fenster dar.



In diese Bilddatei sollen GPS-Daten (geogafische Länge und Breite) eingezeichnet werden. Dazu brauchen Sie eine Zuordnung von Pixel-Koordinaten im Bild zu geographischen Koordinaten.

Die Zuordnung

$$\text{Bild-Koordinaten} \longleftrightarrow \text{Welt-Koordinaten}$$

heißt *Bildregistrierung*. Sie verwendet *Kontrollpunkte*: Pixel-Punkte im Bild mit bekannten Welt-Koordinaten, hier geogr. Länge λ und Breite ϕ. Daten für 9 Kontrollpunkte sind im Musterprogramm enthalten.

Finden Sie eine Anpassung der Form

$$x = a_0 + a_1\lambda + a_2\phi$$
$$y = b_0 + b_2\lambda + b_2\phi$$

Pixel-Koord.		Welt-Koord.	
x	y	λ	ϕ
369	2299	12.983	47.417
1143	2299	13.000	47.417
1917	2299	13.017	47.417
⋮	⋮	⋮	⋮

Wie groß ist die maximale Abweichung der Bild-Koordinaten x, y zwischen Modell und Originaldaten? (Idealer Weise sollte die Abweichung nicht mehr als 1–2 Pixel betragen. Die Welt-Koordinaten wurden aber hier um eine Dezimalstelle zu viel gerundet.)

Der MATLAB-Befehl `load trackpnts.mat;` lädt eine lange Liste mit $\lambda\phi$-Koordinaten eines GPS-Tracks. Rechnen Sie diese Koordinaten in Pixel-xy-Koordinaten um und zeichnen Sie diese.

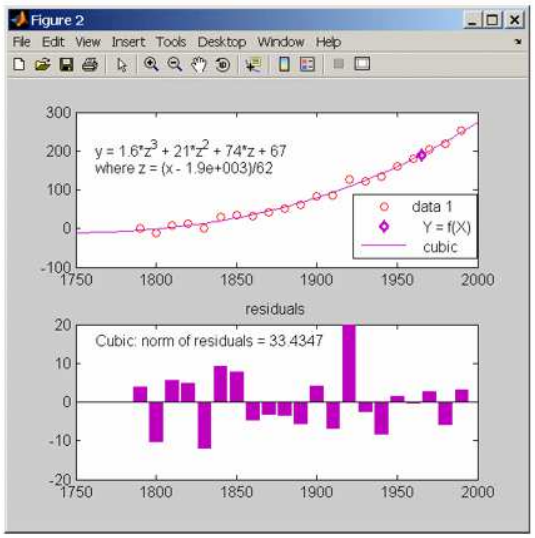
Ü 5.4 MATLAB-Werkzeuge zum Anpassen von Funktionen an Daten

(Bonus-Material für Interessierte: Im Vergleich zu älteren Versionen bietet MATLAB inzwischen tolle neue Werkzeuge. Schaut euch 's an, es zahlt sich aus!)

Zur MATLAB-Grundausstattung gehört das *Basic Fitting Tool*. Damit sollten Sie unbedingt umgehen können. Machen Sie sich gleich einmal damit vertraut: Finden Sie die entsprechende Anleitung in der MATLAB-Hilfe (Vers. 2022b) unter MATLAB › Data Import and Analysis › Descriptive Statistics › Interactive Fitting oder (geht schneller) suchen Sie direkt nach dem Stichwort *Interactive Fitting*.

Sie finden dort ein durchgearbeitetes Beispiel, das die Verwendung des *Basic Fitting Tools* für interaktives Anpassen von Kurven an Datenpunkte erklärt.

Arbeiten das Beispiel in der MATLAB-Hilfe durch und stellen Sie die Approximation graphisch dar, etwa so wie in der nebenstehenden Abbildung.



Darüber hinaus bietet MATLAB in der *Statistics and Machine Learning Toolbox* und in der *Curve Fitting Toolbox* viele Werkzeuge zur Datenanpassung, weit mehr als wir in diesen Übungen behandeln können. Die Aufgaben 45 und 47 zeigen beispielhaft, welche Möglichkeiten die Befehle `cftool` und `polytool` bieten.

Aufgabe 45: Basic Fitting Tool im Vergleich zu Curve Fitting Toolbox

Hier lösen Sie das nichtlineare Ausgleichsproblem aus Aufgabe 42 mit MATLAB-Toolboxen.

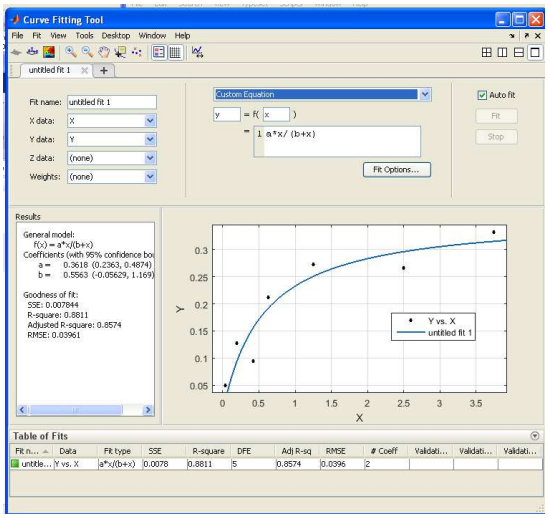
Legen Sie mit MATLABs *basic fitting tool* eine Ausgleichsgerade, ein quadratisches und dann ein kubisches Ausgleichspolynom durch folgende Datenpunkte und plotten Sie die Ergebnisse.

```
X = [0.038 0.194 0.425  0.626  1.253  2.500  3.740];
Y = [0.05  0.127 0.094 0.2122 0.2729 0.2665 0.3317];
```

Es handelt sich hier um die Daten aus Aufgabe 42; dort finden Sie auch eine Abbildung mit gut angepasster Kurve. Welche anderen Optionen des *basic fitting tool* liefern plausible Kurven? Was bedeuten die Fehlermeldungen, die Sie ab Grad 7 bekommen?

Es stellt sich heraus: Kein Ausgleichspolynom kann die Datenpunkte befriedigend approximieren. Das liegt teils daran, dass die Datenpunkte aufgrund der Messunsicherheit weit gestreut sind, aber auch daran, dass sich im gemessenen Experiment die y-Werte mit zunehmendem x asymptotisch einem konstanten Wert nähern. Kein Polynom kann so ein asymptotisches Verhalten beschreiben.

Der Befehl `cftool` öffnet die *Curve Fitting App*. Hier können Sie neben Polynomen auch weitere Modellfunktionen anpassen. Versuchen Sie es: Wählen Sie links oben unter *X Data* und *Y Data* die entsprechenden Datenvektoren aus den im Workspace vorhandenen Variablen. Oben Mitte können Sie den Funktionstyp wählen. Probieren Sie zuerst *Polynomial* mit verschiedenen Polynom-Graden. Wählen Sie anschließend *Custom Equation* und geben Sie den Funktionstern `a*x/(b+x)` ein. (Das ist die Modellfunktion aus Aufgabe 42!) Ihre Anpassung sollte so aussehen wie hier gezeigt.

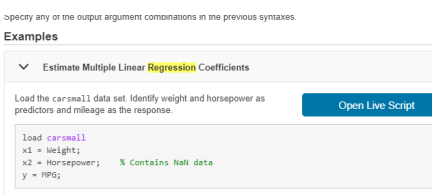


Zusätzlich zu den berechneten Werten der Koeffizienten a und b liefert MATLAB auch ein Konfidenzintervall. Interpretation: Falls den Daten tatsächlich ein Modell der Form $y = ax/(b + x)$ zugrunde liegt, aber die y-Daten zufallsbedingt verrauscht sind, dann berechnet MATLAB *Schätzungen* $\hat{a}$ und $\hat{b}$ für die Parameter a beziehungsweise b. Zum Beispiel:

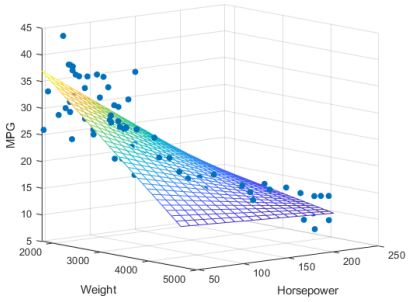
```
Coefficients (with 95% confidence bounds):
       a =      0.3618  (0.2363, 0.4874)
       b =      0.5563  (-0.05629, 1.169)
```

Die Interpretation „Die tatsächlichen Werte von a und b liegen mit 95%-iger Sicherheit im berechneten Intervall“ ist so nicht korrekt; die tatsächlichen Werte von a und b liegen entweder drinnen oder nicht – da ist kein Spielraum für Wahrscheinlichkeiten. Es ist umgekehrt: die berechneten Grenzen des Konfidenzintervalls sind unsicher. Jede unabhängige Wiederholung der Messung liefert (für die immer gleichen Modellparameter a und b) einen neuen Satz von Datenpunkten mit anderen zufallsbedingten Fehlern. MATLABs Schätz-Methode berechnet dazu Konfidenzintervalle mit jeweils etwas anderen Grenzen. Meistens (in 95% der Fälle) schätzt Matlab die Fehlergrenzen richtig, aber in 5% der Fälle überdeckt das berechnete Konfidenzintervall die wahren Werte nicht.

Aufgabe 46: Der Befehl *regress* aus der Statistics Toolbox



Die Matlab-Hilfe Zum Stichwort „regress“ zeigt ein Beispiel mit Daten zu Gewicht, Motorleistung und Benzinverbrauch von Autos und findet dazu eine Anpassungs-Funktion. Vergleichen Sie mit Aufgabe 39: dort wird ein Datenmodell derselben Art berechnet, die Form der Matrix ist gleich.



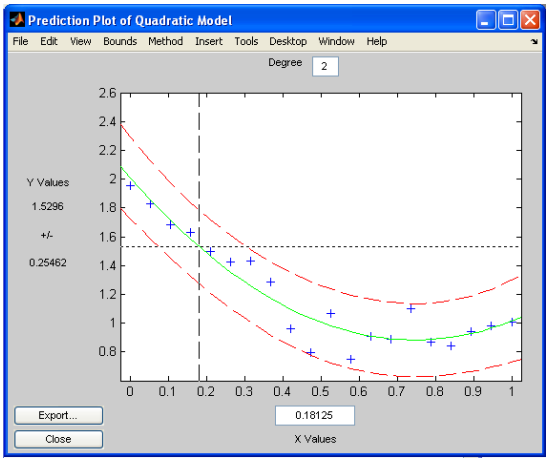
Sie können in der MATLAB-Hilfe auf *Open Live Script* klicken und gelangen so in den *Live Editor*, der mehr Möglichkeiten bietet als der Standard-Editor. Die Dateiendung `*.mls` kennzeichnet Live-Skript-Dateien.
Wenn Sie neugierig sind, probieren Sie den Live-Editor aus. Wenn Sie in der gewohnten Arbeitsumgebung bleiben wollen, speichern Sie dieses Musterbeispiel mittels *save as* als Dateityp *MATLAB Code Files*, Dateiendung `*.m`

Orientieren Sie sich an diesem Muster und erstellen Sie für die Daten aus Aufgabe 39 (Durchmesser, Ausgangstemperatur, Kochzeit) mit dem `regress`-Befehl eine Anpassung und eine ähnliche Graphik wie oben (mit Kochzeit als z-Achse).

Aufgabe 47: Der Befehl *polytool* aus der Statistics toolbox

Die folgenden MATLAB-Befehlszeilen erzeugen entlang der Funktion $y = 2 - 3x + 2x^2$ Datenpunkte, die durch zufällige, normalverteilte Störungen verrauscht sind.

```
n=20;
x=linspace(0,1,n);
y=2 - 3*x + 2*x.^2 + 0.1*randn(1,n);
```



Der Befehl `polytool(x,y)` öffnet ein Fenster ähnlich dem basic fitting tool. Erzeugen Sie damit ein Bild wie die nebenstehende Abbildung und erklären Sie:

- Was bedeuten die verschiedenen Kurven?
- Wozu dient das verschiebbare Achsenkreuz, was sind die angezeigten X-Values und Y-Values?
- Wenn Sie auf Export… klicken, was bedeuten „Parameters“ und Parameters CI“?

Ü 6 Sechste Übungseinheit

Inhalt der sechsten Übungseinheit:

- Polynomiale Interpolation
- Numerische Integration

Ü 6.1 Polynomiale Interpolation

Informieren Sie sich zu diesem Themenbereich: Folien der 6. Vorlesung und Kapitel 7 im Skriptum. Werfen Sie ruhig auch einen Blick auf den Wikipedia-Artikel zum Stichwort Polynominterpolation.

Häufig sind Werte einer Funktion tabellarisch gegeben, wie im folgenden Datensatz:

T	cp	
300	537.0	Spezifische Wärmekapazität von kohlenstoffarmem Stahl in J/kg K für Temperaturen zwischen 300 und 600C
400	593.3	
500	666.8	
600	760.8	

Tabelle 1: Musterdatensatz

Gesucht ist ein Polynom durch diese Datenpunkte. Mögliche Vorgangsweisen:

- Ansatz des Polynoms mit unbestimmten Koeffizienten: $p(x) = a + bx + cx^2 + dx^3$, Einsetzen der Datenpunkte, Aufstellen und Lösen des Gleichungssystems. Siehe Aufgabe 48.
- Lagrangesche Interpolationsformel. Kennen Sie aus der Mathematik-I-Vorlesung. Auch hier im Skriptum behandelt. Vorteil: Polynom lässt sich direkt hinschreiben; Änderung der y-Daten leicht möglich. Nachteil: nicht die günstigste Form zur rechnergestützten Auswertung.
- Das Skriptum erklärt zwei weitere Rechenschemen (Verfahren von Neville und Newton, dividierte Differenzen). Diese Methoden sind für das Rechnen von Hand optimiert und stammen aus der Zeit, bevor es moderne Computer gab. Rechnen von Hand ist heute nicht mehr so wichtig.
- Interpolation in der Newton-Form ist auch am Computer vorteilhaft, wenn es – etwa bei Echtzeit-Anwendungen – auf möglichst kurze Rechenzeit ankommt. Diese Form benötigt bei geschickter Programmierung, siehe Aufgabe 51, im Vergleich zu anderen Verfahren die wenigsten Rechenoperationen.

Ü 6.1.1 Ansatz mit verschiedenen Polynomtypen: Standard-Ansatz, Polynome in Newton-Form, diskrete Orthogonalpolynome

Aufgabe 48: Interpolation: Ansatz mit der Vandermonde-Matrix

Bestimmen Sie die Koeffizienten a, b, c, d des kubischen Interpolationspolynoms zum Datensatz aus Tabelle 1 mit dem Standard-Ansatz $p(x) = a + bx + cx^2 + dx^3$.

Überlegen Sie: Einsetzen der Datenpunkte führt auf ein Gleichungssystem mit der Matrix

$$\begin{bmatrix} 1 & 300 & 300^2 & 300^3 \\ 1 & 400 & 400^2 & 400^3 \\ 1 & 500 & 500^2 & 500^3 \\ 1 & 600 & 600^2 & 600^3 \end{bmatrix}.$$

Eine Matrix, deren Zeilen der Reihe nach die Potenzen von x-Werten enthalten, heißt *Vandermonde-Matrix*. Die gute Nachricht: Falls alle x-Werte verschieden sind, ist das Gleichungssystem eindeutig lösbar.
Die schlechte Nachricht: Diese Matrix kann sehr hohe Konditionszahl haben. Bei Polynomen höheren Grades entstehen gravierende Rundungsfehler. MATLAB's Befehl `polyfit` verwendet diesen Ansatz (mit all seinen Vor- und Nachteilen).

Aufgabe 49: Interpolation: Ansatz mit Polynomen in Newton-Form

Diese Aufgabe sollten Sie mit Stift auf Papier durcharbeiten. Sie erklärt Ihnen die wichtigen Eigenschaften des Newtonschen Interpolationsverfahrens.

Sie können das kubische Polynom auch in folgender Form ansetzen:

$$p(x) = a_0 + a_1(x - 300) + a_2(x - 300)(x - 400) + a_3(x - 300)(x - 400)(x - 500)$$

Setzen Sie hier die Wertepaare des (T, c_p)-Datensatzes aus Tabelle 1 ein. Schreiben Sie die zugehörige Koeffizientenmatrix an. Hier ist schon einmal ein Anfang gemacht:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 100 & 0 & 0 \\ & & \ddots & \\ 1 & 200 & & \\ 1 & 300 & \dots & \dots \end{bmatrix}.$$

Es treten hier praktischer Weise viele Nullen auf. Wie nennt man die spezielle Form einer solchen Matrix?

Bestimmen Sie die Koeffizienten a_0 bis a_3. Warum ist (mit Stift auf Papier) die Lösung des Systems wesentlich einfacher als die eines Vandermonde-Systems?

Der Rechenaufwand zur Lösung eines $n \times n$-Vandermonde-Gleichungssystems wächst mit $O(n^3)$. Mit welcher Potenz wächst der Rechenaufwand bei n Datenpunkten beim Newton-Ansatz?

Lassen Sie nun den letzten Term $a_3(x - 300)(x - 400)(x - 500)$ weg und betrachten Sie das quadratische Polynom

$$p(x) = a_0 + a_1(x - 300) + a_2(x - 300)(x - 400)$$

Ist dieses Polynom auch eine Art von Interpolationspolynom? Welche Daten lassen sich damit beschreiben?

Und wenn Sie auch noch den a_2-Term weglassen? Was beschreibt

$$p(x) = a_0 + a_1(x - 300) \quad ?$$

Sie wollen nun ein zusätzliches Wertepaar in die Interpolation einbeziehen:

T	cp
200	496.4

Fügen Sie dieses Wertepaar *am unteren Ende* der Datentabelle an und ergänzen Sie den Newton-Ansatz um einen weiteren Term

$$a_4(x-300)(x-400)(x-500)(x-600)$$

Was verändert sich an der Matrix des Gleichungssystems? was verändert sich am Ergebnis, also am Wert der Koeffizienten $a_0, a_1, \ldots$?

> Die einzelnen Terme im Newton-Interpolationspolynom beschreiben der Reihe nach lineare, quadratische, kubische… Interpolation zwischen zwei, drei, vier… Wertepaaren.
>
> Jedes zusätzliches Wertepaar ergänzt das Newton-Interpolationspolynom um einen weiteren Term; die anderen Terme bleiben unverändert.

Vergleichen Sie dazu den Polynom-Ansatz in Standard-Form $p(x) = a_0 + a_1x + a_2x^2 + \ldots$, wie ihn Aufgabe 48 verwendet. Da hätten das lineare, quadratischs, kubische… Interpolationspolynom jeweils völlig andere Koeffizienten.

Aufgabe 50: Interpolation: Ansatz mit diskreten Orthogonal-Polynomen

Auch bei dieser Aufgabe sollten Sie die Rechnungen mit Stift auf Papier durchführen. Sie sollen hier sehen: es gibt neben dem Lagrange- und dem Newton-Ansatz noch andere polynomiale Basisfunktionen.

Als dritten Ansatz wählen Sie nun

$$p(x) = a_0p_0 + a_1p_1 + a_2p_2 + a_3p_3$$

mit Ansatz-Polynomen

$$\begin{aligned}
p_0(x) &= 1,\\
p_1(x) &= \frac{1}{50}(x-450),\\
p_2(x) &= \frac{1}{10\,000}\big[(x-450)^2 - 12500\big],\\
p_3(x) &= \frac{1}{300\,000}(x-450)\big[(x-450)^2 - 20500\big]
\end{aligned}$$

Und wie kommt man auf diese Polynome? Die Regeln zum Konstruieren der Polynome $p_0, p_1, \ldots$ sind nicht besonders kompliziert, aber für den Lerneffekt in dieser Aufgabe nicht so wichtig. Nehmen Sie die Polynome als gegeben hin. Die Aufgabe will Sie dafür sensibilisieren:

- Neben der Standardbasis $1, x, x^2, x^3 \ldots$ gibt es noch andere, und oft günstigere Basis-Polynome, um den Raum aller Polynomfunktionen darzustellen.
- Orthogonalität ist eine wichtige Eigenschaft von Vektoren; diese Eigenschaft lässt sich auf Polynome übertragen.

Aufgabe 53 verwendet die Musterprogramme `orth_polyfit.m` und `orth_polyval.m` zur Konstruktion solcher Polynome, aber auch dort wird nicht näher darauf eingegangen, wie das funktioniert. Was Sie aber in den Musterprogrammen erkennen können: es sind wirklich nur wenige Code-Zeilen zur Konstruktion dieser Polynome nötig, nicht mehr als für Newton- oder Vandermonde-Ansatz.

Setzen Sie die Wertepaare des (T, c_p)-Datensatzes aus Tabelle 1 ein. Schreiben Sie die zugehörige Koeffizientenmatrix A an. Sie sieht harmlos und uninteressant aus:

$$A = \begin{bmatrix} 1 & -3 & 1 & \dots \\ 1 & -1 & & \\ 1 & & \ddots & \\ \vdots & & & \end{bmatrix}.$$

Sie hat aber eine wichtige Eigenschaft: $A^T \cdot A$ ist eine Diagonalmatrix. Anders gesagt: die Spaltenvektoren von A stehen aufeinander orthogonal. Damit erfüllt A fast die Definition einer Orthogonalmatrix – Welche Eigenschaft fehlt?

Die quasi-Orthogonalität von A vereinfacht die Lösung des Gleichungssystems für die Koeffizienten: Stellen Sie A auf; berechnen Sie $A^T \cdot A$ und $A^T\mathbf{b}$. Welcher einfache Schritt fehlt noch zur Lösung?

Die Ansatz-Polynome p_0, p_1, p_2, p_3 sind hier so speziell gewählt, dass ihre Werte für den x-Datenvektor (das sind genau die Spaltenvektoren in der A-Matrix) aufeinander orthogonal stehen. Man nennt Polynome mit so einer Eigenschaft *Orthogonalpolynome*. Speziell handelt es sich hier um *diskrete Orthogonalpolynome*, weil sich die Orthogonalität auf die diskreten Datenpunkte auf der x-Achse bezieht.

Falls Sie jetzt fragen: gibt es auch Polynome oder Funktionen, die nicht nur für einige diskrete x-Werte, sondern für alle x in einem bestimmtenm Bereich orthogonal sind? Ja, tatsächlich! Der Begriff „Orthogonalität“ lässt sich von Vektoren auf Funktionen übertragen.

Die wichtige Eigenschaft bei einem Ansatz mit Orthogonalpolynomem ist: Wenn Sie hier den letzten Term weglassen, erhalten Sie ein quadratisches Polynom. Anders als beim Newton-Ansatz interpoliert es zwar keine Datenpunkte mehr, aber es ist das beste Approximationspolynom (im kleinsten-Quadrate-Sinn) an die Daten.

Entsprechend erhalten Sie, wenn sie auch noch den vorletzten Term weglassen, mit $p(x) = a_0 p_0(x) + a_1 p_1(x)$ das beste lineare Polynom (die Ausgleichsgerade) an die Daten.

> Die einzelnen Terme in einem Ansatz mit Orthogonalpolynomen beschreiben der Reihe nach die bestmögliche lineare, quadratische, kubische… Approximation der Wertepaare. Letzlich, bei gleich viel Termen wie Wertepaaren, ergibt sich das Interpolationspolynom.

Aufgabe 51: Interpolation: Newtons Dividierte-Differenzen-Schema

Diese Aufgabe richtet sich an programmier-affine Leute, die wissen wollen, wie man Interpolationsverfahren möglichst rechengünstig implementiert. Solange Sie MATLAB als Rechenumgebung zur Verfügung haben, müssen Sie sich damit nicht befassen. Aber Sie lernen Sie hier clevere Programmier-Tricks, die sich anderswo einsetzen lassen.

Speziell geht es hier um effiziente Berechnung von Tabellen, wobei Zwischenergebnisse, sobald sie nicht mehr gebraucht werden, überschrieben werden, um Speicherplatz zu sparen.

Seite 66 im Skriptum zeigt eine Dreiecks-Tabelle für das kubische Interpolationspolynom nach Newton.

Bei vielen numerischen Verfahren sind Tabellen in Dreiecksform auszufüllen. Nicht alle Einträge werden zum Schluss gebraucht. Deswegen lässt sich der Rechengang geschickt so organisieren, dass immer nur eine Spalte der Tabelle mit gerade benötigten Werten gespeichert ist. Die neu berechneten Einträge überschreiben dann nicht mehr notwendige Zwischenergebnisse. Zum Schluss bleibt ein Vektor mit den benötigten Endergebnissen übrig.

Das folgende Code-Segment berechnet aus Datenvektoren x und y den Vektor dd als obere Schrägzeile des Dividierte-Differenzen-Schemas:

```
dd = y;
for i=2:n
    for k=n:-1:i
        dd(k) = (dd(k)-dd(k-1))/(x(k)-x(k-i+1));
    end
end
```

Implementieren Sie diesen Code als Funktion `dd=newtonpoly(x,y))` und überzeugen Sie sich für das durchgerechnete Beispiel im Skriptum (Seite 66) oder für die Daten der vorigen Aufgabe an einem Durchlauf im Debug-Modus, dass alle Zwischenresultate in dd berechnet werden und erst dann überschrieben werden, wenn sie für den weiteren Rechengang nicht mehr notwendig sind.

Die rechengünstige Auswertung eines Interpolationspolynoms in der Newton-Form an der Stelle z, wenn die Koeffizienten aus dem Dividierte-Differenzen-Schema auf einem Vektor dd und die x-Stützstellen auf x gespeichert sind, leistet eine Schleife der Form

```
y = dd(n);
for i=n-1:-1:1
    y = y.*(z-x(i)) + dd(i);
end
```

Implementieren und testen Sie eine Funktion `y = newtonval(x, dd, z)`, die für gegebene x-Datenpunkte und Dividierte-Differenzen-Koeffizienten dd das Polynom an der Stelle z auswertet.

Vergleichen Sie Anwendung der Funktionen `newtonpoly` und `newtonval` mit den auch MATLAB-Befehlen `polyfit` und `polyval`. MATLAB verwendet dabei den Vandermonde-Ansatz; für die Daten aus dem Skriptum sollte sich kein Unterschied ergeben.

Ü 6.1.2 Güte der Interpolation. Warnung vor hohem Grad. Runges Phänomen

Aufgabe 52: Runges Phänomen, Tschebyschoffs Stützstellen

Früher, in der präcomputerischen Ära, kam niemand leichtfertig auf die Idee, Interpolationspolynome zehnten oder zwanzigsten Grades zu verwenden. Heute reicht ein simpler MATLAB-Befehl, um Polynome hundertsten Grades zu verwenden. Aber, nur weil 's leicht geht, ist es auch vernünftig? Diese Aufgabe untersucht die Frage.

Berechnen Sie Interpolationspolynome für die Funktion

$$y = \frac{1}{1+x^2}, \qquad -5 \le x \le 5$$

an 5, 10, 15 und 20 äquidistanten Stützstellen. Am einfachsten verwenden Sie Matlabs `polyfit, polyval` oder den Code aus Aufgabe 51. Zeichnen Sie Funktion und Polynome. Interpretieren Sie die Graphik, experimentieren Sie auch noch mit höheren Polynomgraden.

Machen Sie sich in den Vorlesungs-Folien und in der Wikipedia zum Stichwort „Runge-Phänomen“ schlau[34] und beantworten Sie dann: Gilt bei polynomialer Interpolation automatisch: „je höher der Grad, desto besser die Näherung?“

Der Approximationssatz von Weierstraß besagt, dass sich jede stetige Funktion auf einem abgeschlossenen Intervall beliebig genau durch Polynome approximieren lässt. Warum werden dann in diesem Beispiel die Approximationen immer schlechter? Hat sich Weierstraß geirrt? – Natürlich nicht, Weierstraß war ja Mathematiker. Das Problem in diesem Fall sind die äquidistanten Stützstellen. Die meisten Funktionen (abgesehen von den völlig harmlosen sin, cos, exp-Funktionen) lassen sich nur extrem widerwillig durch äquidistante Stützstellen zwingen. Es ist natürlich möglich, aber die Funktion reagiert trotzig und weicht zwischen den Stützstellen mächtig vom vorgesehenen Verlauf ab. Siehe Runge-Phänomen

Besonders gegen den Rand des x-Bereiches hin kommt es zu heftigen Oszillationen. Deswegen ist es besser, zum Rand hin die x-Werte dichter zu legen. Besonders günstig im Intervall $[-5;\ 5]$ ist die Anordnung der Stützstellen gemäß der Formel

$$x_i = 5\cos\left(\frac{2i-1}{2n}\pi\right), \quad i = 1, \ldots, n \qquad \text{(Tschebyschow-Stützstellen).}$$

Interpolieren Sie die Funktion wieder mit 5, 10, 15 und 20 Stützstellen, die aber nicht äquidistant verteilt sind, sondern nach der obigen Formel. Experimentieren Sie auch noch mit höheren Polynomgraden und beantworten Sie dann: Gilt bei polynomialer Interpolation vielleicht doch: „je höher der Grad, desto bessere Näherung?“

Aufgabe 53: Approximation, sehr hoher Polynomgrad

Auch wenn bisher betont wurde, dass Polynome hohen Grades nur mit großer Vorsicht zu verwenden sind, gibt es Anwendungen (z.b. Bildverarbeitung, Daten entrauschen. . .), wo Approximationen mit hohem Grad sinnvoll sind. Hier untersuchen wir an einem einfachen Datensatz, welche numerischen Verfahren mit hohem Grad zurecht kommen. Die Datei `SatPressH2O.m` (Download von Übungs-Homepage) enthält reale Daten zum Sättigungsdampfdruck $p(T)$ von Wasser in Abhängigkeit von der Temperatur T.

Angenommen, sie wollen für diese Daten ein hochgenaues Approximationspolynom finden[35].

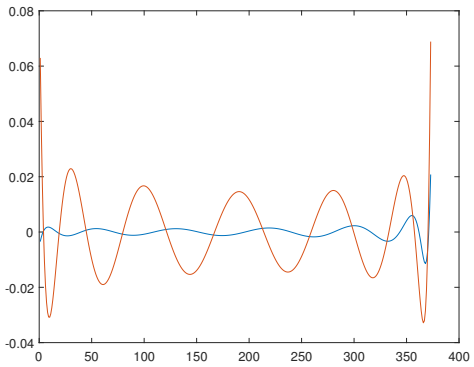
Vergleichen Sie drei Verfahren:

1. Klassischer Ansatz mit Vandermonde-Matrix und Aufstellen der Normalengleichungen
2. QR-Zerlegung der Vandermonde-Matrix – das machen die Matlab-Befehle `polyfit, polyval`
3. Ansatz mit diskreten Orthogonalpolynomen. Dazu gibt es Dateien `orth_polyfit.m` und `orth_polyval.m` zum Herunterladen; diese Dateien funktionieren analog zu `polyfit, polyval`, verwenden aber diskrete Orthogonalpolynome.

[34] Zur Aussprache: Carl Runge (1856–1927) war ein deutscher Mathematiker, sein Name reimt sich auf „Junge“, nicht auf den Musikstil „Grunge“.

[35] Für professionellen Einsatz sollte man die Daten noch geeignet umformen und skalieren; es macht auch praktisch wenig Sinn, Genauigkeit im Mikrobar- oder Nanocelsius-Bereich zu verlangen; niemand kann Temperaturen oder Drücke im praktischen Einsatz mit solcher Genauigkeit messen. Aber es geht ums Prinzip; die Ergebnisse sind typisch auch für andere Funktionen oder Datensätze.

Beginnen Sie mit Approximation durch ein Polynom 10-ten Grades und stellen Sie in einem Plot den Fehler in den Datenpunkten dar (Datenwert - Polynomwert). Erhöhen Sie schrittweise den Polynomgrad und stellen Sie fest, ob und wie weit der Fehler abnimmt. Je nach Verfahren lassen sich nur bis zu einem gewissen Grad sinnvoll Approximationen finden. Bis zu welchem Grad lässt sich die Methode der Normalengleichungen, die QR-Zerlegung, die Orthogonalpolynome numerisch stabil einsetzen?



Hier ist ein Beispiel-Plot: Er zeigt für Polynomgrad 12 in blau den Fehler des Approximationspolynoms aus der QR-Zerlegung (Matlab-Befehle `polyfit, polyval`), in orange den entsprechenden Fehler bei der Methode der Normalengleichungen. Er ist signifikant größer; das kommt nur von den Rundungsfehlern. Bei Grad 11 gibt es noch kaum Unterschied. Normalengleichungen können Sie also bei diesem Datensatz ab Grad 12 vergessen. Wie sieht es mit den anderen beiden Methoden bei höheren Graden aus?

Ü 6.2 Numerische Integration

Siehe die Folien der 6. und 7. Vorlesung. Im Skriptum unter Kapitel 8.

Das Romberg-Verfahren (Aufgabe56) ist nicht nur als genaues Verfahren zur numerischen Integration interessant, sondern illustriert ein allgemeines Prinzip: Berechne einen Näherungswert mehrmals mit unterschiedlicher Schrittweite $h_1, h_2, \ldots$, extrapoliere von diesen Daten auf $h = 0$. Diese allgemeine Idee heißt *Richardson-Extrapolation*. Ihre Anwendung auf die zusammengesetzte Trapezregel ist das Romberg-Verfahren.

Ü 6.2.1 Integration von Tabellenwerten

Wenn zu einer Funktion nur diskrete Datenpunkte vorliegen, kann daraus ihr Integral näherungsweise mit verschiedenen numerischen Quadraturformeln berechnet werden.

Hier sind nochmals die Tabellenwerte der spezifische Wärmekapazität von kohlenstoffarmem Stahl gegeben.

T	cp
0	460.8
100	471.1
200	496.4
300	537.0
400	593.3
500	666.8
600	760.8

Spezifische Wärmekapazität von kohlenstoffarmem Stahl in J/kg K für Temperaturen zwischen 0 und 600C

Aufgabe 54: Integral der spezifischen Wärmekapazität

Das Integral der spezifischen Wärmekapazität ist die Enthalpie. Berechnen Sie für die gegebenen Daten das Integral von $T = 0$ bis $T = 600$

- mit der Simpson-Regel (3 Stützstellen $T = 0, 300, 600$)
- mit der 3/8-Regel (4 Stützstellen $T = 0, 200, 400, 600$)
- mit der zusammengesetzten Trapezregel ($h = 100$)
- mit der zusammengesetzten Simpson-Regel ($h = 100$)

Hinweis: In MATLAB lässt sich die Multiplikation von Funktionswerten mit Gewichtsfaktoren elegant als Skalarprodukt zweier Vektoren schreiben. Am Beispiel der 3/8-Regel:

```
b=600;
a=0;
f= [460.8   496.4   593.3   760.8];
w=[1 3 3 1]/8;
int38=(b-a)*f*w'
```

Ü 6.2.2 Numerische Integration von Funktionen

Wenn eine Funktion nicht in geschlossener Form integrierbar ist, oder der Rechenausdruck zu kompliziert wird, ist numerische Integration (auch: *numerische Quadratur*) möglich. Der MATLAB-Befehl `q = quad(fun,a,b)` berechnet das Integral von f in den Grenzen von a nach b mit einer adaptiven Simpson-Regel. Dabei kann f *inline* gegeben sein,

```
>> q=quad('1./x',1,2)
q =
    0.6931
>>
```

Achtung, punktweise Vektor-Rechenoperationen `./`, `.^` usw. verwenden

auch als anonymous function oder als Funktions-Henkel, genauso wie beim Aufruf von `fzero()`.

Aufgabe 55: Die Agnesi-Kurve

Am 16. Mai jährt sich zum 305. mal der Geburtstag von Maria Gaetana Agnesi, einer altösterreichischen Mathematikerin, die Kurven vom Typ

$$y = \frac{a^3}{x^2 + a^2}$$

untersucht hat. Ihr zu Ehren sollen Sie diese Kurve numerisch integrieren. Berechnen Sie numerisch das Integral

$$\int_{-1}^{1} \frac{1}{x^2+1} dx \qquad \text{(exakter Wert: } \frac{\pi}{2}\text{)}$$

(a) mit der zusammengesetzten Trapezregel und drei verschiedenen Schrittweiten: $h = \frac{4}{10}, \frac{2}{10}, \frac{1}{10}$. Vergleichen Sie mit dem exakten Wert $\frac{\pi}{2}$ und geben Sie an: Bei halber Schrittweite reduziert sich der Fehler (annähernd) um einen Faktor ...

(b) mit MATLABs `quad`-Befehl

Aufgabe 56: Die Agnesi-Kurve mit Romberg-Integration

Romberg-Verfahren: Nennen Sie die drei Ergebnissen von Punkt a der vorigen Aufgabe (Trapezregel mit jeweils halbierter Schrittweite) $T_{1,1}, T_{1,2}, T_{1,3}$, Berechnen Sie ein Dreiecks-Schema der Form

$$\begin{array}{lll} T_{1,1} & & \\ & T_{2,2} & \\ T_{1,2} & & T_{3,3} \\ & T_{2,3} & \\ T_{1,3} & & \end{array}$$

nach der Vorschrift

$$T_{2,2} = \frac{4T_{1,2} - T_{1,1}}{3} \quad , \quad T_{2,3} = \frac{4T_{1,3} - T_{1,2}}{3} \text{ und } , \quad T_{3,3} = \frac{16T_{2,3} - T_{2,2}}{15}$$

Die Resultate werden zunehmend genauer. Vergleichen Sie mit dem exakten Resultat!

Und jetzt der allgemeine Fall (Bonus-Aufgabe): die Formel des Romberg-Verfahrens bei k Ergebnissen $T_{1,1}, T_{1,2}, \ldots T_{1,k}$ aus der Trapezregel mit jeweils halbierter Schrittweite lautet:

$$T_{i+1,j} = \frac{4^i T_{i,j} - T_{i,j-1}}{4^i - 1}$$

Dabei bezieht sich der Index i auf die Spalten $1, 2, \ldots k$ des Dreiecksschemas, Index j bezeichnet die Aufwärts-Schrägzeilen.

$$\begin{array}{llllll} T_{1,1} & & & & & \\ & T_{2,2} & & & & \\ T_{1,2} & & T_{3,3} & & & \\ & T_{2,3} & \vdots & \ldots & T_{k,k} & \\ T_{1,3} & \vdots & T_{3,k} & & & \\ \vdots & T_{2,k} & & & & \\ T_{1,k} & & & & & \end{array}$$

Sie können dieses Dreiecksschema Speicherplatz sparend auf ähnliche Weise berechnen, wie die dividierten Differenzen des Newton-Schemas in Aufgabe 51 – wenn Sie das schaffen, wäre das dann noch mal ein Extra-Bonus!

Die $T_{i,j}$ Werte nähern mit zunehmender Genauigkeit; wenn sich Werte nicht mehr ändern, (innerhalb einer Fehlerschranke), kann man davon ausgehen, ein hinreichend genaues Resultat erreicht zu haben. Adaptive Quadraturformeln arbeiten im Wesentlichen nach diesem Prinzip.

Ü 6.3 Analytische Integration von Funktionen

Bonus-Material für Interessierte. MATLAB kann symbolisch nicht nur differenzieren, sondern auch integrieren. Das ist gut zu wissen, und wir laden Sie herzlich ein, den Abschnitt durchzuarbeiten.

Während es zum Differenzieren fixe Regeln gibt, die schrittweise zum Ergebnis führen, ist analytische (man sagt auch: symbolische) Integration wesentlich komplizierter. Nur für recht einfache Funktionen gibt es Integrationsregeln; darüber hinaus findet man eine Sammlung von Tricks und „Kochrezepten“. Oft läßt sich nicht von vornherein absehen, ob eine gegebene Funktion eine Stammfunktion hat und welche Methode ein Ergebnis liefern wird.

Früher gab es umfangreiche Integraltafeln zum Nachschlagen, jetzt übernehmen Computeralgebrasysteme diese Aufgabe. Die sind inzwischen sehr leistungsfähig und können für viele Funktionstypen Integrale berechnen. Der zuständige Befehl in MATLABs *symbolic math toolbox* lautet `int( )`.

Zum Aufwärmen:

```
>> syms x
```

Definieren Sie x als symbolische Variable

Damit können Sie Ihre Mathematik-Kenntnisse auffrischen: Wie lautet $\int x^2\,dx$?

```
>> int(x^2)
ans =
1/3*x^3
```

Gibt MATLAB die richtige Antwort? Lassen Sie sich dazu eine Geschichte erzählen...

> Zwei Mathematiker sitzen beim Bier und klagen über das allgemein recht spärliche mathematische Grundlagenwissen. Dann muss der erste Mathematiker aufs Klo, das nützt der zweite aus und ruft die Kellnerin. Er erklärt ihr, in ein paar Minuten, wenn sein Freund wieder zurück sei, werde er sie nochmals zum Tisch bitten und etwas fragen. Sie müsse einfach nur antworten: „Ein Drittel x der Dritten". Sie fragt nach: „Ein Tritt – Elixta tritt ihn??" Er wiederholt geduldig: „Ein Drittel x der Dritten". Darauf sie: „Ein Driddelix, der tritt 'n?" „Ja, genau so", seufzt er. Gut, sie ist einverstanden und murmelt nun immerfort in sich hinein: „Ein Driddelix, der tritt 'n..."
>
> Der erste Mathematiker kommt zurück und der zweite schlägt vor: „Was wetten wir, die meisten Leute haben sehr wohl etwas Ahnung von Mathematik. Ich frage die blonde Kellnerin dort nach einem Integral". Lachend nimmt der erste die Wette an. Der zweite ruft also die Kellnerin und fragt: „Was ist das Integral von x Quadrat?" Die Kellnerin sagt: „Ein Drittel x der Dritten", dreht sich um und ruft im Weggehen über die Schulter hin noch zurück, „plus C!".

Versuchen Sie weitere Integrale:

```
>> syms x n t
```

Definieren Sie noch einige symbolische Variable

```
>> int(x^n)
ans =
piecewise([n = -1, log(x)], [n <> -1, x^(n + 1)/(n + 1)])
>>
```

Das sieht kompliziert aus, ist aber korrekterweise MATLABS Interpretation der Regel

$$\int x^n\,dx = \begin{cases} \log x & (n=-1) \\ \frac{x^{n+1}}{n+1} & (n\neq -1) \end{cases} + C$$

```
>> int(sin(t))
ans =
-cos(t)
>>
```

Auch die gängien Funktionen lassen sich integrieren...

```
>> int(exp(-x^2))
ans =
(pi^(1/2)*erf(x))/2
>>
```

...obwohl es schnell kompliziert werden kann. Hier ist ein Integral nicht durch elementare Funktionen darstellbar. Es gibt aber die spezielle Funktion erf (die eigens für Integrale dieses Typs erfunden wurde).

```
>> int(cos(n*t))
ans =
1/n*sin(n*t)
>>
```

Hat MATLAB hier nach n oder nach t integriert?

MATLAB versucht (ähnlich wie beim Differenzieren) die Integrationsvariable zu erraten. Es nimmt an, Buchstaben am Ende des Alphabets seien Variable, Buchstaben weiter vorne eher Parameter und Konstante. Deswegen interpretiert es `int(cos(n*t))` als $\int \cos(nt)\,dt$ und nicht als $\int \cos(nt)\,dn$. Im Zweifelsfall akzeptiert der `int()` Befehl ein zweites Argument:

```
>> int(log(x)*cos(t),x)
ans =
x*cos(t)*(log(x) - 1)
>>
```

Das ist $\int \log x \cos t\,dx$

```
>>int(log(x)*cos(t),t)
ans =
log(x)*sin(t)
>>
```

Das ist $\int \log x \cos t\,dt$

Weitere Möglichkeiten: `int(S,a,b)` gibt das bestimmte Integral von S in den Grenzen von a bis b, `int(S,v,a,b)` legt zusätzlich die Integrationsvariable v fest.

Aufgabe 57: Symbolische Integration

Berechnen Sie symbolisch in MATLAB

$$\int \frac{1}{1+x^2}\,dx \qquad \int \sqrt{1-ax^2}\,dx$$

$$\int e^{-tx^2}\,dt \qquad \int_1^2 \frac{1}{x}\,dx$$

Ü 7 Siebte Übungseinheit

Inhalt der siebten Übungseinheit:

- Eigenwertaufgaben
  - Anschauliche Erklärung: `eigshow`
  - Vektoriteration, QR-Iteration
  - Einige Anwendungen
- Einschrittverfahren für gewöhnliche Differentialgleichungen erster Ordnung
  - MATLAB-Löser
  - Klassisches Euler-Verfahren und andere einfache explizite Verfahren

Hinweis: Die Unterlagen verweisen auf Seiten im Vorlesungsskriptum. Seitenzahlen beziehen sich auf die aktualisierten Kapitel, die über die Homepage des Lehrstuhls (klick!) abrufbar sind.

Ü 7.1 Eigenwerte und Eigenvektoren

Was Sie an theoretischen Grundlagen für diese Einheit brauchen, fasst Abschnitt 9.1 (klick!) im aktuellen Skriptum zusammen. Siehe dazu auch die Folien der 8. Vorlesung (klick!). Informieren Sie sich dort über die Definitionen und grundlegenden Eigenschaften! Die Übungsaufgaben stellen den Zusammenhang zu typischen Anwendungen her.

MATLAB-Befehle:

`d = eig(A)` liefert Vektor von Eigenwerten

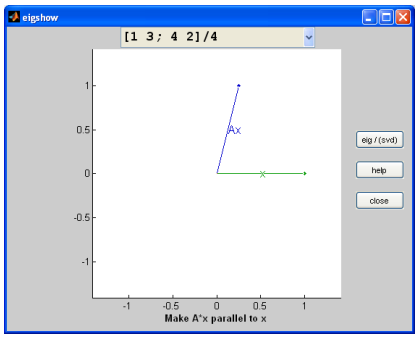
`[V,D] = eig(A)` Spalten von V sind Eigenvektoren, Diagonalelemente von D sind Eigenwerte.

Ein interaktives, recht lehrreiches Beispiel zu Eigenvektoren liefert das MATLAB-Demoprogramm `eigshow`. Laden Sie von der Übungs-Homepage (klick!) die Unterlagen zur 7. Übung herunter und starten Sie `eigshow.m`. Tipp: Unter `http://blogs.mathworks.com/cleve/2013/07/08/eigshow-week-1` erklärt Ihnen Cleve Moler, einer der MathWorks-Gründer, persönlich, worum es in eigshow geht.

Bewegen Sie den Vektor $\mathbf{x}$ und beobachten Sie, wie sich $A\mathbf{x}$ entsprechend ändert. Drehen Sie den Vektor $\mathbf{x}$ so, dass er zu $A\mathbf{x}$ parallel liegt. Jeder solche Vektor $\mathbf{x}$ ist ein *Eigenvektor* der Matrix A.
Das Ergebnis $A\mathbf{x}$ ist in diesem Fall ein Vielfaches der Ausgangsvektors, es gilt also die Gleichung

$$A\mathbf{x} = \lambda\mathbf{x} \ .$$

Der Proportionalitätsfaktor λ ist der zu $\mathbf{x}$ gehörende *Eigenwert* von A.



Die Matrix A können Sie im Fenster oben aus einer Liste wählen. Voreingestellt ist

$$A = \frac{1}{4}\begin{bmatrix} 1 & 3 \\ 4 & 2 \end{bmatrix} \ .$$

- Finden Sie für die obige Matrix A Eigenvektoren. Lesen Sie die Komponenten von $\mathbf{x}$ ab und schätzen Sie λ. Wie viele verschiedene Werte von λ gibt es hier?
- Vergleichen Sie mit dem Resultat des Befehls `[V,D] = eig(A)`.
- Versuchen Sie andere Matrizen aus der Liste. Nicht immer können Sie Eigenvektoren finden. Es gibt Beispiele, wo in *gar keiner* Richtung $\mathbf{x}$ und $\mathbf{Ax}$ auf einer Linie liegen. Was liefert `[V,D] = eig(A)` in solchen Fällen?
- Im Normalfall (bei regulärer Matrix A) beschreibt $A\mathbf{x}$ eine Ellipse, wenn Sie $\mathbf{x}$ bewegen. Singuläre Matrizen bilden die Ausnahme. Sie haben $\lambda = 0$ als Eigenwert. Wie sieht die Bahn von $A\mathbf{x}$ dann aus? Finden Sie eine Beispielmatrix aus der Liste.

Aufgabe 58: Spannungstensor, Hauptachsentransformation

Diese Aufgabe testet, ob Sie die MATLAB-Befehle aufrufen und deren Ergebnisse interpretieren können.

Reine Druck- oder Zugkräfte wirken normal auf ein Flächenelement, reine Scherkräfte parallel dazu. In einem elastisch deformierten Körper kann die Kraft auf ein Flächenelement irgendwie schräg wirken. Der Spannungstensor stellt den Zusammenhang zwischen Flächennormalrichtung und Kraftrichtung her:

Kraftvektor = Spannungstensor mal Normalenvektor des Flächenelements

In einem geeignet gedrehten Koordinatensystem nimmt der Spannungstensor Diagonalgestalt an. Die Achsen dieses Koordinatensystems sind die *Hauptachsen* des Spannungstensors. Sie zeigen in Richtung seiner Eigenvektoren. Der diagonalisierte Spannungstensor enthält die Eigenwerte in der Hauptdiagonalen.

Der Spannungstensor in einem Material sei (in willkürlichen Einheiten)

$$p = \begin{bmatrix} 5 & 10 & -8 \\ 10 & 2 & 2 \\ -8 & 2 & 11 \end{bmatrix}.$$

a) Welcher Kraftvektor wirkt auf ein Flächenelement parallel zur xy-Ebene? — zur Ebene $x + y + z = 0$? (Dazu brauchen Sie noch keine Eigenwerte.)

b) In welche Richtungen zeigen die Hauptachsen des Spannungstensors (`format rat` liefert „schöne“ Zahlen)? Wie lautet der auf Diagonalgestalt transformierte Tensor?

c) In zwei Richtungen wirkt reine Zug-, in einer reine Druckspannung (Vorzeichenkonvention bei den Diagonaltermen des Spannungstensors: Druck ist negativ[36]. In welcher Richtung wirkt
— die größere Zugspannung?
— die kleinere Zugspannung?
— die Druckspannung?

[36] wie auch sonst im Leben: Prüfungsdruck, Leistungsdruck, Erfolgsdruck,...

Aufgabe 59: Vektoriteration

Was passiert, wenn man einen Vektor wieder und wieder mit derselben Matrix multipliziert? Testen Sie für den Vektor $x = (1, 0, 0)^T$ und die Matrix A

$$A = \begin{bmatrix} 8 & 1 & 6 \\ 3 & 5 & 7 \\ 4 & 9 & 2 \end{bmatrix}$$

Wenn man den Befehl `x = A*x` zehnmal iteriert, werden die Zahlen in x schon recht groß.

Dividieren Sie nun zusätzlich in jedem Schritt den Vektor x nach der Multiplikation mit A durch seine erste Komponente. Das ändert nicht die Richtung von x, hält aber die Zahlenwerte der Komponenten in übersichtlichen Grenzen. Sie können das in der Form `>> y=A*x; x = y/y(1)` im Workspace durchspielen. Was beobachten Sie? Zu welchem Wert konvergiert y_1?

Iterieren Sie, beginnend mit dem Startvektor $x = (1, 0, 0)^T$,

```
>> y=A*x; x = y/y(1)
```

Zu welchem Wert konvergiert y_1?

Vergleichen Sie: Für diese Matrix berechnet das Skriptum (klick!) auf Seite 78 die Eigenwerte aus dem charakteristischen Polynom.

Im Skriptum auf Seite 80 ist Vektoriteration (in etwas allgemeinerer Fassung) als Pseudocode gegeben. Dabei ist noch berücksichtigt, dass die einfache Skalierung `x = y/y(1)` nicht funktioniert, falls $y(1) = 0$.

Die Beispielmatrix $A = \begin{bmatrix} -1 & 0 \\ 1 & 2 \end{bmatrix}$ im Skriptum auf Seite 76 ist so ein Fall, wo Skalierung mit der ersten Komponente nicht zielführend ist. Rechnen Sie wie oben einige Schritte, dann sehen Sie, warum.

Ein Computerprogramm skaliert am besten mit jener Komponente, die größten Absolutbetrag hat. Der MATLAB-Befehl `[m,i]=max(abs(x))` findet im Vektor x den Index i dieser Komponente.
Eine simple MATLAB-Implementierung dazu wäre

```
for k=1:100
    x1 = A*x0;
    [m,i]=max(abs(x1));
    lam = x1(i);

    x1 = x1/lam;
    if(norm(lam - lam0)<eps)
        break
    end
    x0=x1;
    lam0=lam;
end
```

Konkrete Aufgabe Schreiben Sie ein Skript, das eine 10×10-Matrix A mit zufällig gewählten ganzzahligen Elementen $a_{ij} \in \{0, \ldots, 9\}$ erzeugt[37]. Berechnen Sie mit Vektoriteration den betragsgrößten Eigenwert und einen zugehörigen Eigenvektor. Wählen Sie ein Abbruchkriterium mit Genauigkeit $\epsilon < 10^{-6}$.

Lassen Sie auch MATLABs `eig`-Befehl Eigenwert und Eigenvektor berechnen und vergleichen Sie die Ergebnisse. Beachten Sie, dass MATLAB Eigenvektoren als Einheitsvektoren in der 2-Norm, während die oben gezeigte Vektoriteration in der Unendlichnorm auf 1 skaliert. Zum Vergleich der Eigenvektoren müssen Sie MATLABs Eigenvektor gleich skalieren wie den aus der Vektoriteration berechneten!

Aufgabe 60: Ein Top-10-Algorithmus: Eigenwerte aus QR-Iteration

Eigenwertprobleme für 2×2- oder 3×3-Matrizen sind Ihnen bei der Hauptachsentransformation des Spannungstensors (vergleiche Aufgabe 58) begegnet. Darüber hinaus gab es bis etwa 1920 aus Sicht der technisch-wissenschaftlichen Anwendungen keine besondere Motivation, Eigenwerte größerer Matrizen zu berechnen. Aber dann nahm die Wichtigkeit von Eigenwertaufgaben dramatisch zu: es kamen Quantentheorie, Matrizenmechanik, Finite-Elemente-Methoden und, ganz aktuell, Data Science.

Der klassische Lösungsweg über das charakteristische Polynom ist unter anderem deswegen für größere Matrizen ungeeignet, weil die Nullstellen extrem empfindlich gegenüber kleinen Störungen in den Polynomkoeffizienten sein können (vergleiche Übungsbeispiel 13).

Um 1960 entstand der QR-Algorithmus, der es in die Top-10 Algorithmen von größtem Einfluss auf Wissenschaft und Technik im 20. Jahrhundert geschafft hat[38][39]. Was die Experten dabei auch begeistert, ist: es handelt sich um einen tatsächlich völlig neuen Lösungsansatz und nicht bloß (wie uns hier schon häufig begegnet) eine verfeinerte Ausarbeitung der Ideen von Gauss oder Newton.

Sie haben die QR-Zerlegung einer Matrix bereits bei der Lösung überbestimmter Gleichungssysteme kennengelernt. (Vergleiche Folien zur 5. Vorlesung). Für die rechnerische Ausführung haben die Unterlagen bis jetzt immer nur auf MATLABs Befehl `[Q R]=qr(A)` verwiesen. Aber die QR-Zerlegung lässt sich in wenigen Zeilen implementieren. Auch das soll in dieser Aufgabe angesprochen werden. (Vergleiche Übungsaufgabe 32, sie befasst sich mit den Codezeilen für LR-Zerlegung).

Bemerkung: Die Matrix R in der Zerlegung $A = Q \cdot R$ ist **nicht** dieselbe wie in der Zerlegung $A = L \cdot R$. (Die englischsprachigen Literatur schreibt die LR-Zerlegung als $A = L \cdot U$; dort kann diese Verwirrung gar nicht erst entstehen.)

Diese Aufgabe lässt Sie mit der Basis-Version des QR-Algorithmus experimentieren. In seiner einfachsten Form zerlegt er eine Matrix in das Produkt einer orthogonalen und einer rechten oberen Dreiecksmatrix, $A = Q \cdot R$. Anschließend multipliziert er die Faktoren in umgekehrter Reihenfolge. Diese beiden Schritte werden iteriert.

---

[37] Wir verwenden Zufallsmatrizen hier nur als Testprobleme für den Vektoriterations-Algorithmus. Das ist bei weitem nicht die interessanteste Anwendung. In einem zufälligen Gespräch in Princeton 1972 entdeckten der Mathematiker Hugh Montgomery und der Physiker Freeman Dyson eine überraschende Verbindung zwischen Nullstellen der Riemannschen Zetafunktion, Eigenwerten von Zufallsmatrizen und quantenphysikalischen Systemen *(Montgomery's pair correlation conjecture)*

[38] J. Dongarra and F. Sullivan, "Guest Editors Introduction to the top 10 algorithms" in Computing in Science & Engineering, vol. 2, no. 01, pp. 22-23, 2000.

[39] B. Parlett, "The QR Algorithmïn Computing in Science & Engineering, vol. 2, no. 01, pp. 38-42, 2000.

Konkrete Aufgabe Schreiben Sie ein Skript, das eine *symmetrische*[40] 10×10-Matrix A mit zufällig gewählten ganzzahligen Elementen $a_{ij} \in \{0, \ldots, 9\}$ erzeugt; eine einfache Möglichkeit:

```
A = randi(5,10)-1;
A=A+A';
```

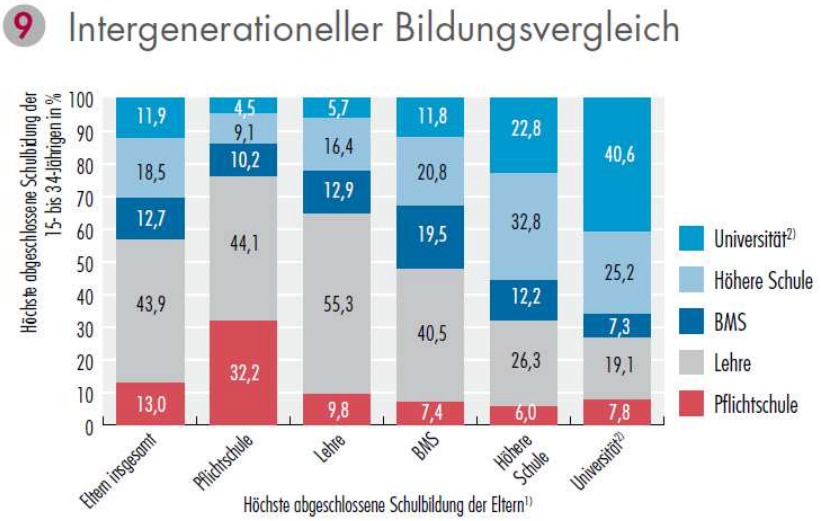
Lassen Sie Matlab die Eigenwerte von A bestimmen; merken Sie sich den Ergebnis-Vektor. Nun wiederholen Sie 100 mal die Schritte

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

- Sehen Sie sich bei mehreren Versuchen mit unterschiedlichen Zufalls-Matrizen jeweils die Struktur der Ergebnis-Matrix an: In welchen Positionen treten Werte näherungsweise gleich Null auf? Wo immer, wo meistens?
- Vergleichen Sie die Diagonalelemente der Ergebnis-Matrix mit den Eigenwerten der Original-Matrix. Was können Sie erkennen?
- Ersetzen Sie MATLABs `qr`-Befehl durch den Aufruf von `myQR` (Im Material zur 7. Übung). Das ist eine einfache Implementierung der QR-Zerlegung. Prüfen Sie nach: An den Aussagen zu den zwei vorherigen Punkten sollte sich dadurch nichts ändern.

Aufgabe 61: Bildungsmobilität zwischen den Generationen

Es hängt stark von der sozialen Herkunft ab, welche Ausbildung Kinder und Jugendliche erhalten. Die Abbildung[41] zeigt Daten aus Österreich.



Q: STATISTIK AUSTRIA, Mikrozens.-Arbeitskräfteerhebung Ad-hoc-Modul „Eintritt junger Menschen in den Arbeitsmarkt" – 2. Quart. 2009. Bevölkerung in Privathaushalten (15- bis 34-Jährige). – Pers. nicht in Ausbild. – 1) Höchste abgeschl. Schulbildung d. Eltern: Haben Mutter und Vater nicht denselben Ausbildungsabschluss, wird jeweils die höhere Ausbildung verwendet. – 2) Inkl. hochschulverwandte Lehranstalten.

[40] Der QR-Algorithmus kann auch unsymmetrische Matrizen behandeln; unsere Basisversion funktioniert besserr für symmetrische Matrizen.

[41] Quelle: *Bildung in Zahlen 2010/11, Schlüsselindikatoren und Analysen.* Statistik Austria, Wien, 2012

Erstellen Sie aus diesen Daten eine Matrix $A = [a_{ij}]$, in der das Element a_{ij} angibt:

> Von Kindern, deren Eltern Ausbildung j haben, erreicht der Anteil a_{ij} die Ausbildung i.

Beispiel: Kinder aus Akademikerfamilien ($j = 5$) erreichen zu 19,1% eine Lehre ($i = 2$) als höchste Bildungsstufe; daher $a_{25} = 0{,}191$.

Der Balken ganz links in der Graphik gibt den Ist-Zustand (Bildungsniveau der Jugendlichen insgesamt) an. Setzen Sie diese Daten in einen Vektor $\mathbf{x}^{(1)}$ ein. Angenommen, das Bildungsniveau der Eltern insgesamt ist durch einen Vektor $\mathbf{x}^{(0)}$ beschrieben (diese Daten lassen sich nicht direkt aus der Grafik ablesen).

a) Begründen Sie: Das Bildungsniveau der nächsten Generation insgesamt errechnet sich durch Matrix-Vektor-Multiplikation $\mathbf{x}^{(1)} = A \cdot \mathbf{x}^{(0)}$.

b) Berechnen Sie $\mathbf{x}^{(0)}$ aus den gegebenen Daten für A und $\mathbf{x}^{(1)}$.

c) Berechnen Sie, ausgehend von $\mathbf{x}^{(1)}$, den Zustand nach 1, 2 und 3 weiteren Generationen;

d) Ein *stabiler Zustand* besteht, wenn sich von einer Generation zur nächsten nichts ändert. Begründen Sie: dabei handelt es sich um einen Eigenvektor von A. Berechnen Sie den stabilen Zustand.

Die Berechnung des stabilen Zustandes lässt sich als Vektoriteration (Aufgabe 59) durchführen oder mit MATLABs `eig` Befehl.

Hinweis: Die Prozentangaben in der Grafik sind gerundet, deswegen summieren sich nicht alle Spalten auf exakt 100%, woraus sich kleine Ungenauigkeiten ergeben.

Aufgabe 62: Schwingungsgleichung

Die Eigenschwingungsformen und zugehörigen Frequenzen einer schwingenden Saite lassen sich näherungsweise aus dem Eigenwertproblem

$$A\mathbf{x} = \lambda\mathbf{x}$$

bestimmen. Man denkt sich die Saite dazu in n Einzelmassen, wie Perlen auf einer Schnur, zerlegt. Für grosse n nähert sich die Perlenschnur einer kontinuierlichen Massenverteilung an. Die $n \times n$ Matrix A hat eine einfache Tridiagonal-Form.

$$A = \begin{bmatrix} 2 & -1 & 0 & 0 & \dots & 0 \\ -1 & 2 & -1 & 0 & \dots & 0 \\ 0 & -1 & 2 & -1 & \dots & 0 \\ \vdots & & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & -1 & 2 & -1 \\ 0 & \dots & & 0 & -1 & 2 \end{bmatrix}.$$

Ist die Saite mit Länge ℓ und Gesamtmasse m mit Kraft τ gespannt, dann ist die zum Eigenwert λ dieser Matrix zugehörige Eigenschwingungsfrequenz ν (in Hz) näherungsweise bestimmt durch:

$$\nu = \frac{n+1}{2\pi}\sqrt{\lambda\frac{\tau}{m\ell}}$$

Dem kleinsten Eigenwert von A entsprichtdie Frequenz der Grundschwingung, die weiteren Eigenwerte entsprechen den Obertönen. In der Praxis sind Grundschwingung und einige

Oberschwingungen niedriger Ordnung relevant. Je größer n, desto genauer sind die aus der „Perlenschnur-Näherung“ berechneten Frequenzen.

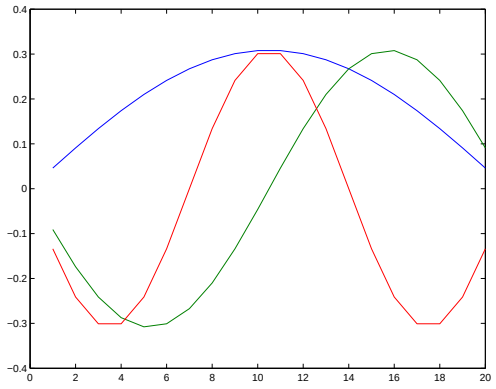
In diesem einfachen Fall lassen sich Eigenfrequenzen und Schwingungsformen auch durch exakte Formeln angeben. Aber schon geringfügig allgemeinere Aufgabenstellungen, zum Beispiel ungleichförmige Massenverteilung entlang der Saite, sind nicht analytisch lösbar. Die Analyse des Schwingungsverhaltens von Bauteilen ist eine typische Anwendung für numerische Simulation.

Siehe dazu auch die Vorlesungsfolie 10 der 8. Vorlesung, sie zeigt ein Perlenschnur-Modell einer frei pendelnden Kette.

Ihre Aufgabe: Für die E-Saite einer Konzertgitarre gilt: freier Länge $\ell = 65\,\mathrm{cm}$, Masse $m = 3{,}58\,\mathrm{g}$, Spannkraft $\tau = 63{,}11\,\mathrm{N}$. Mit diesen Daten lautet die obige Frequenzgleichung

$$\nu = 164{,}7\frac{n+1}{2\pi}\sqrt{\lambda}$$

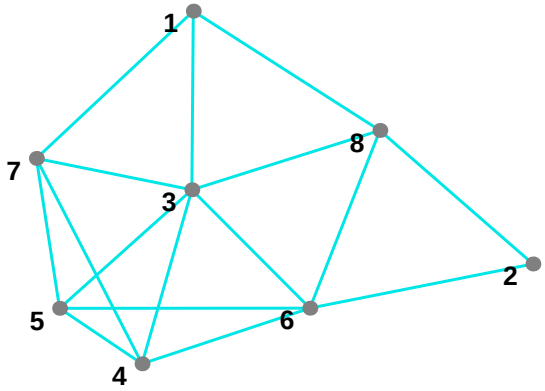
Erstellen Sie für $n = 5, 10, 20$ die Matrix A und berechnen Sie die Frequenzen der Grundschwingung. Sie sollten erkennen: die berechneten Näherungen konvergieren rasch. (Für eine Genauigkeit von zwei Nachkommastellen müssten Sie aber $n = 80$ wählen!)



Zeichnen Sie für $n = 20$ die Eigenvektoren zu den niedrigsten drei Eigenwerten; sie geben die entsprechenden Schwingungsformen der Saite (Grundschwingung, erste und zweite Oberschwingung) an. Siehe Abbildung!

Aufgabe 63: Erreichbarkeit in einem Netzwerk

Ein Netzwerk (Verkehrsverbindungen, verlinkte Seiten im Internet, soziales Netz. . . , mathematisch: ein Graph) lässt sich durch seine Adjazenzmatrix beschreiben.



$$A = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ \vdots & & & & & & & \vdots \end{bmatrix}$$

(a) Stellen Sie zum hier gezeigten Netzwerk die (01)-Adjazenzmatrix A auf (siehe Vorlesungsfolie 8, 8. Vorlesung 2023) und berechnen Sie einen Eigenvektor zum größten Eigenwert. Welcher Knoten im Netz ist (jedenfalls laut Eigenvektor-Daten) am besten, welcher am schlechtesten vernetzt?

(b) Gegeben ist ein Vektor $\mathbf{x}^{(0)}$ aus lauter Einsen. Welches Maß für „Vernetzung“ oder „Erreichbarkeit ist durch $\mathbf{x}^{(1)} = A\mathbf{x}^{(0)}$ definiert?

(c) Führen Sie, beginnend mit dem Vektor $\mathbf{x}^{(0)}$ aus (b), zehn Schritte der Vektoriteration durch. Was erhalten Sie dadurch näherungsweise?

Die Idee, Links zwischen Internet-Seiten auf diese Art mathematisch zu modellieren, untersuchte 1997 ein Doktorand (ein gewisser Larry Page) an der Stanford University. Seine Dissertation hat er zwar bis heute nicht abgeschlossen, aber wenn es Sie interessiert, googeln Sie, was aus ihm und seinem Forschungsprojekt geworden ist…

Die Folien zur 8. Vorlesung zeigen ein ähnliches Netzwerk. Für ein praktisch relevantes Beispiel sehen Sie in Wikipedia zum Stichwort Page Rank nach. Im Material zur 7. Übung finden Sie die Datei `pagerank.m`, sie vollzieht die Rechnungen im Wikipedia-Artikel nach.

Ü 7.2 Gewöhnliche Differentialgleichungen erster Ordnung

Aufgabenstellung:
Explizite gewöhnliche Differentialgleichung 1. Ordnung mit Anfangsbedingung
Gegeben ist eine Funktion $f(x, y)$. Gesucht ist eine Funktion $y(x)$. Sie soll erfüllen

$$y'(x) = f(x, y(x)) \qquad \text{Differentialgleichung}$$
$$y(x_0) = y_0 \qquad \text{Anfangsbedingung}$$

Die Folien der 9. Vorlesung (klick!) zeigen die geometrische Interpretation dieser Aufgabe als Richtungsfeld im $\mathbb{R}^2$ mit Lösungsfunktionen, die dem Richtungsfeld folgen.

In den aktuellen Übungsunterlagen finden Sie die MATLAB-Dateien `BeispieleGDG.m` und `GDGDemo.m`. Laden Sie sich diese Skript-Dateien herunter. Übersichtlich formatiert und gut lesbar werden diese Dateien, wenn Sie MATLABs Publish-Funktion nützen oder die Dateien als Live-Scripts öffnen!

Die Datei `BeispieleGDG.m` zeigt zuerst auch die symbolische Lösung von Differentialgleichungen. Wir konzentrieren uns auf die numerische Lösung, denn brauchbare symbolische Lösungen existieren nur in einfachen Fällen.

Ü 7.2.1 MATLAB-Löser für gewöhnliche Differentialgleichungen

MATLAB bietet fertige Löser für Anfangswertprobleme an.

Es sind sind sieben Solver verfügbar: `ode23`, `ode23s`, `ode23t`, `ode23tb`, `ode45`, `ode113`, `ode15s`. Eine genauere Beschreibung können Sie in der MALAB-Hilfe nachlesen.

Die Funktion `ode45` ist das Standardverfahren, welches man für ein „neues“ Anfangswertproblem zuerst probieren wird. Sie ist die Implementierung eines expliziten Einschritt-Runge-Kutta-Verfahrens (Fehlerordnung 5 und Kontrollrechnung mit Fehlerordnung 4).

Es folgt ein kleines Beispiel, das die Verwendung von `ode45` demonstriert. Angenommen, Sie wollen für $y = y(x)$ das Anfangswertproblem

$$y' = g(x, y) = 2x(1 + y^2), \quad y(0) = 0,$$

im Intervall $[0, 1]$ mit diesem Solver lösen. Sie speichern dafür die Funktion g in einer Datei `g.m` ab

```
function ydot=g(x,y)
ydot=2*x*(1+y^2);
end
```

und fassen dann die Lösung des Anfangswertproblems in einer Scriptdatei (`test.m`) zusammen:

```
y0=0.0;
xspan=[0, 1];
ode45(@g,xspan,y0)
```

So einfach ist das! Sie erhalten eine graphische Darstellung der Lösung. (Noch kürzer ist die Implementierung in der Beispieldatei `BeispieleGDG.m`!)

Sie erhalten Vektoren von x- und y-Werten statt der grafischen Ausgabe, wenn Sie folgende Syntax verwenden:

```
[X, Y] = ode45(@g,xspan,y0);
```

MATLAB wählt sich aber die Lage der x-Werte selbst aus (um die Fehler unterhalb einer bestimmten Schranke zu halten). Wenn Sie die Lösung numerisch für bestimmte, von Ihnen gewählte x-Werte berechnen wollen, schreiben Sie beispielsweise

```
sol=ode45(@g,xspan,y0);
x = [0 0.25 0.5 0.75 1];
y = deval(sol,x)
```

Aufgabe 64:

Berechnen Sie mit MATLAB numerisch für die nachfolgend gegebenen Funktionen $y = y(x)$ die Lösung des Anfangswert-Problems. Stellen Sie die Lösung graphisch dar.

a) $y' = y$ mit $y(0) = 1$, Lösung gesucht im Bereich $0 \le x \le 2$.

Das ist die klassische Differentialgleichung der Exponentialfunktion, exakte Lösung ist $y = e^x$. Stellen Sie auch (in einem separaten Diagramm) den Fehler zwischen numerischer und exakter Lösung dar.

b) $y' = xy/4 - 1$ mit drei verschiedenen Anfangsbedingungen: $y(0) = 2;\ 2,5;\ 3$, Lösungen gesucht im Bereich $0 \le x \le 4$.

Vergleichen Sie: im Skriptum sind Lösungen dieser Gleichung im Richtungsfeld dargestellt.

c) $y' = \dfrac{1}{x^2}$ mit $y(-1) = 1$, Lösungen gesucht im Bereich $-1 \le x \le 1$. Prüfen Sie nach: Die Funktion $y = -\dfrac{1}{x}$ erfüllt in ihrem Definitionsbereich Differentialgleichung und Anfangsbedingung. Bei der numerischen Lösung müssen Sie allerdings auf Probleme gefasst sein. Die Funktion y' hat hier eine spezielle Eigenschaft, die eine Lösung im gesamten Intervall $-1 \le x \le 1$ nicht zulässt – welche Eigenschaft ist das? Vergleichen Sie mit der numerischen Lösung.

Ü 7.2.2 Explizite Einschrittverfahren

Bei der numerischen Lösung einer DG bestimmt man ausgehend von den Anfangsbedingungen eine Folge von Wertepaaren $(x_0, y_0), (x_1, y_1), (x_2, y_2), \ldots$, die den Verlauf der gesuchten Funktion $y = y(x)$ annähern sollen. Schema:

Wähle Schrittweite h und maximale Schrittzahl N;
setze x_0 und y_0 laut Anfangsbedingung;
für $i = 0, 1, \ldots, N-1$
$\quad x_{i+1} = x_i + h$;
$\quad y_{i+1} = y_i + hF(x_i, y_i, h)$.

Die Funktion $F(x, y, h)$ heißt die *Verfahrensfunktion* des jeweiligen Verfahrens. Geometrisch interpretiert gibt $F(x, y, h)$ die *Fortschreit-Richtung*. Die verschiedenen Verfahren unterscheiden sich in der Verfahrensfunktion, also in der Definition der Fortschreit-Richtung. Nur beim expliziten Euler-Verfahren ist die Fortschreit-Richtung im Punkt $(x; y)$ auch gleich der Steigung $y'(x, y)$, also

$$F(x, y, h) = f(x, y),$$

Andere Verfahrensfunktionen versuchen, die Fortschreit-Richtung besser an den Verlauf der Lösung anzupassen. Die Folien der 8. Vorlesung stellen verschiedene Einschrittverfahren und deren Verfahrensfunktionen bildlich dar.

Beim modifizierten Euler-Verfahren ist

$$F(x, y, h) = f\left(x + \frac{h}{2}, y + \frac{h}{2} f(x, y)\right),$$

beim Verfahren von Heun

$$F(x, y, h) = \frac{1}{2}(k_1 + k_2)$$

mit

$$\begin{aligned} k_1 &= f(x, y) \\ k_2 &= f(x + h, y + hf(x, y)). \end{aligned}$$

Dazu gibt es ein Musterprogramm `GDGdemo.m` zum Herunterladen!

Tipp: Lassen Sie sich von Cleve Moler, einer der MathWorks-Gründer, persönlich erklären, wie das Runge-Kutta-Verfahren das Anwachsen einer Flamme berechnet!

Ü 7.2.3 Handrechnung und einfache Programme fürs Verständnis

„Ich habe ein numerisches Verfahren erst dann verstanden, wenn ich mich selbst dabei verrechnet habe“

Die numerische Lösung einer Differentialgleichung durch händische Rechnung dient heutzutage nur mehr der Illustration der Rechenverfahren. Die praktische Durchführung überlassen Sie dem Computer. Ihr Verständnis für die Rechenverfahren müssen Sie aber zumindest bei der Vorlesungsprüfung dadurch demonstrieren, dass Sie einige Rechenschritte selbst auf dem Papier ausarbeiten.

Die Folien der 9. Vorlesung (klick!) zeigen sehr ausführlich die einzelnen Rechenschritte der Verfahren, die sie hier durchrechnen sollen. Das Musterprogramm `GDGdemo.m` implementiert die Rechenschritte in MATLAB.

Ein Beispiel sei das Anfangswertproblem für die Funktion $y(x)$

$$\begin{aligned} y' &= 4xy + 3 \\ y(0) &= 0 \end{aligned}$$

In diesem Fall ist also $f(x, y) = 4xy + 3$ und $x_0 = y_0 = 0$. Wir wollen mit Schrittweite $h = 0{,}2$ den Wert der Funktion $y(x)$ an den Stellen $x_1 = 0{,}2$; $x_2 = 0{,}4$; $x_3 = 0{,}6$ näherungsweise bestimmen. Die exakte Lösung ist bekannt, aber nicht durch elementare Funktionen gegeben; auf sechs Nachkommastellen genau beträgt $y(0{,}6) = 2{,}973\,414$.

Ein tabellarisches Rechenschema ist hilfreich. Beachten Sie: Die vorletzte Spalte berechnet immer die Verfahrensfunktion, die letzte Spalte entspricht dem allgemeinen Schritt $y_{i+1} = y_i + hF(x_i, y_i, h)$, in einfachen Worten: „neuer y-Wert = alter Wert plus h mal Fortschreit-Richtung F“

Für das explizite Euler-Verfahren:

i	x	y	$F = f(x,y) = 4xy + 3$	$y + hF$
0				
1				
2				

Modifiziertes Euler-Verfahren:

i	x	y	$k_1 = f(x,y)$	$y + \frac{h}{2}k_1$	$F = f(x + \frac{h}{2}, y + \frac{h}{2}k_1)$	$y + hF$
0						
1						
2						

Verfahren von Heun:

x	y	$k_1 = f(x,y)$	$y + hk_1$	$k_2 = f(x + h, y + hk_1)$	$F = (k_1 + k_2)/2$	$y + hF$

Aufgabe 65: Einfache Einschrittverfahren programmieren, Fehlerordnung erkennen

Verwenden Sie das Musterprogramm `GDGdemo.m` und berechnen Sie für das oben gegebene Beispiel den Wert $y(0{,}6)$ mit verschiedenen Schrittweiten und verschiedenen Verfahren:

1. Rechnen Sie mit $h = 0{,}2$ und den drei obigen Verfahren. (Vergleichen Sie mit den Werten aus der Handrechnung.) Implementieren Sie auch das dreistufige Verfahren

$$\begin{aligned} k_1 &= f(x,y) \\ k_2 &= f(x + \frac{h}{2}, y + \frac{h}{2}k_1) \\ k_3 &= f(x + h, y - hk_1 + 2hk_2) \\ F &= \frac{1}{6}(k_1 + 4k_2 + k_3) \end{aligned}$$

2. Rechnen Sie nun mit feinerer Schrittweite $h = 0{,}05$. Wie groß ist jeweils der Fehler (Differenz zwischen Näherungswert und exakter Lösung $y(0,6) = 2,973414$?

3. Halbieren Sie die Schrittweite, rechnen Sie also mit $h = 0{,}025$ und vergleichen Sie die Fehler. Um welchen Faktor hat sich jeweils der Fehler reduziert?

Die verscheidenen MATLAB-Löser für Anfangswertprobleme arbeiten im Prinzip wie das Demo-Programm `GDGdemo.m`, aber

- die Verfahrensfunktion bestimmt die Fortschreit-Richtung aus mehreren, optimal gewählten Zwischenpunkten,
- der (globale) Fehler wird abgeschätzt,
- die Schrittweite wird automatisch angepasst.
- für *steife Probleme* gibt es Methoden (`ode23s, ode15s`) mit besserem Stabilitätsverhalten.

Ü 8 Achte Übungseinheit

Inhalt der achten Übungseinheit:

- Optionen in MATLABs `ode45` setzen
- Stabilität, implizite Verfahren
- Systeme von Differentialgleichungen 1. Ordnung

Ü 8.1 Gewöhnliche Differentialgleichungen erster Ordnung (Fortsetzung von Ü 7.2)

Aufgabe 66: Setzen der Fehlerschranken

MATLABs `ode45`-Befehl löst mit den Standard-Einstellungen viele unkomplizierte Aufgaben zufriedenstellend. Manchmal reichen aber die voreigestellten Optionen nicht aus. Sie können umfangreiche Anpassungen durchführen. Diese Aufgabe zeigt Ihnen, wie Sie die Rechengenauigkeit beeinflussen können.

Gegeben sei für $y = y(x)$ das Anfangswertproblem

$$y' = f(x, y)$$

mit

$$f(x, y) = -200xy^2, \quad y(-3) = \frac{1}{901}.$$

Berechnen Sie unter Verwendung des MATLAB-Solvers `ode45` die Lösung für $x = -2, -1, 0$.

Fehlerschranken: Verwenden Sie die Befehle

```
options=odeset('reltol',1.e-8,'abstol',1.e-8);
sol=ode45(@g,xspan,y0,options)
```

um die vorgegebenen Fehlerschranken zu ändern. Versuchen Sie auch Fehlerschranken $10^{-10}, 10^{-12}, 10^{-14}$. Berechnen Sie jeweils den Wert $y(0)$.

Für dieses Problem lässt sich die exakte Lösung leicht angeben:

$$y_{\text{exakt}}(x) = \frac{1}{100x^2 + 1}$$

Vergleichen Sie für $x = 0$ die numerisch berechneten Werte $y(0)$ mit der exakten Lösung $y_{\text{exakt}}(0) = 1$. Berechnen Sie die Fehler (relativer und absoluter Fehler sind hier gleich!). Werden die gewünschten Fehlergrenzen erreicht?

Ü 8.1.1 Stabilität eines Verfahrens; implizite Verfahren

Rechenverfahren sollten den Verlauf der Lösung, wenn schon nicht genau, so doch wenigstens *qualitativ* richtig bestimmen.

Konkret: wenn die exakte Lösung eine exponentiell abklingende Funktion ist, dann soll auch die numerische Lösung abklingen. Leider berechnen bei zu großer Schrittweite die bisher vorgestellten *expliziten* Verfahren exponentiell *anwachsende* Lösungen – ziemlich genau das Gegenteil von dem, was passieren sollte.

Ein Verfahren heißt *stabil* bei Schrittweite h, wenn für das Modellproblem

$$y' = -y \qquad y(0) = 1$$

die numerische Lösung mit wachsendem x nach Null konvergiert.

Sie lernen nun *implizite* Verfahren kennen; diese machen bei jedem einzelnen Rechenschritt ähnlich große Fehler wie explizite Verfahren. Aber bei expliziten Verfahren schaukeln sich Fehler von einem Schritt zum nächsten exponentiell auf; bei impliziten Verfahren klingen die Fehler in Summe ab und zerstören nicht den qualitativen Verlauf der Lösung.

Aufgabe 67: Stabilität expliziter Einschrittverfahren

Modifizieren Sie das Musterprogramm `GDGdemo.m` und testen Sie die bisher behandelten Einschritt-Verfahren (Euler explizit, Euler modifiziert, Heun sowie das dreistufigen Verfahren aus Aufgabe 65) für verschiedene Schrittweiten h beim Anfangswertproblem für $y(x)$

$$\begin{aligned} y' &= -y \\ y(0) &= 1 \end{aligned}$$

Rechnen Sie bis $x = 10$, und finden Sie für jedes Verfahren h-Werte, für die das Verfahren gerade noch abklingende, beziehungsweise bereits leicht anschwellende Lösungen berechnet – das sind also h-Werte gerade noch innerhalb, beziehungsweise knapp außerhalb der Stabilitätsgrenze.

Vergleichen Sie dazu die Abbildungen auf der vorletzten Vorlesungsfolie, 9. Vorlesung!

Aufgabe 68: Implizites Eulerverfahren

Hier ist die Verfahrensfunktion gegeben durch

$$F(x, y, h) = f(x + h, y(x + h))$$

und der neue Näherungswert daher

$$y(x + h) = y(x) + hf(x + h, y(x + h))$$

Anders als beim expliziten Verfahren lässt sich der gesuchte Wert nicht direkt finden: Sie können $y(x + h)$ auf der linken Seite nur berechnen, wenn Sie es schon kennen, weil Sie es auf der rechten Seite der Gleichung einsetzen müssen. Da beißt sich die Katze in den Schwanz...

Mathematisch gesprochen handelt es sich um eine eine *implizite Gleichung* für $y(x + h)$. Sie lässt sich bei unserem Testproblem mit der einfachne Funktion $f(x, y) = -y$ leicht auflösen.

Orientieren sie sich am Musterprogramm `GDGdemo.m` und implementieren Sie das implizite Euler-Verfahren für das Testproblem

$$\begin{aligned} y' &= -y \\ y(0) &= 1 \end{aligned}$$

Untersuchen Sie die Stabilität dieses Verfahrens. Gibt es hier Schrittweiten h, für die die numerische Lösung anschwillt?

Ü 8.2 Systeme von Differentialgleichungen erster Ordnung

Ü 8.2.1 Beispiel: SIR-Modell zur Ausbreitung einer Epidemie

Hintergrundinformation

Das SIR-Modell (*susceptible-infected-removed model*) beschreibt in sehr vereinfachter Form die Ausbreitung von ansteckenden Krankheiten mit Immunitätsbildung.

Das Modell teilt eine Gesamtpopulation in drei Gruppen, S, I und R. Die erste Gruppe, S wie *susceptible* (empfindlich), enthält die anfälligen Individuen: sie sind nicht infiziert, können sich aber jederzeit anstecken. Gruppe I sind die Infizierten. In der dritten Gruppe befinden sich alle, die sich sich selbst und niemand anderen mehr infizieren können. Schön wär's, wenn wir R wie *recovered* sagen könnten, R steht aber für *removed*, weil das Modell Genesene und Verstorbene in dieser Gruppe zusammenfasst.

Die Urheber dieses Modells, der Biochemiker William Ogilvy Kermack und der Militärarzt Anderson Gray McKendrick, konnten trotz der Einfachheit des Modells gut die Daten einer Pestepidemie in Bombay 1905/06 modellieren. Nur etwas über 100 Jahre später, 2014, publizierten Harko und Koautoren eine exakte analytische Lösung dieses Differentialgleichungssystems. In aller Regel werden diese Gleichungen aber numerisch gelöst, und das werden wir hier tun.

In verfeinerter und weiterentwickelter Form haben solche Modellrechnungen in den letzten Jahren unser Alltagsleben mehr als uns lieb sein konnte beeinflusst. Sie können leicht und reichlich vertiefende Informationen finden. Im Moment reicht uns, dass das SIR-Modell als System von drei Differentialgleichungen 1. Ordnung formuliert ist:

$$\begin{aligned}\frac{dS}{dt} &= -\frac{\beta IS}{N}\\ \frac{dI}{dt} &= \frac{\beta IS}{N} - \gamma I\\ \frac{dR}{dt} &= \gamma I\end{aligned}$$

Die Funktionen $S(t), I(t)$ und $R(t)$ beschreiben die Größe der jeweiligen Gruppen in Abhängigkeit von der Zeit t.

Die Gesamtgröße der Population $N = S(t) + I(t) + R(t)$ bleibt darin konstant. Die allgemeine demografische Entwicklung, also Geburten und nicht durch die Epidemie bedingte Todesfälle, wird nicht modelliert. Das ließe sich aber mit wenigen Zusatz-Termen berücksichtigen.

Der Parameter β gibt an, wie viele enge Kontakte ein Individuum pro Tag durchschnittlich hat (wobei nur ein Kontakt zwischen einem S- und einem I-Individuum zu einer Ansteckund führt; Kontakte vom Typ S-S, S-R, I-I und I-R bleiben folgenlos).

Der Parameter γ lässt sich einfacher in Form seines Reziprokwerts interpretieren als

$$\frac{1}{\gamma} \quad \ldots \text{ Typische Zeitdauer der Infektion.}$$

Maßgeblich für die Dynamik der Epidemie ist die Basisreproduktionszahl

$$R_0 = \frac{\beta}{\gamma} \qquad .$$

Dieser Wert gibt an, wie viele weitere Personen eine infizierte Person in der Anfangsphase der Epidemie, (wenn in der allgemeinen Population noch kaum jemand immun ist), im Durchscnitt ansteckt[42]

Numerische Lösung in MATLAB

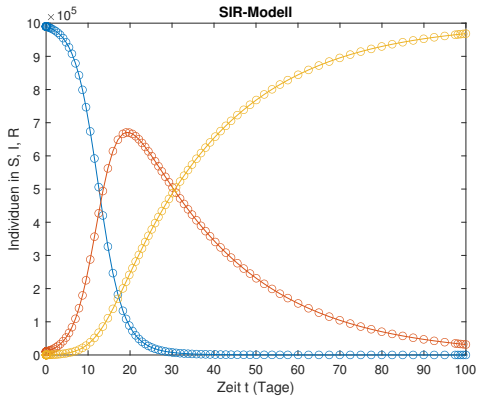
Es gibt dazu ein fertiges Musterprogramm, `SIRmodel.m`, aber die wesentlichen Programmcode-Zeilen sind so kurz, dass wir sie hier dem Differentialgleichungssystem gegenüberstellen:

Gleichungssystem	Implementierung

$\frac{dS}{dt} = -\frac{\beta IS}{N}$
$\frac{dI}{dt} = \frac{\beta IS}{N} - \gamma I$
$\frac{dR}{dt} = \gamma I$
	`f = @(t,y)[`
	`  - beta*y(2)*y(1)/N;`
	`  beta*y(2)*y(1)/N - gamma*y(2);`
	`  gamma*y(2) ];`

$0 \le t \le 100$ $S(0) = 999, \quad I(0) = 1, \quad R(0) = 0.$	`ode45(f,[0 100],[999, 1,0])`

Der Aufruf von `ode45` in seiner einfachsten Form, mit Parametern $\beta = 0{,}4$; $\gamma = 0{,}04$; $N = 10\,000\,000$ liefert die nebenstehende Abbildung.



Aufgabe 69: Aufgabe zum SIR-Modell

Wenden Sie ein SIR-Modell mit plausiblen Parametern auf die Situation in Österreich im Sommer und Herbst 2021 an. Setzen Sie die Start-Zeit $t = 0$ mit 17. Mai fest; damals war $I(0) \approx 10\,000$, $S(0) \approx 6\,400\,000$ (8 Millionen, minus 1 Million Geimpfte und 600 000 Genesene), $R(0) = N - S(0) - I(0)$. Die Wahl $\gamma = 1/14$ entspricht einer durchschnittlicher Dauer der infektiösen Phase von 14 Tagen. Wählen Sie für β einige verschiedene Werte, so dass die

[42] Die aus den Medien bekannte *effektive Reproduktionszahl* R_{eff} liegt unter R_0, im einfachsten Fall ist $R_{\text{eff}} = R_0 S/N$. Sie berücksichtigt bereits bestehende teilweise Immunität und gesenkte Übertragungswahrscheinlichkeit (durch Lockdown, Masken,...)

effektive Reproduktionszahl $R_{\text{eff}} = \beta/\gamma \cdot S/N$ zwischen $\approx 0{,}8$ (das war der Wert im Mai 2021) und 1,4 liegt (diesen Wert erreichte R_{eff} im Jänner 2022).

Zeichnen Sie für verschiedene R_{eff}-Szenarien den Verlauf der I- und R-Größen für die nächsten paar Monate (ohne S, weil es sich sonst mit dem Maßstab in der Abbildung nicht ausgeht: die S-Gruppe ist so groß, dass die I- und R-Anteile dann kaum sichtbar sind.)

Sie können dieses einfache SIR-Modell mit dem tatsächlichen Verlauf vergleichen: Im Mai und Juni 2021 blieb $R_{\text{eff}} < 1$, dem entsprechend sank I in diesem Zeitraum beständig. Von Juli bis November war $R_{\text{eff}} \approx 1{,}2$. In diesem Zeitraum baute sich eine neue Welle auf, die Mitte November mit $I \approx 150\,000$ ihren Höhepunkt erreichte und nur durch einen Lockdown gestoppt werden konnte.

Ü 8.2.2 Selbst programmierte Einschrittverfahren

Die Bewegungsgleichungen für Weg und Geschwindigkeit und ein einfaches Kraftgesetz lauten:

> Zeitableitung des Weges = Geschwindigkeit
> Zeitableitung der Geschwindigkeit = Beschleunigung
> Beschleunigung ist proportional der Kraft
> Rücktreibende Kraft proportioal zum Weg

Wenn die zeitabhängigen Funktionen $y_1(t)$ und $y_2(t)$ jeweils den Weg und die Geschwindigkeit bezeichnen, dann lautet die mathematische Formulierung der obigen Beziehungen im einfachsten Fall

$$\begin{aligned} \dot{y}_1(t) &= y_2(t) \\ \dot{y}_2(t) &= -y_1(t) \end{aligned}$$

Es handelt sich hier um ein *System von zwei Differentialgleichungen erster Ordnung*. Die gute Nachricht: Sowohl die selbst programmierten Einschrittverfahren nach dem Muster des Programmes `GDGdemo.m` als auch die MATLAB-Löser funktionieren – mit wenigen und offensichtlichen Änderungen – auch für Systeme.

Änderung 1 Sie formulieren die rechte Seite des Systems als vektorwertige Funktion. Konkret setzen Sie in `GDGdemo.m`

```
function ystrich = f(x,y)
    ystrich = [ y(2)
                -y(1)];
end
```

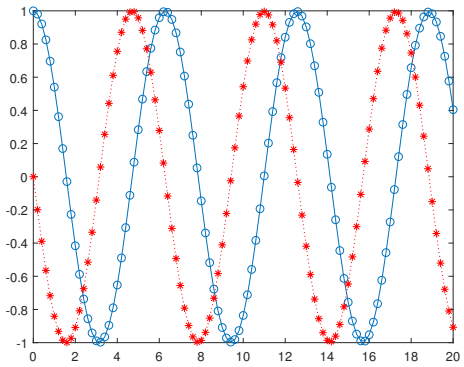
Änderung 2 Sie brauchen, weil es sich um *zwei* Diffgleichungen erster Ordnung handelt, auch zwei Anfangsbedingungen; konkret Anfangsposition $y_1(0) = 1$ und Anfangsgeschwindigkeit $y_2(0) = 0$. Setzen Sie dazu in `GDGdemo.m`

```
%% Setze Anfangswerte, x-Intervall und Schrittweite
x = 0;
y = [1;0]
% Endwert von x und Schrittweite
x_end = 20;
h = 0.2;
```

Änderung 3 Das Ergebnis `y` ist eine zweizeilige Matrix; erste Zeile enthält die berechnete Näherungslösung für den Weg $y_1(t)$, die zweite Zeile die entsprechenden Näherungen an die Geschwindigkeit $y_2(t)$. Sie stellen beide Funktionen in einem Plot dar, zum Beispiel so:

```
plot(xLsg, yLsg(1,:),'-o',xLsg, yLsg(2,:),'-or' );
```

So sollte Ihr Plot aussehen, wenn `GDGdemo.m` mit dem dreistufigen Verfahren aus Aufgabe 65 rechnet (die anderen Verfahren liefern ähnliche Plots:



MATLAB-Löser

Die Funktion `ystrich` können Sie von oben übernehmen. Anfangswerte setzen und Löser aufrufen erfolgt in einfachster Form als

```
y0=[1; 0];
xspan=[0, 20];
ode45(@f,xspan,y0)
```

`ode45` erzeugt ein ganz ähnlich aussehendes Bild wie oben.

Wenn Sie die anderen möglichen Varianten des `ode45`-Befehls verwenden,

```
[X, Y] = ode45(@f,xspan,y0);
```

oder

```
sol=ode45(@g,xspan,y0);
x = linspace(0,20};
y = deval(sol,x)
```

können Sie Form und Darstellung des Ergebnisses gezielt steuern.

Aufgabe 70: Populationsdynamik

Ein mathematisches Modell, das Entwicklung des Bestandes zweier Arten („Räuber“ und „Beute“) in vereinfachter Weise darstellt, wird durch das System nichtlinearer Differentialgleichungen

$$\begin{aligned} \dot{y}_1(t) &= k_1 y_1(t) - k_2 y_1(t) y_2(t) \\ \dot{y}_2(t) &= k_3 y_1(t) y_2(t) - k_4 y_2(t) \end{aligned}$$

ausgedrückt. Dabei stellen $y_1(t)$ und $y_2(t)$ den zeitlichen Bestand der Beutetiere beziehungsweise der Räuber dar.

Lösen Sie dieses System für $0 \leq t \leq 4$ unter der Annahme, daß der Anfangsbestand an Beutetieren 1000, der der Räuber 200 und die Konstanten $k_1 = 3, k_2 = 0,002, k_3 = 0,0006$ und $k_4 = 0,5$ betragen. Methode: `ode45`. Um die Ergebnisse zu veranschaulichen, stellen Sie die Lösung, also die Anzahl der beiden Populationen versus Zeit, graphisch dar.

Wiederholen Sie die Rechnung für einen Anfangsbestand von 2000 Beutetieren und 1500 Räubern.

Aufgabe 71: Schmetterlingseffekt

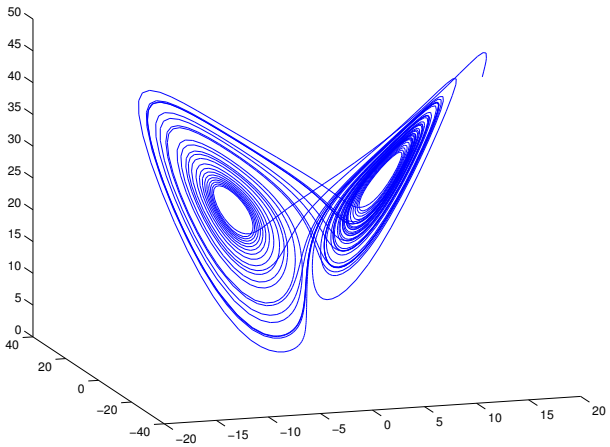
Der Meteorologe Edward N. Lorenz formulierte 1963 ein System von drei gekoppelten, nichtlinearen gewöhnlichen Differentialgleichungen zur Modellierung der Erdatmosphäre zum Zweck einer Langzeitvorhersage. Die Position eines Luftteilchens im dreidimensionalen Raum in Abhängigkeit von der Zeit wird duch die Koordinaten $x(t), y(t), z(t)$ bestimmt. In Lorenz' Modell gehorchen sie den Bewegungsgleichungen

$$\begin{aligned} dx/dt &= a(y - x) \\ dy/dt &= x(b - z) - y \\ dz/dt &= xy - cz \end{aligned}$$

Die numerische Lösung des Systems zeigt bei bestimmten Parameterwerten chaotisches Verhalten. Die typische Parametereinstellung mit chaotischer Lösung lautet:

$$a = 10, \quad b = 28, \quad c = \frac{8}{3}$$

Lösen Sie dieses System für $0 \leq t \leq 50$, Anfangsbedingungen $x = 20, y = 20, z = 40$. Stellen Sie die Lösung als Kurve in Parameterform in einem xyz-Koordinatensystem (Befehl `plot3`) dar.



Die Lösung ist für ihre typische Schmetterlings-Form bekannt.

Ü 9 Neunte Übungseinheit

Inhalt der neunten Übungseinheit:

- Eine Differentialgleichung höherer Ordnung: Umformen auf System 1. Ordnung
- Stabilität: weitere Beispiele
- Systeme von Differentialgleichungen höherer Ordnung: Umformen auf Systeme 1. Ordnung

Ü 9.1 Differentialgleichungen höherer Ordnung

Differentialgleichungen höherer Ordnung lassen sich als Systeme von Differentialgleichungen erster Ordnung schreiben.

Einfaches Beispiel: Gesucht ist eine Funktion $y(x)$. Ihre zweite Ableitung soll erfüllen

$$y''(x) = f(x, y, y')$$

mit Anfangsbedingungen (hier sind x_0, c_1 und $c_2 \in \mathbb{R}$ gegebene Werte)

$$\begin{aligned} y(x_0) &= c_1 \\ y'(x_0) &= c_2 \end{aligned}$$

Es gilt: Aus **einer** Differentialgleichung **zweiter Ordnung** wird ein System von **zwei** Differentialgleichungen **erster Ordnung**.

Einführen von zwei Hilfsfunktionen $z_1(x)$ und $z_2(x)$, definiert durch

$$\begin{aligned} z_1 &= y \\ z_2 &= y' \end{aligned}$$

Dann lautet das entsprechende System

$$\begin{aligned} z_1' &= z_2 \\ z_2' &= f(x, z_1, z_2) \end{aligned}$$

Die Anfangsbedingungen des Systems lauten

$$\begin{aligned} z_1(x_0) &= c_1 \\ z_2(x_0) &= c_2 \end{aligned}$$

Allgemeines Schema: DG höherer Ordnung $\longrightarrow$ System von DG 1. Ordnung

Aus **einer** Differentialgleichung **d-ter Ordnung** wird ein System von **d** Differentialgleichungen **erster Ordnung**.

Gegeben eine Differentialgleichung d-ter Ordnung. Mache die d-te Ableitung explizit in der Form $y^{(d)} = f(x, y, y', \ldots, y^{(d-1)})$

Führe d Hilfsfunktionen $z_1(x), z_2(x), \ldots, z_d(x)$ ein. Setze $z_1 = y,\ z_2 = y', \ldots,\ z_d = y^{(d-1)}$.

Achtung: für die d-te Ableitung braucht man *keine* Hilfsfunktion mehr!

Schreibe ein System von d Differentialgleichungen erster Ordnung für die $z_i = z_i(x)$ an:

$$\begin{aligned} z_1' &= z_2 \\ z_2' &= z_3 \\ &\vdots \\ z_{d-1}' &= z_d \\ z_d' &= f(x, z_1, \ldots, z_d) \end{aligned}$$

Die d Anfangsbedingungen des Original-Problems legen die Werte für $y(x_0), y'(x_0), \ldots, y^{(d-1)}(x_0)$ fest. Daraus werden d Anfangsbedingungen für die $z_1, z_2, \ldots, z_d$:

$$\begin{aligned} z_1(x_0) &= y(x_0) \\ z_2(x_0) &= y'(x_0) \\ &\vdots \\ z_d(x_0) &= y^{(d-1)}(x_0) \end{aligned}$$

Aufgabe 72: Mathematisches Pendel

Die Differentialgleichung des mathematischen Pendels (Auslenkung $\phi = \phi(t)$, Fallbeschleunigung g, Pendellänge ℓ) lautet

$$\ddot{\phi} + \frac{g}{\ell} \sin \phi = 0$$

Formen Sie auf ein System zweier DG erster Ordnung um und lösen Sie für $\ell = 1\,\mathrm{m}$, $g = 9{,}8\,\mathrm{m/s^2}$, Anfangsauslenkung $\phi(0) = 0{,}1\,\mathrm{rad}$, Anfangs-Winkelgeschwindigkeit $\dot{\phi}(0) = 0\,\mathrm{rad/s}$ Lösen sie für eine Zeitspanne von 4 Sekunden und zeichnen Sie den Schwingungsverlauf (Auslenkung ϕ als Funktion von t. Ändern Sie die Anfangsauslenkung auf $\phi(0) = 1{,}4\,\mathrm{rad}$ und vergleichen Sie Schwingungsform und -periode.

Aufgabe 73: Blasius'sche Gleichung

Die Blasius'sche Gleichung für die Funktion $u = u(x)$ beschreibt laminare Strömung in einer wandnahen Grenzschicht.

$$uu'' + 2u''' = 0$$

Anfangsbedingungen sind

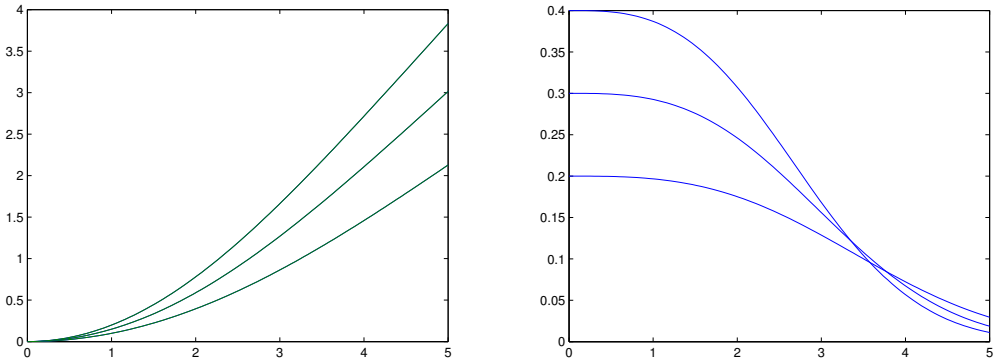
$$\begin{aligned} u(0) &= 0 \\ u'(0) &= 0 \\ u''(0) &= a \text{ , wobei } a \text{ noch frei wählbar ist.} \end{aligned}$$

Formen Sie die Differentialgleichung in ein System dreier Differentialgleichungen 1. Ordung um.

Lösen Sie die Differentialgleichung für $a = 0.2$, $a = 0.3$ und $a = 0.4$ jeweils bis $x = 5$.

Stellen Sie die drei Lösungen $u(x)$ in einem Bild graphisch dar. Stellen Sie in einem zweiten Bild die drei Ableitungen $u''(x)$ dar.

Finden Sie (z.B. durch systematisches Probieren) möglichst genau jenen a-Wert, für den $u''(5) = 0,02$ wird.



Ü 9.2 Stabilität: weitere Erklärungen, allgemeinere Testprobleme, steife Systeme

Der Abschnitt Ü 8.1.1 ist bereits kurz auf den Begriff „Stabilität“ eingegangen: Ein Verfahren heißt *stabil* bei Schrittweite h, wenn für das Modellproblem – die Differentialgleichung der abklingenden Exponentialfunktion $y = e^{-x}$ –

$$y' = -y \qquad y(0) = 1$$

die numerische Lösung mit wachsendem x nach Null konvergiert.

Ü 9.2.1 Allgemeinere Testprobleme, Stabilitätsgebiet

Es geht beim Begriff „Stabilität“ darum: wenn die exakte Lösung exponentiell abklingt, dann soll die numerische Lösung bei der gewählten Schrittweite ebenfalls irgendwie abklingende Näherungswerte liefern.

Die Differentialgleichung $y' = -y$ ist ein *sehr* einfaches Testproblem. Es ist nützlich, die Stabilität von Verfahren für etwas allgemeinere Gleichungen zu untersuchen:

Gegeben sei für die Funktion $y(x)$ das Anfangswertproblem

$$y' = \lambda y$$
$$y(0) = 1$$

für $\lambda \in \mathbb{R}$ und (erst dadurch wird das Problem wirklich allgemeiner) auch $\lambda \in \mathbb{C}$. Aber damit es nicht zu unübersichtlich wird, halten wir jetzt die Schrittweite fest[43] auf $h = 1$.

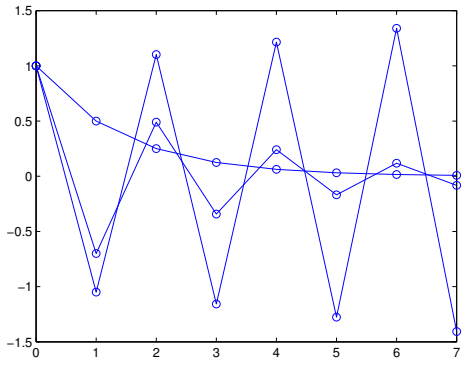
[43] Es bringt nichts, sowohl λ als auch h zu variieren. Alle für Schrittweite 1 und Parameter λ gewonnenen Resultate gelten für allgemeine h mit Parameter λ/h.

Das *Stabilitätsgebiet* eines Verfahrens ist die Menge aller $\lambda \in \mathbb{C}$, für die beim obigen Modellproblem mit Schrittweite $h = 1$ die Folge der berechneten Näherungslösungen nach Null konvergiert.

Die exakte Lösung des Anfangswertproblems ist bekanntlich $y = e^{\lambda x}$. Für $\lambda = a + bi \in \mathbb{C}$ lautet sie $y(x) = e^{ax}(\cos(bx) + i\sin(bx)$. Real- und Imaginärteil der Lösung sind also Cosinus- beziehungsweise Sinusschwingungen, multipliziert mit dem Faktor e^{ax}.

Interessant in diesem Zusammenhang sind die exponentiell abklingenden Schwingungen, also $a < 0$ oder, gleichbedeutend, $\mathrm{Re}(\lambda) < 0$. Wir wünschen uns, dass ein stabiles Lösungsverfahren solche abklingenden Schwingungen qualitativ richtig berechnet. Das Stabilitätsgebiet eines Lösungsverfahrens sollte daher idealer Weise die gesamte linke Halbebene der komplexen Zahlenebene umfassen.

Einfache Testrechnungen zeigen aber: Beim expliziten Euler-Verfahren liegen nur gewisse λ-Werte im stabilen Bereich. Hier sind Näherungslösungen des expliziten Euler-Verfahrens für verschiedene λ gezeichnet: Für $\lambda = -0,5$ monoton abnehmend, für $\lambda = -1,7$ oszilllierend, aber immer noch konvergent, und für $\lambda = -2,05$ oszillierend und divergent.



Aufgabe 74: Stabilität von Einschrittverfahren für $\lambda \in \mathbb{C}$

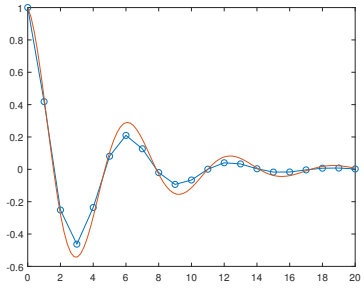
Echt spannend wird es, wenn Sie im Testproblem für $y(x)$:

$$y' = \lambda y \ ,$$
$$y(0) = 1 \ ,$$

Werte $\lambda \in \mathbb{C}$ ausprobieren. Wählen Sie zum Beispiel $\lambda = -0{,}2 + \mathrm{i}$. Untersuchen Sie das explizite und das modifizierte Eulerverfahren, sowie das dreistufigen Verfahren aus Aufgabe 65. Sie brauchen am Musterprogramm `GDGdemo.m` sonst nichts zu ändern, es rechnet frisch und munter auch mit komplexen Zahlen. Stellen Sie den Realteil der numerischen beziehungsweise der exakten Lösung graphisch dar.

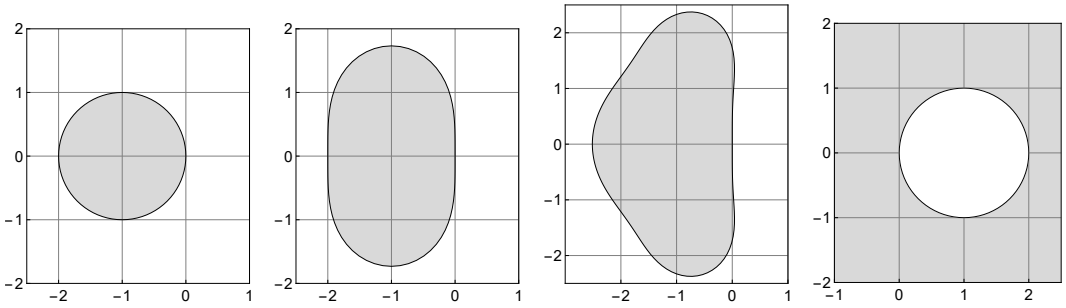
Der Realteil der exakten Lösung ist eine gedämpfte Schwingung. Für $\lambda = a + b\mathrm{i}$ lautet er $y(x) = e^{ax}\cos(bx)$.

Rechts sehen Sie numerische und exakte Lösung des Modellproblems für $\lambda = -0{,}2 + \mathrm{i}$. Gerechnet wurde mit dem dreistufigen Verfahren aus Aufgabe 65. Die numerische Lösung klingt ab, also liegt λ im Stabilitätsgebiet des Verfahrens. Wie sieht es mit den anderen Verfahren aus (explizites, modifiziertes und implizites Eulerverfahren, Heun-Verfahren)? Was passiert für $\lambda = -0{,}2 + 2\mathrm{i}$?



Die unten stehenden Abbildungen zeigen als graue Bereiche in der komplexen Zahlenebene die Stabilitätsgebiete der Verfahren (von links nach rechts:) Explizites Eulerverfahren; Modifiziertes Euler-Verfahren; Dreistufiges explizites Verfahren; implizites Euler-Verfahren. Sie können ausprobieren: wählen Sie ein λ aus dem grauen Bereich, dann berechnet das Verfahren eine Folge abklingender Werte. Weisser Bereich – exponentiell anschwellende Werte.

Ihre Aufgabe: Wählen Sie für jedes Verfahren (Explizites Eulerverfahren, Modifiziertes Euler-Verfahren, Dreistufiges explizites Verfahren, implizites Euler-Verfahren) jeweils ein λ im stabilen und eines im instabilen Bereich und lassen Sie die entsprechenden Näherungslösungen zeichnen.



Ü 9.2.2 Steife Systeme

Gäbe es nur gewöhnliche Differentialgleichungen erster Ordnung, dann wäre zur numerischen Lösung ein Allzweckverfahren wie MATLABs `ODE45` in den allermeisten Fällen völlig ausreichend; über Stabilität oder Stabilitätsgebiet bräuchte man sich auch keine großen Gedanken zu machen.

In der Praxis treten aber oft *Systeme* von Differentialgleichungen auf, und zwar ein spezieller Typ: *steife Systeme*: Siehe Vorlesungsfolien 8. Vorlesung.

Kurze Zusammenfassung:

> Wenn ein Verfahren stabil ist, kann es steife Systeme mit vernünftiger, problemgerechter Schrittweite lösen. Instabile Verfahren geben bei steifen Systemen erst mit unrealistisch kleinen Schrittweiten brauchbare Näherungslösungen.

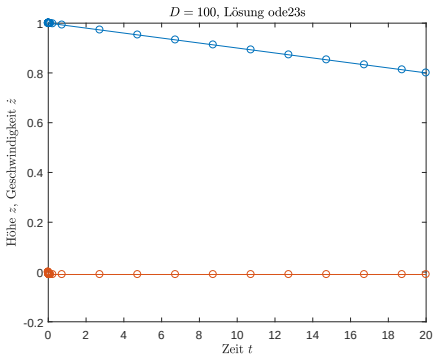
Aufgabe 75: Zähe Angelegenheit

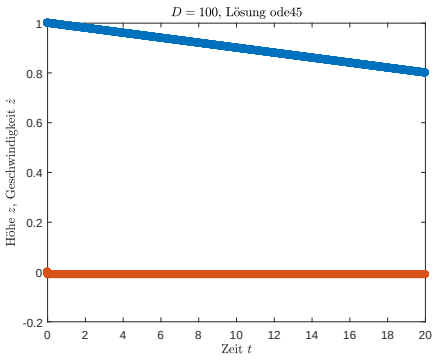
Die Vorlesungsfolien präsentieren ein Beispiel zum freien Fall im viskosen Medium: eine Glasmurmel versinkt im Honig. Die Bewegungsgleichung für die Höhe $z(t)$ lautet bei geeigneter Skalierung der Einheiten:

$$\ddot{z}(t) + D\dot{z}(t) + 1 = 0 \text{ mit } D \gg 1$$

Wählen Sie als Anfangsbedingungen $z(0) = 1$, $\dot{z}(0) = 0$. Versuchen Sie damit, für $D = 100$ die beiden Abbildungen (siehe auch die Vorlesungsfolien) zu reproduzieren.

Was passiert für $D = 10$, was für $D = 1000$? Vergleichen Sie auch die MATLAB-Löser `ode23`, `ode23t`, `ode113` und `ode15s`: welche eignen sich für welches D, welche nicht?





Zum physikalischen Hintergrund: Der Wert für D lässt sich aus dem Gesetz von Stokes zur Reibungskraft sphärischer Körper ableiten als

$$D = \frac{6\pi\eta}{m}\sqrt{\frac{r^3}{g}}$$

mit η: dynamische Viskosität, für Honig können Sie $\eta \approx 10^4\,\mathrm{Pa\,s}$ annehmen. Für eine 10 g schwere Kugel mit 1 cm Durchmesser kommen Sie auf D-Werte von einigen Tausend. Die Funktion $z(t)$ misst in dimensionslosen Einheiten. Zum Umrechnen auf Höhe in Metern und Zeit in Sekunden müssen Sie z mit dem Radius r und t mit $\sqrt{r/g}$ multiplizieren.

Aufgabe 76: Ein einfaches steifes System

Gegeben: Anfangswertproblem für $\mathbf{y} = \mathbf{y}(x), \quad 0 \le x \le 3$

$$\begin{array}{rcrcl} y_1'(x) & = & -1999y_1(x) & -1998y_2(x) \\ y_2'(x) & = & 999y_1(x) & +998y_2(x) \end{array}, \qquad \text{Anfangsbedingungen} \quad \begin{array}{rl} y_1(0) & = +1 \\ y_2(0) & = -1 \end{array}.$$

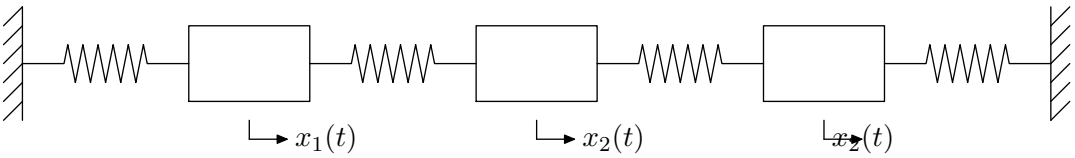
Auch dieses System wird in den Folien der 8. Vorlesung behandelt. Testen Sie, ähnlich wie in der vorigen Aufgabe, die MATLAB-Löser `ode45`, `ode15s`, `ode23`, `ode23t`, `ode113` und `ode23s`: welche eignen sich, welche nicht? Vergleichen Sie mit den beiden Abbildungen auf den Vorlesungsfolien und stellen Sie die Lösungen ähnlich dar.

Ü 9.3 Systeme von Differentialgleichungen höherer Ordnung

Abschnitt Ü 9.1 hat bereits die Umformung einer Differentialgleichung höherer Ordnung in ein System von Differentialgleichungen erster Ordnung gezeigt. Hier kommt nun der allgemeine Fall:

> Aus n Differentialgleichung d-ter Ordnung wird ein System von $n\,d$ Differentialgleichungen erster Ordnung.

Als Beispiel werden hier die Bewegungsgleichungen von drei durch Federn gekoppelten Massen diskutiert.



Die Positionen $x_1(t),\ x_2(t),\ x_3(t)$ der drei Massen sind als Funktion der Zeit t durch das folgende System von Differentialgleichungen 2. Ordnung bestimmt (Alle Massen und Federsteifigkeiten sind hier der Einfachheit halber gleich 1 angenommen).

$$\begin{aligned} \ddot{x}_1 &= -2x_1 + x_2 \\ \ddot{x}_2 &= x_1 - 2x_2 + x_3 \\ \ddot{x}_3 &= x_2 - 2x_3 \end{aligned}$$

Anfangswerte sollen sein

$$x_1(0) = 1,\ x_2(0) = x_3(0) = \dot{x}_1(0) = \dot{x}_2(0) = \dot{x}_3(0) = 0\ .$$

Das bedeutet, anfangs ist nur die erste Masse ausgelenkt, die anderen beiden befinden sich in ihren Gleichgewichtslagen; alle Anfangsgeschwindigkeiten sind 0.

Zur Schreibweise: In der Physik heißt die durch Differentialgleichungen bestimmte Funktion häufig nicht $y(x)$, sondern $x(t)$. Dabei ist x eine Ortskoordinate und t die Zeit. Der Ausdruck $\dot{x}$ bedeutet die erste, $\ddot{x}$ die zweite Ableitung von x nach t.

Wir transformieren durch Einführen folgender Hilfsfunktionen auf ein System von sechs Differentialgleichungen 1. Ordnung.

$$\begin{aligned} &z_1(t) = x_1(t),\ z_2(t) = x_2(t),\ z_3(t) = x_3(t), \\ &z_4(t) = \dot{x}_1(t),\ z_5(t) = \dot{x}_2(t),\ z_6(t) = \dot{x}_3(t)\ . \end{aligned}$$

Das äquivalente System 1. Ordnung lautet dann

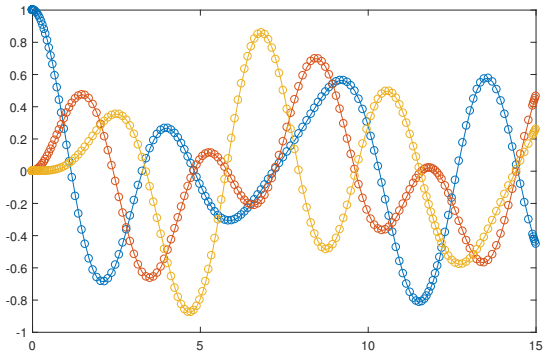
$$\begin{aligned} \dot{z}_1 &= z_4 \\ \dot{z}_2 &= z_5 \\ \dot{z}_3 &= z_6 \\ \dot{z}_4 &= -2z_1 + z_2 \\ \dot{z}_5 &= z_1 - 2z_2 + z_3 \\ \dot{z}_6 &= z_2 - 2z_3 \end{aligned}$$

Der Standard-Löser `ode45` braucht eine Funktionsdatei `dreimassen.m`, welche für einen Eingabevektor `z` die sechs Ableitungen als Vektor `zdot` berechnet.

```
function zdot=dreimassen(t,z)
zdot=[z(4);
      z(5);
      z(6);
      -2*z(1)+z(2);
      z(1)-2*z(2)+z(3);
      z(2)-2*z(3)
      ];
```

Damit benötigen wir nur noch eine Scriptdatei `loesung1.m` in der die Anfangsbedingungen für die gesuchte vektorwertige Funktion `y` sowie deren Ableitung spezifiziert werden und das betrachtete Intervall definiert wird. Danach können wir schon den Solver `ode45` aufrufen, der hier mit der Option, nur die ersten drei Komponenten des Systems auszugeben, gestartet wird. Unser Plot zeigt dann nur die Kurven für die Auslenkungen $z_1(t) = x_1(t), z_2(t) = x_2(t), z_3(t) = x_3(t)$.

```
z0=[1;0;0;0;0;0];
tspan=[0 15];
options=odeset('OutputSel',[1,2,3])
ode45(@dreimassen,tspan,z0,options)
```



Versuchen Sie auch den Befehl `ode45(@dreimassen,tspan,z0,options)` durch `[t,z]=ode45(@dreimassen,tspan,z0,options)` zu ersetzen. Was ändert sich an der Ausgabe?

Aufgabe 77: Kepler-Ellipsen

Planeten durchlaufen elliptische Bahnen; bei jeder Ellipse steht die Sonne in einem Brennpunkt (1. Keplersches Gesetz). Gesucht sind die Koordinaten eines Planeten in der $x-y$-Ebene als Funktion der Zeit: $x = x(t), y = y(t)$. Die Bewegungsgleichungen lauten:

$$\begin{aligned} r &= \sqrt{x^2+y^2} \\ \ddot{x} &= -\frac{x}{r^3} \\ \ddot{y} &= -\frac{y}{r^3} \end{aligned}$$

(Gravitationskonstante und Sonnenmasse hier der Einfachheit halber $= 1$ gesetzt). Anfangsbedingungen:

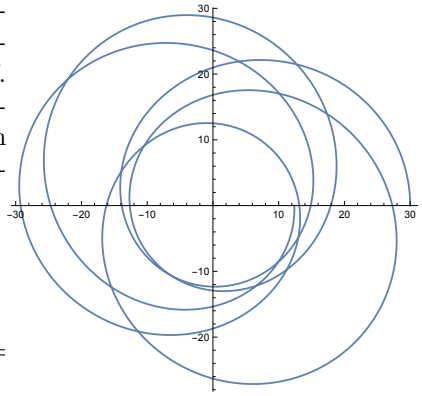
$$x(0) = 10,\ \dot{x}(0) = 0,\ y(0) = 0,\ \dot{y}(0) = 0{,}2$$

Mit diesen Anfangswerten dauert ein Umlauf $T = 31{,}25\,\pi \approx 98{,}174\,770\,4$. Bei exakter Rechnung würde sich die Umlaufbahn schließen, das heißt, die Werte zu dieser Zeit T wären gleich den Startwerten. Wie groß ist der Fehler des Standardlösers?

Stellen Sie die elliptische Umlaufsbahn in der (x, y)-Ebene dar. **Achtung:** Gefragt ist **nicht** ein Diagramm, in dem $x(t)$ und $y(t)$ über der t-Achse aufgetragen sind!

Aufgabe 78: Kepler war um 1600. Update Interstellar 2014

Im Science-Fiction-Film *Interstellar* (2014) versucht Prof. Brand (Michael Caine), Gravitationsfeldgleichungen zu lösen. Seine Tochter Amalia Brand (Anne Hathaway) wagt sich in ihrem Raumschiff an ein Schwarzes Loch heran. Im wirklichen Leben stellt Ihnen Prof. Brand (Clemens Brand) folgende Aufgabe zur Bahn eines Raumschiffes, 30 km entfernt von einem Schwarzen Loch mit Sonnenmasse. Die Bahngleichung in Polarkoordinaten lautet (für r in km)



$$r''(\phi) = -\frac{r(\phi)^2}{24} + \frac{2r'(\phi)^2}{r(\phi)} + r(\phi) - \frac{9}{2}$$

Lösen Sie für die Anfangsbedingungen $r(0) = 30,\ r'(0) = 0$ und $0 \le \phi \le 10\pi$.
Zeichnen Sie die Bahn in einem kartesischen Koordinatensystem $x = r\cos(\phi),\ y = r\sin(\phi)$.

Wie nahe kommt das Raumschiff an das Schwarze Loch, bei welchem Winkel? MATLAB-Tipp: Wert und Position des Minimums in einem Datenfeld; Zugriff auf entsprechendes Element in anderem Datenfeld;

```
[rMin,indMin]=min(r)
phiMin=phi(indMin)
```

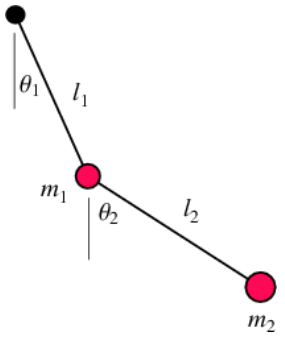
Lesen Sie zur Kontrolle auch aus der Grafik den minimalen r-Wert ab.

Die relativistischen Effekte bewirkt nur der Term $-\frac{9}{2}$ (das ist laut Theorie $-\frac{3}{2}$ mal Schwarzschildradius; der beträgt für die Sonne ziemlich genau 3 km). Wenn Sie ihn weglassen, erhalten Sie einen klassischen Kepler-Orbit. Zeichnen Sie auch den zum Vergleich in der Graphik ein.

Aufgabe 79: Doppelpendel

Ein Doppelpendel besteht aus einem Pendel, an dem ein weiteres Pendel angehängt ist.

Es ist ein einfaches Beispiel für ein System mit komplexem dynamischen Verhalten. Bei geringer Auslenkung schwingt es annähernd periodisch, oberhalb einer bestimmten Systemenergie zeigt es chaotisches Verhalten. Die Bewegungsgleichungen für die Auslenkungen θ_1 und θ_2 als Funktionen der Zeit t lauten für kleine Amplituden (in dieser Näherung tritt noch kein chaotisches Verhalten auf):



$$\ddot{\theta}_1 = -\frac{g}{\ell_1}\left((1+\mu)\theta_1 - \mu\theta_2\right)$$
$$\ddot{\theta}_2 = -\frac{g(1+\mu)}{\ell_2}\left(\theta_2 - \theta_1\right)$$

(a) Wählen Sie $\ell_1 = \ell_2 = 1,\ \mu = 1/10$ ($\mu = m_2/m_1$, Masseverhältnis), $g = 10$. Lenken Sie zu Beginn nur das untere der beiden Pendel aus seiner Ruhelage aus, also

$$\theta_1 = 0, \quad \theta_2 = 1, \quad \dot{\theta}_1 = 0, \quad \dot{\theta}_2 = 0 \text{ für } t = 0,$$

Lösen Sie für $0 \le t \le 50$.

(b) Stellen Sie die Amplituden θ_1, θ_2 als Funktionsgraphen in Abhängigkeit von t dar.

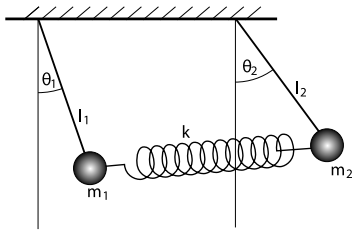
(c) Bestimmen Sie aus den numerischen Daten und prüfen Sie in der Grafik nach: Welchen Maximalwert erreicht θ_1, und zu welcher Zeit tritt dies ein?

Aufgabe 80: Gekoppelte Pendel

Zwei Pendel, zwischen denen ein Energieaustausch stattfinden kann (beispielsweise durch eine Schraubenfeder), werden als *gekoppelte Pendel* bezeichnet.

Die Bewegungsgleichungen für die Winkelauslenkungen $\theta_1 = \theta_1(t)$ und $\theta_2 = \theta_2(t)$ lauten:

$$\ddot{\theta}_1 = -\frac{g}{\ell}\theta_1 + \frac{k}{m}(\theta_2 - \theta_1)$$
$$\ddot{\theta}_2 = -\frac{g}{\ell}\theta_2 - \frac{k}{m}(\theta_2 - \theta_1)$$



(a) Wählen Sie $\ell = 1, m = 1, g = 10, k = \frac{1}{2}$ und lösen Sie für $0 \le t \le 50$ mit den Anfangsbedingungen

$$\theta_1 = 1,\ \theta_2 = 0,\ \dot{\theta}_1 = 0,\ \dot{\theta}_2 = 0 \text{ für } t = 0.$$

(b) Stellen Sie die Auslenkungen $\theta_1 = \theta_1(t), \theta_2 = \theta_2(t)$ als Funktionsgraphen dar.

(c) Lesen Sie aus der Graphik ab: Wann etwa ($\approx$ sekundengenau) ist das erste Pendel fast in Ruhe, während das zweite Pendel mit maximaler Amplitude schwingt?

(d) Verwenden Sie MATLABs `max`-Befehl und finden Sie im Ergebnisvektor für das zweite Pendel die maximale Auslenkung und den zugehörigen Zeitpunkt.

Aufgabe 81: Dreikörperproblem

Im Gravitationsfeld von Erde und Mond bewegt sich ein Raumschiff. Die Bewegungsgleichungen sind hier rechts gegeben.
μ ist das Masseverhältnis im Mond-Erde-System. Die weiteren Größen sind nebenstehend definiert.
Die Lösung $[x(t); y(t)]$ gibt die Position des Raumschiffes in der Bahnebene. Koordinatenursprung im Erdmittelpunkt, x-Achse zeigt immer in Richtung Mond. Längeneinheit ist die Erde-Mond-Entfernung (380 000 km). Zeiteinheit ist ein Mondumlauf (27 Tage). (Der Mond hat also immer Position [1; 0], das Koordinatensystem rotiert gegenüber einem Inertialsystem mit einer Umdrehung pro Zeiteinheit. Die Bewegungsgleichungen im Inertialsystem wären noch komplizierter!)

$$\ddot{x} = 2\dot{y} + x - \frac{\mu^*(x+\mu)}{r_1^3} - \frac{\mu(x-\mu^*)}{r_2^3}$$
$$\ddot{y} = -2\dot{x} + y - \frac{\mu^* y}{r_1^3} - \frac{\mu y}{r_2^3}$$

$$\mu = 1/82{,}45$$
$$\mu^* = 1 - \mu$$
$$r_1 = \sqrt{(x+\mu)^2 + y^2}$$
$$r_2 = \sqrt{(x-\mu^*)^2 + y^2}.$$

(a) Formen Sie die Bewegungsgleichungen in ein System von Differentialgleichungen 1. Ordung um und lösen Sie für $0 \le t \le 10$ mit Anfangsbedingungen

$$x = 1{,}2; \quad y = 0; \quad \dot{x} = 0; \quad \dot{y} = -1 \qquad \text{für } t = 0\ .$$

(b) Stellen Sie die Bahn des Raumschiffs in der xy-Ebene dar.

(c) Die Rechnung startet mit dem Raumschiff genau hinter dem Mond. Zu welcher Zeit T erreicht das Raumschiff erstmals negative x-Werte? (Dann ist es in Erdnähe, und es wäre Zeit für ein Bremsmanöver, wenn die Astronauten nach Hause wollen.)

(d) Ändern Sie die y-Komponente der Anfangsgeschwindigkeit im Prozentbereich und finden Sie durch Probieren jenen Wert (auf $\pm 0{,}5\%$ genau), für die sich die Umlaufbahn möglichst gut wieder schließt.

Ü 10 Zehnte Übungseinheit

Inhalt der zehnten Übungseinheit:

- Zweiter Kenntnisnachweis: Themengebiete
- Weitere Beispiele zu nichtlinearen Datenmodellen, Eigenwertaufgaben und Einschrittverfahren für gewöhnliche Differentialgleichungen.
- Für den 2. Kenntnisnachweis nicht mehr relevant, aber als Ergänzung zum Vorlesungsstoff und -skript als eigenes Kapitel angehängt: Fourierreihen, Fourieranalyse, Fast Fourier Transform (FFT)

Ü 10.1 Zweiter Kenntnisnachweis: Themengebiete

Der zweiten Kenntnisnachweis stellt Ihnen zwei Teilaufgaben aus den Themengebieten

- Überbestimmte nichtlineare Systeme, nichtlineare Datenmodelle, Gauß-Newton-Verfahren. Typische Beispiele: Aufgaben 42 und 43. Es folgt noch zwei Beispiele: Aufgaben 82 und 83.

  Siehe auch die Vorlesungsfolien zur 7. Vorlesung (klick!) mit einem Musterprogramm `GaussNewtonCoronaFit.m` (klick!) .

- Eigenvektoren und Vektor-Iteration. Typische Beispiele: Aufgabe 61 und die hier folgenden Aufgaben 84 und 85.

- Systeme von Differentialgleichungen höherer Ordnung, Umformen auf Systeme 1. Ordnung, Typisch: Aufgaben77–81.

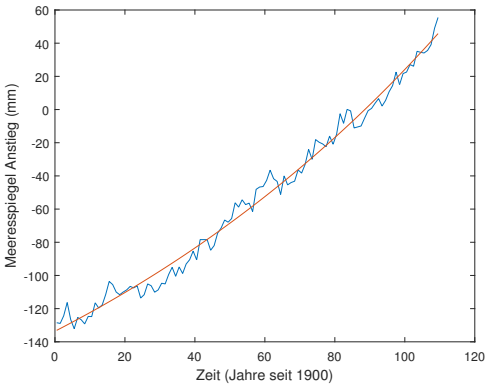
  Explizite und implizite Einschrittverfahren selbst implementiert: Aufgabe65 mit Musterprogramm `GDGdemo.m` (klick!), weitere Aufgaben 86 und 87.

Ü 10.2 Nichtlineare Datenmodelle: Beispiele zur Wiederholung

Aufgabe 82: Noch ein Untergangs-Szenario

Die nebenstehende Grafik zeigt Daten zum globalen Anstieg des Meeresspiegels. (Zeit in Jahren seit 1900, Höhenänderung Δh in mm. Der Datensatz lässt sich als `KNW_2.dat` hier (klick!) herunterladen.)
Erstellen Sie Modelle für den Datensatz:

1. Linear: $\Delta h = a + bt$
2. Nichtlinear $\Delta h = a + b\exp(ct)$



Für das exponentielle Modell braucht das Gauß-Newton-Verfahren Startwerte. Verwenden Sie $a = -300$; $b = 100$; $c = 0.01$ Stellen Sie die Datenpunkte und die zwei Modelle in einer Grafik dar.
Aus dieser Grafik sollte sich die Antworten auf folgende Fragen ablesen lassen:

a) Ab $\Delta h = 150\,\text{mm}$ bekommen auf dem Markusplatz in Venedig Tauben und Touristen nasse Füße. Wann passiert das, je nach Modell?

b) Der an der Atlantikküste gelegene US-Bundesstaat North Carolina hat per Gesetz 2011 dem Meeresspiegels verboten, mehr als linear anzusteigen:

> *Rates of sea-level rise may be extrapolated linearly but shall not include scenarios of accelerated rates of sea-level rise.*

Für den Fall, dass der Meeresspiegel sich nicht daran hält: Wie groß ist der Unterschied zwischen linearem und nichtlinearem Modell am Ende des 21. Jahrhunderts?[44]

Aufgabe 83: Arrhenius-Gleichung

Das Technische Merkblatt zum Zweikomponentenkleber UHU PLUS ENDFEST 300 gibt nebenstehende Werte für die Aushärtezeit in Abhängigkeit von der Temperatur an.
Als mögliches Modell beschreibt die Arrhenius-Gleichung die Abhängigkeit der Reaktionsgeschwindigkeit von der Temperatur:

$$t_{\text{react}} = a \exp(b/T)$$

Bestimmen Sie die Parameter a und b ausgehend von den Startwerten $a^{(0)} = 2 \times 10^{-5}$ und $b^{(0)} = 5 \times 10^{3}$. (Zwei Gauß-Newton-Iterationen reichen.)

Temperatur T (K)	Aushärtezeit t_{react} (min)
313	180
343	45
353	30
363	20
373	10
393	7
413	6

Versuchen Sie auch ein Modell der Form

$$t_{\text{react}} = a_0 + a_1 T + a_2 T^2 + a_3 T^3$$

Zeichnen Sie Datenpunkte und Anpassungsfunktionen. Wie groß ist jeweils der Fehler bei $T = 373\,\text{K}$, und welche Aushärtezeit sagen die Modelle für Raumtemperatur (20 °C) voraus?

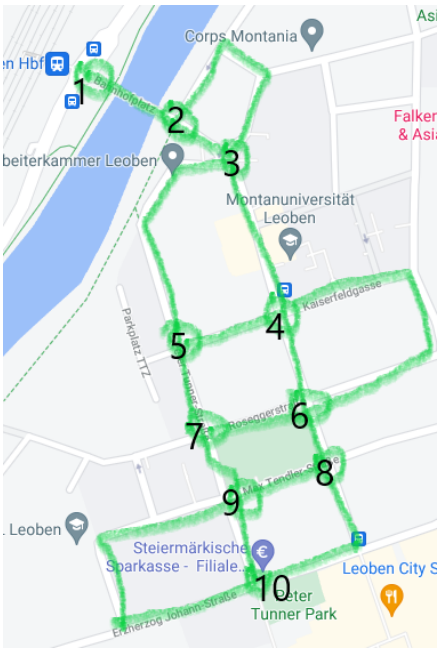
Ü 10.3 Vektor-Iteration, Eigenvektoren: Beispiele zur Wiederholung

Aufgabe 84: Random Walk

Das ist Aufgabe 61 in etwas anderer Einkleidung. Modelliert wird hier ein *Random Walk* oder auch ein *Markov-Prozess*. Es macht nichts, wenn Sie diese für viele Anwendungen wichtigen Fachbegriffe noch nicht kennen, hier begegnen sie Ihnen in einem anschaulichen Beispiel.

[44] Fairer Weise muss man dazu sagen: North Carolina hat diesen Gesetzestext inzwischen revidiert und erlaubt – unter strengen Auflagen – doch zunehmende Anstiegsrate.

Die Skizze rechts zeigt vereinfacht das Straßennetz zwischen Hauptplatz und Bahnhof von Leoben. Eine Gruppe Studierender hat am Hauptplatz das nahende Semesterende etwas zu ausgiebig gefeiert und torkelt nun planlos durch das Uni-Viertel. An jedem der Punkte 1–10 wählt jede Person zufällig und mit gleicher Wahrscheinlichkeit eine der möglichen Richtungen. Dadurch verteilen sich alle nach einiger Zeit im Straßennetz.



Dieser fröhliche Abend lässt sich in folgender Form modellieren:
Ein Vektor $\mathbf{x} \in \mathbb{R}^{10}$ enthält in Komponente i die Anzahl der Personen, die sich gerade auf Punkt i oder auf dem Weg dorthin befinden. Eine Matrix $A = [a_{ij}]$ gibt in Spalte j an, mit welcher Wahrscheinlichkeit jemand Knoten j in Richtung i verlässt.
Dann gibt (etwas vereinfacht gesagt) das Matrix-Vektor-Produkt $A \cdot \mathbf{x}$ die Verteilung der Personen eine Zeiteinheit später an. Fortgesetzte Matrix-Vektor-Multiplikation führt zu einem stabilen Zustand: die Verteilung der Personen im Netzwerk ändert sich nicht mehr.

Beginnen Sie mit Startvektor $\mathbf{x}^{(0)} = [0; 0; \ldots 0; 100]$ Das bedeutet: Aufenthaltswahrscheinlichkeit 100% bei Knoten 10, die gesamte Gruppe startet beim Peter-Tunner-Park.

Setzen Sie als Matrix der Übergangs-Wahrscheinlichkeiten

$$A = \begin{bmatrix}
0 & 1/3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1/2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 2/3 & 0 & 1/4 & 1/3 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1/4 & 0 & 1/3 & 1/2 & 0 & 0 & 0 & 0 \\
0 & 0 & 1/4 & 1/4 & 0 & 0 & 1/3 & 0 & 0 & 0 \\
0 & 0 & 0 & 1/2 & 0 & 0 & 1/3 & 1/3 & 0 & 0 \\
0 & 0 & 0 & 0 & 1/3 & 1/4 & 0 & 0 & 1/4 & 0 \\
0 & 0 & 0 & 0 & 0 & 1/4 & 0 & 0 & 1/4 & 1/3 \\
0 & 0 & 0 & 0 & 0 & 0 & 1/3 & 1/3 & 0 & 2/3 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/3 & 1/2 & 0
\end{bmatrix}$$

(Tipp: die üblichen PDF-Viewer erlauben, die Matrixelemente rauszukopieren; mit wenig Aufwand können sie in MATLAB-Dateien als Matrix übernommen werden.)

- Berechnen Sie ausgehend von $\mathbf{x}^{(0)}$ die Zustandsvektoren

$$\begin{aligned}
\mathbf{x}^{(1)} &= A\mathbf{x}^{(0)} \\
\mathbf{x}^{(2)} &= A\mathbf{x}^{(1)} \\
&\vdots \\
\mathbf{x}^{(k)} &= A\mathbf{x}^{(k-1)}
\end{aligned}$$

  so lange, bis die Änderung in der 2-Norm $\|\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\|_2 < 10^{-2}$ wird.

- Wie groß ist Komponente 3 von $\mathbf{x}^{(k)}$ (die Aufenthaltswahrscheinlichkeit einer Person am Knoten 3, bei der MUL)?

- Berechnen Sie auch mit MATLAB-Befehlen direkt den Eigenvektor zum betragsgrößten Eigenwert von A. Wie lautet der Eigenwert?
- Skalieren Sie den zugehörigen Eigenvektor so, dass die Summe der Komponenten 100 ergibt und vergleichen Sie mit dem Ergebnis der Vektor-Iteration. Sie sollten im Rahmen der Rechengenauigkeit Übereinstimmung feststellen können.

Aufgabe 85: Kredit(un)würdigkeit

Größere Banken kategorisieren die Kreditwürdigkeit ihrer Kunden gemäß einem Stufensystem: von 1=„höchste Kreditwürdigkeit“ bis zu 5=„kein Kredit möglich“. Monatlich werden Daten erhoben, die den Wechsel der Kunden zwischen den Stufen beschreiben. Nehmen Sie folgendes Modell, gegeben durch die untenstehende (fiktive) Matrix A an, das diesen Wechsel darstellt:

$$A = \begin{pmatrix} 0.92 & 0.06 & 0.03 & 0.003 & 0.001 \\ 0.07 & 0.8 & 0.14 & 0.007 & 0.0007 \\ 0.005 & 0.1 & 0.62 & 0.15 & 0.012 \\ 0.003 & 0.03 & 0.11 & 0.44 & 0.46 \\ 0.002 & 0.01 & 0.1 & 0.4 & 0.52 \end{pmatrix}$$

Hierbei gibt der Eintrag in Zeile i und Spalte j an, wieviel Prozent der Kunden in Stufe j im nächsten Monat in Stufe i wechseln werden. Falls man eine Verteilung v_0 der Kunden in die Stufen 1–5 anteilsmäßig gegeben hat, etwa

$$v_0 = \begin{pmatrix} 0.1 \\ 0.2 \\ 0.2 \\ 0.3 \\ 0.2 \end{pmatrix},$$

so erhält man die Verteilung für den Folgemonat v_1 durch $v_1 = Av_0$.

Berechnen Sie die Verteilung nach drei Monaten mit der gegebenen Ausgangsverteilung v_0.

Welche Verteilung v_s verändert sich gegenüber den monatlichen Stufenwechseln nicht? Schreiben Sie diese Fragestellung als Eigenwertproblem an (als Kommentar in Ihrem .m-File). Wie groß ist der Kundenanteil in Stufe 5?

Welche anderen Eigenwerte besitzt diese Matrix noch?

Ü 10.4 Gewöhnliche Differentialgleichungen: Weitere explizite und implizite Verfahren

Aufgabe 86: Implizites Trapez-Verfahren

Ein Schritt des *impliziten Trapez-Verfahrens* zur numerischen Lösung einer Differentialgleichung $y' = f(x, y)$ lautet

$$y(x+h) = y(x) + \frac{h}{2}\left[f\Big(x, y(x)\Big) + f\Big(x+h, y(x+h)\Big)\right].$$

(a) Rechnen Sie für die Differentialgleichung $y' = (1-x)y$ mit Schrittweite $h = 1/2$ ausgehend von der Anfangsbedingung $y(0) = 1$ einen Näherungswert für $y(3)$.

(b) Wiederholen Sie die Rechnung mit $h = 1/4$ und geben sie auch dafür den Endwert $y(3)$ an.

(c) Der 12-stellig genaue Wert ist $y(3) = 0{,}223\,130\,160\,148$. Vergleichen Sie die Diskretisierungsfehler aus den Ergebnissen (2a) und (2b). In welchem Verhältnis ϵ_1/ϵ_2 stehen die Fehler? Welche Fehlerordnung p vermuten Sie daher?

Aufgabe 87: *Rocket Science* aus dem vorigen Jahrtausend

In einem NASA Report aus 1969 beschreibt Erwin Fehlberg folgendes Einschritt-Verfahren

$$F(x,y,h) = \frac{1}{512}(k_1 + 510k_2 + k_3) \quad \text{mit}$$

$$k_1 = f(x,y)\ , \quad k_2 = f\Big(x + \frac{h}{2},\ y + \frac{h}{2}k_1\Big)\ , \quad k_3 = f\Big(x + h,\ y + \frac{h}{256}(k_1 + 255k_2)\Big)$$

(a) Implementieren Sie das Verfahren für die Differentialgleichung $y' = \sin(y) + x$ und rechnen Sie mit Schrittweite $h = 1/2$ ausgehend von der Anfangsbedingung $y(1) = -1$ einen Näherungswert für $y(4)$.

(b) Wiederholen Sie die Rechnung mit $h = 1/4$ und geben sie auch dafür den Endwert $y(4)$ an.

(c) Der (auf 12 Nachkommastellen) genaue Wert ist $y(4) = 5{,}787\,447\,802\,140$. Vergleichen Sie die Diskretisierungsfehler aus den Ergebnissen (2a) und (2b). In welchem Verhältnis ϵ_1/ϵ_2 stehen die Fehler? Welche Fehlerordnung p vermuten Sie daher?

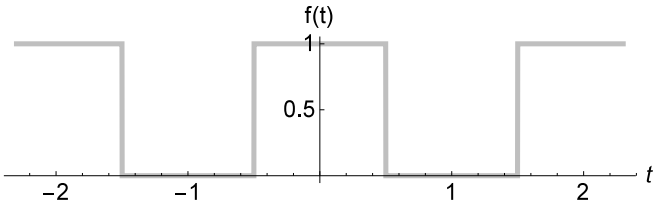
Ü 11 Fourierreihen, Fourieranalyse, Fast Fourier Transform (FFT)

Dieses Kapitel ist nicht mehr im Stoffumfang des zweiten Kenntnisnachweises enthalten. In der Vorlesungsprüfung werden aber sehr wohl Fragen dazu gestellt.

Es gibt im Hauptteil des Skriptums (noch) kein eigenes Kapitel dazu. Der Stoff wird anhand der Vorlesungsfolien (klick!) und der Übungsaufgaben erklärt und erarbeitet.

Ü 11.1 Fourierreihen approximieren periodische Funktionen

Gegeben ist ein periodisches Signal $f(t)$ in Form einer Rechtecks-Kurve mit Periode $T = 2$. Für $-\frac{1}{2} < t < \frac{1}{2}$ ist $f(t) = 1$, für $\frac{1}{2} < t < \frac{3}{2}$ ist $f(t) = 0$.



Approximieren Sie diese Funktion durch Fourierreihen. Verwenden Sie zuerst Terme bis zur Ordnung 1, dann 5 und schließlich 55.

Anleitung:

Fourierreihe einer periodischen Funktion $f(t)$ mit Periode T:

$$f(t) = \frac{a_0}{2} + a_1 \cos(\frac{2\pi}{T}t) + a_2 \cos(\frac{2\pi}{T}2t) + \cdots + a_n \cos(\frac{2\pi n}{T}t) + \cdots$$
$$+ b_1 \sin(\frac{2\pi}{T}t) + b_2 \sin(\frac{2\pi}{T}2t) + \cdots + b_n \sin(\frac{2\pi n}{T}t) + \cdots$$

mit

$$a_n = \frac{2}{T}\int_{-\frac{T}{2}}^{\frac{T}{2}} f(t) \cdot \cos(\frac{2\pi n}{T}t)\,\mathrm{d}t \qquad b_n = \frac{2}{T}\int_{-\frac{T}{2}}^{\frac{T}{2}} f(t) \cdot \sin(\frac{2\pi n}{T}t)\,\mathrm{d}t$$

Summen-Symbol Σ und Kreisfrequenz $\omega = 2\pi/T$ vereinfachen die Schreibweise. Außerdem kommt es bei den Integralen nur darauf an, über eine volle Periode zu integrieren; ob von $-T/2$ bis $T/2$, von 0 bis T oder über irgend einen anderen Bereich, der eine volle Periode abdeckt, ist egal. Daher sind in folgender Formulierung die Grenzen beginnend mit irgendeinem t_0 bis $t_0 + T$ angegeben.

Fourierreihe für $f(t)$ mit Periode T und Kreisfrequenz $\omega = \frac{2\pi}{T}$:

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

mit

$$a_n = \frac{2}{T}\int_{t_0}^{t_0+T} f(t)\cdot\cos(n\,\omega t)\,\mathrm{d}t \qquad b_n = \frac{2}{T}\int_{t_0}^{t_0+T} f(t)\cdot\sin(n\,\omega t)\,\mathrm{d}t$$

Im Beispiel hier ist $T = 2$. Die Funktion f ist aber nur im Bereich $-\frac{1}{2} < t < \frac{1}{2}$ ungleich 0 und dort sogar konstant mit Wert $f(t) = 1$. Das vereinfacht die Berechnung der Integrale ganz wesentlich:

- Die Interation läuft nicht von $-T/2$ bis $T/2$, sondern nur dort wo $f \neq 0$, also im Intervall $-\frac{1}{2} < t < \frac{1}{2}$.

- Beachten Sie: dieses Signal ist symmetrisch bezüglich der y-Achse! Weil $\sin(a) = -\sin(-a)$, heben sich bei der Integration von (Signal mal Sinus-Term) die Beiträge der positiven und der negativen t-Werte auf, alle b_n-Koeffizienten sind hier gleich 0.

Es bleibt also zu berechnen:

$$a_n = \int_{-1/2}^{1/2} 1\cdot\cos(n\pi t)\,dt$$

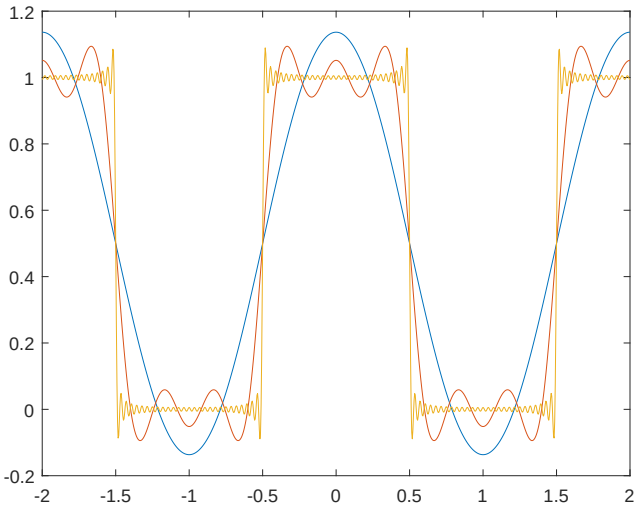
Ergebnis:

$$\begin{aligned} a_0 &= 1 \\ a_n &= (-1)^{(n-1)/2}\cdot\frac{2}{n\pi} && \text{für } n > 0 \text{ ungerade} \\ a_n &= 0 && \text{für } n > 0 \text{ gerade} \end{aligned}$$

Die Fourierreihe lautet somit

$$f(t) = \frac{1}{2} + \frac{2}{\pi}\left(\cos(\pi t) - \frac{1}{3}\cos(3\pi t) + \frac{1}{5}\cos(5\pi t) - \frac{1}{7}\cos(7\pi t) \pm \cdots\right)$$

Und so sehen die Approximationen bis $n = 1,\ 5,\ 55$ aus:



Sie können versuchen, noch viel mehr Terme zu summieren, damit die Approximation genauer wird. Aber ganz egal, wie viele Terme Sie summieren:

> **Gibbs'sches Phänomen:** Bei Approximation unstetiger Funktionen durch Fourier-Summen treten in der Umgebung von Sprungstellen Überschwingungen auf. Diese überschießenden Zacken werden mit zunehmendem n schmäler, die maximalen Auslenkung bleibt aber konstant im Bereich $\approx 10\,\%$ der Sprunghöhe.

Aufgabe 88: Rechtecks-Funktion und Gibbs-Phänomen

Gegeben ist ein periodisches Rechtecks-Signal wie oben, nur mit schmäleren Rechtecken: Periode $T = 2$ wie vorher, aber nur für $-\frac{1}{4} < t < \frac{1}{4}$ ist $f(t) = 1$, sonst ist $f(t) = 0$.

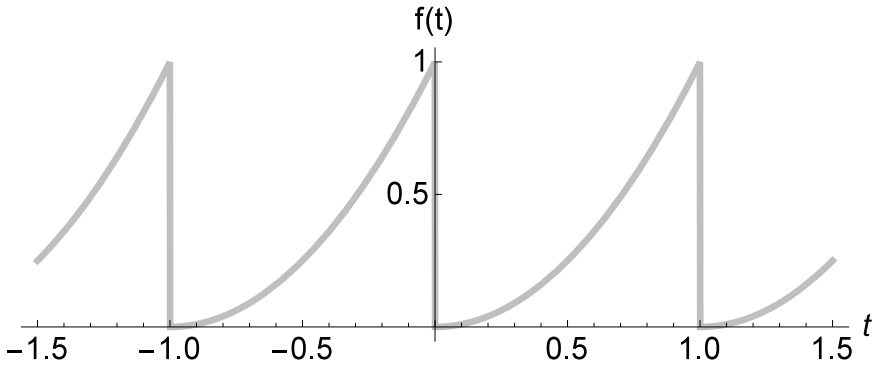
Berechnen Sie die Terme der Fourierreihe (Nehmen Sie MATLABs symbolische Integration zu Hilfe, wenn Sie die Integrale mit Papier, Stift und Hirn alleine nicht auswerten können!)

Zeichnen Sie die Approximationen bis $n = 1,\ 5,\ 55$ und vergleichen Sie mit der obigen Abbildung.

Die Approximationen zeigen das Gibbs-Phänomen. Lesen Sie aus der Grafik die maximale Höhe ab. Bessert sich dieser Wert, wenn Sie n weiter erhöhen? (Wählen Sie n so hoch, wie sie wollen oder MATLAB vernünftig rechnen kann.)

Aufgabe 89: Schräge Säge

Das ist ein Beispiel zum allgemeinen Fall, in dem sowohl Sinus- als auch Cosinus-Terme auftreten. Gegeben ist die Funktion $y = t^2$ für $0 \le t \le 1$, periodisch fortgesetzt außerhalb dieses t-Intervalls.



Berechnen Sie bis $n = 5$ die Koeffizienten der Fourierreihe. Stellen Sie y und die Fourier-Approximation dar.

Sie können MATLABs symbolische Integration verwenden, zum Beispiel

```
>> int((1-t^2)*cos(3*pi*t),t,0,1)
 ans =
 2/(9*pi^2)
```

Zeichnen Sie auch Approximationen mit höherem n und erklären Sie den Begriff „Gibbs'sches Phänomen“ anhand der Darstellungen.

Aufgabe 90: Leistung, quadratischer Mittelwert

Gegeben ist noch einmal das Rechtecks-Signal $f(t)$ mit Periode $T = 2$. Für $-\frac{1}{2} < t < \frac{1}{2}$ ist $f(t) = 1$, für $\frac{1}{2} < t < \frac{3}{2}$ ist $f(t) = 0$.

In vielen Anwendungen gibt der *quadratische Mittelwert*

$$\frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} f(t)^2 \, \mathrm{d}t$$

die Leistung oder Energie an.

Berechnen Sie diesen Wert für das Rechtecks-Signal.

Die Fourierreihe dieser Funktion ist oben angegeben. Summieren Sie die Quadrate aller Fourier-Koeffizienten bis bis $n = 1,\ 5,\ 55$ gemäß folgender Formel (Achtung, der Koeffizient a_0 ist in den Fourier-Formeln immer ein Sonderfall, er stiftet regelmäßig Verwirrung!).

$$P = \frac{a_0^2}{4} + \frac{1}{2} \sum_{n>1} a_n^2 + b_n^2$$

Was fällt Ihnen im Vergleich zum vorher berechneten Integral auf?

In der Physik und den Ingenieurwissenschaften wird dieser Zusammenhang so formuliert: Die Energie eines Signals im Zeitbereich (Integral über f^2) ist (bis auf Skalierung) gleich seiner Energie im Frequenzbereich. Jede Frequenz > 0 trägt $\frac{1}{2}(a_n^2 + b_n^2)$ zur Gesamtenergie bei. Bei komplexer Schreibweise wäre dieser Zusammenhang einfacher, weil dort der Koeffizient zu $n = 0$ kein Sonderfall ist.

Wie groß ist bei der Rechtecks-Funktion der Beitrag der niedrigsten drei Frequenzen (der ersten drei Terme $\neq 0$ in der Fourierreihe) im Verhältnis zur Gesamtenergie?

Ü 11.2 Fourieranalyse findet Frequenzen, FFT findet' flotter.

Aufgabe 91: Frequenz-Analyse eines Audio-Signals

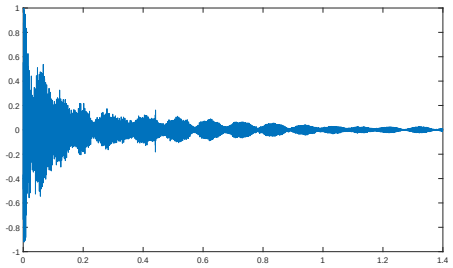
Fourier-Analyse zerlegt ein Signal in seine Frequenzanteile. Diese Anleitung zeigt, wie Sie alle Frequenzkomponenten berechnen, aus denen ein Audio-Signal besteht.

Laden Sie dazu die Datei `klang1.wav` von der Übungshomepage (klick!) und lesen Sie die Daten im MATLAB ein. Es gibt dafür einen speziellen Befehl,

```
[X,fs] = audioread('klang1.wav');
```

Der Vektor `X` enthält die Signal-Messwerte, aufgezeichnet mit der Abtast-Frequenz (*sampling frequency*) `fs` (in Hertz, $1\,\mathrm{Hz} = \mathrm{s}^{-1}$). Das heißt, es liegen pro Sekunde `fs` Datenpunkte vor; die gesamte Signaldauer ist also

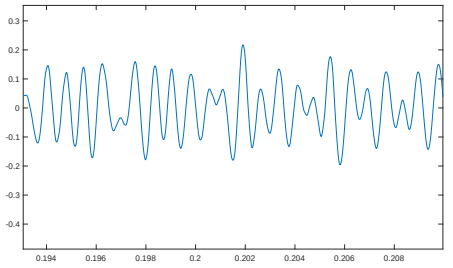
$$\frac{\text{Anzahl der Datenpunkte} - 1}{\text{Abtastfrequenz}}$$



Erzeugen Sie die zu den Datenpunkten gehörige Zeitachse und plotten Sie das Signal.

```
n = length(X);
t = linspace(0,(n-1)/fs,n);
plot(t,X);
```

Sie sehen eine abklingende Schwingung. Wenn Sie neugierig sind und wissen wollen, wie sich das anhört: der Befehl `sound(X,fs)` spielt das Signal ab. Hineinzoomen (Hier abgebildet ist das Zeitintervall $\approx 0{,}194 \leq t \leq 0{,}208$ zeigt einen annähernd sinusförmigen Verlauf, überlagert mit anderen Effekten.



Sie können die Frequenz f dieser deutlich sichtbaren Frequenzkomponente direkt aus der Grafik abschätzen: Zählen Sie die Wellenberge im Intervall $[0{,}194;\ 0{,}208]$, dividieren Sie durch die Zeitdauer. Damit haben Sie schon eine wesentliche Information aus den Daten herausgelesen: Ein Hauptanteil sind Sinus-Schwingungen mit $\approx 1400\,\mathrm{Hz}$, denen sich weitere Schwingungen überlagern.

Das ist aber nur eine ganz grobe Näherung; das Signal ist wesentlich komplexer.

Der Grundgedanke der Fourieranalyse ist:

> Ein Signal, gegeben durch n Datenpunkte, lässt sich als Summe von n Schwingungstermen (Sinus, Cosinus, komplexe E-Funktionen) darstellen

Nun, *eine* Schwingung haben wir gefunden. Es fehlen noch gut 60 000 weitere (so viele Datenpunkte enthält das Signal). Ein direkter Ansatz, diese n Komponenten zu berechnen, würde $O(n^2)$ Rechenoperationen erfordern – viel zu aufwändig bei großen Datenmengen! Im Prinzip hat schon Karl Friedrich Gauß 1805 ein schnelleres Rechenverfahren dazu entworfen. Seit den 1960-er Jahren sind unter dem Begriff "FFT" (*fast Fourier transform*, schnelle Fourier-Transformation) Algorithmen für Computer verfügbar und inzwischen unabdingbar in der

Signalverarbeitung. Die FFT reduziert den Rechenaufwand von $O(n^2)$ auf $O(n \log n)$. Nur dadurch ist FFT in Echtzeit-Anwendungen einsetzbar.

In MATLAB lautet der Befehl einfach

```
Y = fft(X);
```

Den n Signal-Messwerten im Datenvektor `X` entsprechen ebensoviele Fourier-Koeffizienten im der diskreten Fourier-Transformierten `Y`.

Die Formel, die sich hinter dem `fft`-Befehl verbirgt und die MATLAB sehr trickreich und rechengünstig auswertet, lautet

$$Y_k = \sum_{j=1}^{n} X_j \exp\left[\frac{-2\pi i}{n}(j-1)(k-1)\right]$$

Ein kleiner Nachteil: dieser FFT-Befehl in Standard-Ausführung liefert nicht direkt Sinus- und Cosinus-Koeffizienten für eine klassische Fourierreihe (die Formel mit den a_n- und b_n-Termen). Diese Koeffizienten lassen sich aber leicht bestimmen.

Die Darstellung des Signalvektors `X` als Summe komplexer Exponentialfunktionen lautet

$$X_j = \frac{1}{n} \sum_{k=1}^{n} Y_k \exp\left[\frac{2\pi i}{n}(j-1)(k-1)\right]$$

Das sieht komplizierter aus, als es ist. Wichtig ist der Aufbau dieser Formel:

Datenvektor X = Skalierungsfaktor mal Summe von ...
... (Amplitude Y_k mal Schwingung mit Frequenz ω_k)

FFT:
Datenvektor $X \mapsto$ Frequenzvektor Y, in MATLAB: `Y = fft(X)`

Umkehrung, inverse FFT:
Frequenzvektor $Y \mapsto$ Datenvektor X , in MATLAB: `X = ifft(Y)`

Noch ein Detail: Die FFT-Formel arbeitet nur mit den Vektoren `X` und `Y`. Sie indiziert die `X`-Komponenten mit $j = 1, 2, \ldots, n$. Sie weiß nichts von der zugehörigen Zeitachse mit Zeitpunkten $t_1, t_2, \ldots, t_n$ und den zu Y_k zugehörigen Kreisfrequenzen ω_k. Die Umrechnung oder Zuordnung von Index j zu Zeitpunkt t_j und Fourier-Komponente k zu Kreisfrequenz ω_k lautet

$$t_j = \frac{j-1}{f_s} \ , \qquad \omega_k = \frac{2\pi(k-1)}{n} f_s$$

Es gilt bei reellen Datenvektoren mit geradzahliger Länge n: Die Real- und Imaginärteile der Fourier-Terme `Y(2)` bis `Y(n/2)`, mit $2/n$ multipliziert, geben jeweils die Cosinus- und Sinus-Amplituden zu den Frequenzen

$$\frac{f_s}{n}, \quad 2\frac{f_s}{n}, \quad 3\frac{f_s}{n}, \ \ldots, \ \frac{n-1}{2}\frac{f_s}{n}$$

an. (Mit 2π multipliziert sind das die Kreisfrequenzen ω_k.) Ein Sonderfall ist der Term `Y(1)`: er ist die Summe aller Signalwerte; dividiert durch n gibt er den Mittelwert des Signals an. Noch ein Sonderfall ist (bei geradem n) der Term `Y(n/2+1)`: er ist nur mit $1/n$ zu multiplizieren und

gibt die Cosinus-Amplitude zur Frequenz $f_s/2$ an. (Bei ungeradem n gibt es keine Information mehr zu $f_s/2$.)

Wichtig ist noch bei reellen Datenvektoren: nur bis zur Mitte des Y-Vektors ist tatsächlich Information enthalten; die zweite Hälfte enthält gespiegelt dieselben Werte, nur komplex-konjugiert.

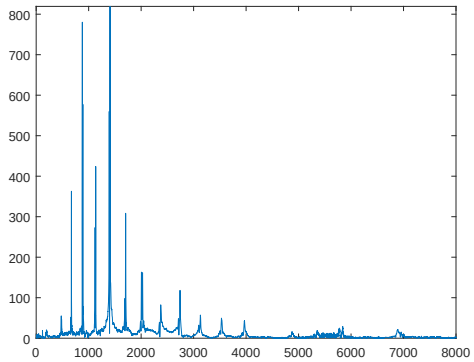
> FFT liefert Frequenz-Information bis zur Hälfte der Abtast-Frequenz f_s.
> Man nennt $f_s/2$ die **Nyquist-Frequenz**.

Um Frequenzanteile eines Signals mindestens bis zu einer Frequenz $f_{\max}$ korrekt zu erfassen, muss die Abtastfrequenz f_s größer sein als $2f_{\max}$. Das menschliche Hörvermögen reicht (in jungen Jahren bestenfalls) bis etwa 22 000 Hz. Deswegen ist auf Audio-CDs (und in unserer Beispieldatei `klang1.wav` die Abtast-Frequenz $f_s = 44\,100$ Hz.

So stellen Sie das Frequenzspektrum dar:

```
f = linspace(0, fs/2, n/2+1);
plot(f, abs(Y(1:n/2+1)))
```

Die Grafik zeigt einige deutliche, scharf begrenzte Frequenzspitzen im Bereich bis etwa 4000 Hz. Andererseits gibt es oberhalb von etwa 8000 Hz keine nennenswerten Beiträge. Es bringt nichts, bis zur Nyquist-Frequenz, hier 22 050 Hz, zu zeichnen. Zoomen Sie in den interessanten Frequenzbereich (interaktiv oder mit dem Befehl: `axis([0 8000 0 max(abs(Y))])`.



Aufgabe 92: Klang einer Gitarren-Saite

Die Datei `klang2.wav` (siehe Übungshomepage (klick!)) enthält eine kurze Audio-Aufnahme einer gezupften Saite. Das Klangspektrum besteht aus dem Grundton und einer Reihe von Obertönen. Analysieren Sie das Frequenzspektrum, stellen Sie das Frequenzspektrum im Bereich bis 5000 Hz dar. Üblich ist eine semilogarithmische Darstellung,

```
f = linspace(0, fs/2, n/2+1);
index=find(f<5000);
semilogy(f(index), abs(Y(index)))
```

Beantworten Sie folgende Fragen:

- Welche Frequenz liefert den stärksten Beitrag?
- Was ist die Frequenz der Grundschwingung? (Das ist die Tonhöhe, die ein Musiker hört.)
- Bis zu welcher Frequenz lassen sich Oberschwingungen deutlich erkennen?

Aufgabe 93: Ebbe und Flut

Dieses Beispiel soll zeigen: Fourier-Analyse kann in einem Datensatz periodische Effekte erkennen und aus dem Hintergrund-Rauschen herausfiltern.

Der Meeresspiegel schwankt aufgrund der Gezeiten (Ebbe und Flut). Dabei überlagern sich verschiedene astronomische und geographische Effekte mit unterschiedlichen Frequenzen (oder entsprechenden Perioden). Wind und Wetter sorgen für zusätzliche, unvorhersehbare Schwankungen. Die Theorie sagt (unter anderm) folgende Komponenten voraus:

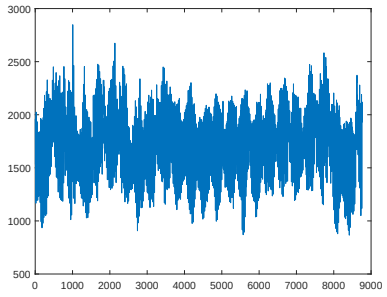
Bezeichnung	Name	Periode (in Stunden)
M2	halbtägige Hauptmondtide	12.4206012
S2	halbtägige Hauptsonnentide	12
N2	große elliptische Tide 1. Ordnung zu M2	12.65834751
K1	eintägige Hauptdeklinationstide	23.93447213
O1	eintägige Hauptmondtide	25.81933871
M4	Flachwasser-Oberschwingung zu M2	6.210300601
M6	Flachwasser-Oberschwingung zu M2	4.140200401

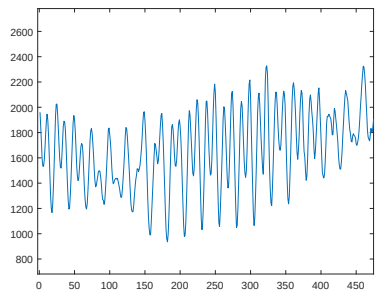
Der Datensatz `Triest2014.dat` auf der Übungs-Homepage enthält für das Jahr 2014 stündlich gemessene Werte des Meeresspiegels am Pegel im Hafen von Triest.
(Quelle: `http://uhslc.soest.hawaii.edu/data/download/fd#uh829`).
Fourier-Analyse kann die verschiedenen periodischen Anteile herausfiltern.

Lesen Sie die Daten ein.

```
X=load('Triest2014.dat');
n=length(X);              % Anzahl Datenpunkte
fs = 1;                   % Abtast-Frequenz 1/Stunde
```

Ein Plot des Datenvektors verschafft Ihnen einen ersten Überblick. Wenn Sie geeignet hineinzoomen, können Sie ungefähr 12- und 24-stündig periodische Schwankungen deutlich erkennen. (Zeiteinheit auf x-Achse ist 1 Stunde!) Sie sehen aber auch, dass sich anscheinend verschiedened kurz- und längerfristige Schwankungen überlagern. Die Fourier-Analyse entdeckt die einzelnen periodischen Schwingungen, deren Überlagerung den Datenvektor erzeugt.





Wichtige Größe ist die Abtastfrequenz f_s (*sampling rate, sampling frequency*, daher das s im Subskript). Bei Zeiteinheit Stunden und einem Messwert pro Stunde ist hier einfach

$$f_s = 1, \qquad \text{(Einheit 1/h)}$$

Führen Sie die Fourier-Transformation durch und berechnen Sie die entsprechenden Frequenzen:

```
Y = fft(X);
f = linspace(0, fs/2, n/2+1);
```

Der Term `Y(1)/N` ist der Mittelwert aller Daten, also der mittlere Pegelstand. Die Terme `abs(Y(2))` bis `abs(Y(N/2+1))` entsprechen den Amplituden harmonischer Schwingungen mit Frequenzen (von f_s/N bis zur halben Abtastfrequenz $f_s/2$ in Schritten f_s/N).

Bei diesen Datensatz ist nicht die Frequenz, sondern die Periode aussagekräftiger. Es sind Schwingungen mit Perioden bis etwa 30 Stunden von Interesse. Längperiodische Vorgänge beschreiben Schwankungen über Tage oder Monate; die wollen wir hier nicht untersuchen.

Berechnen Sie zum Frequenz-Vektor `f` den entsprechenden Perioden-Vektor `T`. Filtern Sie aus dem T-Vektor den relevanten Bereich und zeichnen Sie die zugehörigen Amplituden.

```
T = 1./f;
index = find(T<30);
plot(T(index), abs(Y(index)))
```

Aus dem Diagramm können Sie zum Beispiel ablesen:

- Die Schwingungen mit welcher Periode liefert den stärksten, zweit- und drittstärksten Beitrag?
- In welchem Verhältnis stehen die Amplituden der Hauptmondtide M2 und der Hauptsonnentide S2?
- Lassen sich die Oberschwingungen M4 und M6 in den Daten nachweisen?

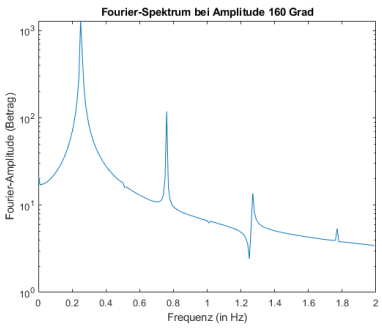
Aufgabe 94: Pendel mit großer Amplitude

Die Differentialgleichung des mathematischen Pendels lautet

$$\ddot{\phi} + \frac{g}{\ell} \sin \phi = 0$$

mit $\phi : t \mapsto \phi(t)$ die Winkelauslenkung (in Radiant) als Funktion der Zeit t (in Sekunden); Fallbeschleunigung g in m/s^2; Pendellänge ℓ in m.

Wählen Sie Werte für ℓ und g und berechnen Sie die Lösung $\phi(t)$ an 1000 äquidistanten Datenpunkten im Bereich $0 < t \leq 100$ für Amplituden 45°, 90°, 170°. Zeichnen Sie jeweils ein Frequenzspektrum der Lösung.
Ihre Zeichnungen könnten etwa so aussehen, wie hier rechts gezeigt. Sie sollten aus den Graphiken ablesen können:



- Die erste Frequenzspitze (Grundfrequenz) liegt in der Nähe von

$$f_0 = \frac{1}{2\pi} \sqrt{\frac{g}{\ell}}$$

(Das ist die klassische Formel für die Schwingungsfrequenz des Pendels bei kleinen Amplituden.)

- bei größerer Anfangsauslenkung verringert sich die Schwingungsfrequenz.
- Je größer die Anfangsauslenkung, desto mehr treten auch Frequenz-Spitzen bei höheren Frequenzen auf (Oberschwingungen).
- Die Frequenzen der Oberschwingungen sind ganzzahlige Vielfache der Grundfrequenz.
- Im Idealfall wären die Frequenz-Spitzen scharfe Zacken an jeweils einem Frequenzwert. Tatsächlich verteilen sich die Frequenzen im Signal über einige benachbarte Datenpunkte im Fourier-Spektrum (Leck-Effekt).

Geben Sie für Amplitude 170° an:

1. Um wieviel Prozent höher ist die Schwingungsdauer, verglichen mit der linearen Näherung für kleine Amplituden?
2. In welchem Verhältnis steht die Amplitude der ersten Oberschwingung zur Amplitude der Grundfrequenz?

(Diese Resultate sind unabhängig von Ihrer konkreten Wahl für ℓ und g.)

Ü 11.3 Aliasing und Leakage

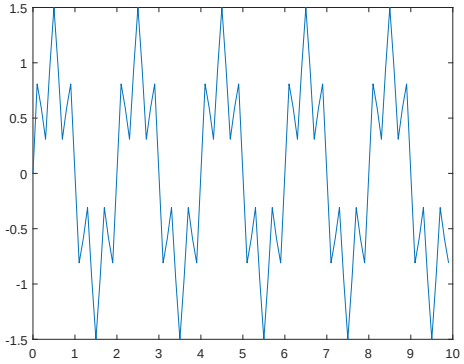
Die Fourieranalyse eines Signals, für das Daten nur zu endlich vielen diskreten Zeitpunkten vorliegen, kann nicht alle im kontinuierlichen, unbegrenzt andauernden Signal vorhandenen Frequenzen exakt erkennen. Zwei wesentliche Einschränkungen sind Aliasing und Leakage. Dazu stellt dieser Abschnitt Beispiele vor.

Die folgenden Befehle erzeugen einen Datenvektor, der das Signal

$$X(t) = \sin(\pi t) + \frac{1}{2}\sin(5\pi t)$$

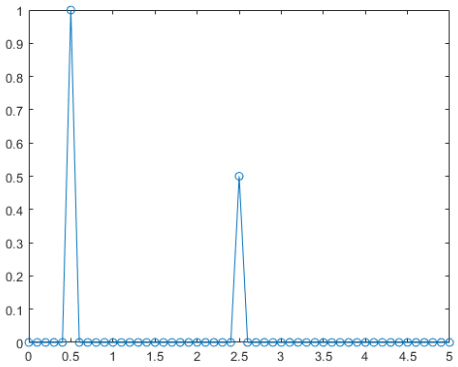
mit $n = 100$ Datenpunkten und Abtastfrequenz $f_s = 10\,\text{Hz}$ speichert.

```
fs=10;
n = 100;
t = linspace(0,(n-1)/fs,n);
X = sin(pi*t) + 1/2*sin(5*pi*t);
```



Fourier-Analyse zeigt korrekt die beiden Frequenzen 0,5 Hz und 2,5 Hz an. Auch die entsprechenden Amplituden stimmen, weil im Plot mit $2/n$ skaliert wurde).

```
%% Fourier-Transformation
Y = fft(X);
%% Frequenz-Achse, Frequenzspektrum zei
f = linspace(0, fs/2, n/2+1);
index=find(f<10000);
plot(f(index), abs(Y(index))*2/n,'-o')
```



Diese Darstellung (scharfe Frequenzpeaks an den korrekten Stellen) ist aber nicht der Normalfall. Die nächste Aufgabe zeigt: Frequenzen könnenn an der falschen Stelle angezeigt (Aliasing) oder unscharf verschmiert (Leakage) sein.

Aufgabe 95: Verschwundene Frequenzen, fälschlich versteckt oder verschmiert

Erzeugen Sie Datenvektoren, die höhere Signalfrequenzen enthalten, zum Beispiel

$$X(t) = \sin(2\pi t) + \frac{1}{2}\sin(12\pi t)$$

oder

$$X(t) = \cos(4\pi t) + \cos(9\pi t)$$

oder

$$X(t) = \cos(11\pi t) + \sin(15\pi t)$$

Welche Signalfrequenzen liefert die Fourieranalyse? Wo stimmen die Ergebnisse mit den Signalfrequenzen überein, wo nicht?

> **Aliasing** ist der Begriff für Fehler, die auftreten, wenn das Signal $X(t)$ Frequenzanteile grösser oder gleich der Nyquist-Frequenz, das ist die halbe Abtastfrequenz, enthält.
>
> $$\text{Nyquist Frequenz } f_{\text{Ny}} = \frac{f_s}{2}$$
>
> Höhere Frequenzanteile interpretiert die Fourier-Analyse fälschlicher Weise als Frequenzen unterhalb der Nyquist-Frequenz. (Diese hohen Frequenzen treten also unter „Alias-Namen“ als niedrige Frequenzen auf.)

Schlagen Sie in Wikipedia unter „Alias-Effekt“ nach, dort finden Sie sehr anschauliche Darstellungen!

Die Datenpunkte im Spektrogramm einer diskreten Fouriertransformation haben Abstand f_s (Abtastfrequenz) auf der Frequenzachse. In unserem Beispiel: 0,1 Hz. Wenn die Signalfrequenz nicht exakt mit so einem diskreten Frequenzwert übereinstimmt, zeigt das Spektrogramm keine scharfen Spitzen. Die Signalfrequenz verteilt sich über mehrere benachbarte Datenpunkte. Sie „sickert“ oder „leckt“ zu benachbarten Datenpunkten.

Ein Beispiel ist ein Signal mit Frequenz $f = \frac{8}{3}$, zum Beispiel

$$X(t) = \cos(\frac{16\pi}{3}t);$$

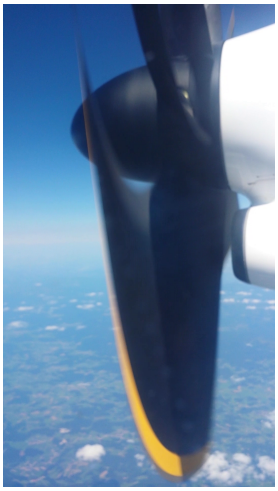
Stellen Sie wie vorhin das Frequenzspektum dar. Welcher Frequenz entspricht die Spitze im Diagramm? Welche Frequenzen treten als die beiden nächststärksten Beiträge auf?

> **Leakage-Effekt**: Fourieranalyse eines diskreten Datenvektors liefert nur in Spezialfällen – Abtastfrequenz ein ganzzahliges Vielfaches der Signalfrequenz – scharfe Frequenz-Datenpunkte. Ein sinusförmiges Signal verteilt sich im Allgemeinen auf mehrere benachbarte Frequenz-Datenpunkte.

Leakage kann auch als eine Form von Aliasing gesehen werden: Die korrekte Frequenz tritt im Spektrogramm zum Teil als benachbarte Frequenzen auf.

Die Abbildung zu Aufgabe 94 lässt auch ein Leakage-Phänomen erkennen: der erste Peak ist nicht so nadelscharf, wie er theoretisch sein sollte. Beim dritten Peak zeigt sich ein Ausreißer nach unten. Diese Störeffekte gehen hauptsächlich auf die endliche Signaldauer und Abtastfrequenz zurück.

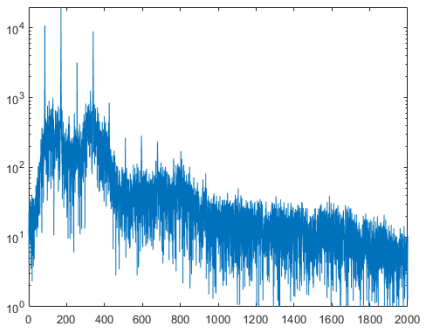
Aufgabe 96: Über den Wolken



Wenn Sie ein Flugticket mit Fensterplatz billig buchen, kann es sein, dass der Blick nach draußen so aussieht. In der Kabine ist es dann auch entsprechend laut. Die kurze Videosequenz `propeller.mp4` (Download von der Übungs-Homepage) vermittelt einen Eindruck davon.
Auf dem Video dreht sich der Propeller scheinbar recht langsam. Wenn Sie genau hinsehen und auf den Anstellwinkel der Propellerblätter achten, dreht er sich sogar rückwärts! Keine Panik, das sind Aliasing-Effekte. Die tatsächliche Propellerdrehzahl können Sie aus dem Betriebsgeräusch durch Fourieranalyse ermitteln.
Der Befehl

```
[y,fs] = audioread('propeller.mp4')
```

liest das Audiosignal ein.



Stellen Sie das Frequenzspektrum im Bereich 0–2000 Hz dar. In semilogarithmischer Darstellung erkennen Sie einige scharfe Peaks über einem breitbandigem Hintergrundrauschen.
Die Peaks entsprechen den Fourier-Komponenten des Signals der rotierenden Propellerblätter: Grundfrequenz und Vielfache davon. Das Triebwerk ist ein Pratt & Whitney Canada Corp PW 150 mit maximaler Propellerdrehzahl 1200 rpm. Der Propeller ist sechsstrahlig, die Grundfrequenz des Signals ist daher die sechsfache Drehzahl.

Welche Drehzahl (in rpm, Umdrehungen pro Minute) können Sie aus dem Signal bestimmen?

Das Video zeigt 29,9 Einzelbilder pro Sekunde. Um welchen Winkel dreht sich ein Propellerblatt von einem Bild zum nächsten? Um welchen Winkel dreht sich der sechsstrahlige Propeller *scheinbar* aufgrund des Alias-Effektes?

Ü 11.4 Filtern von Signalen im Fourier-Raum

Beispiele zum Filtern von Datenvektoren und Datenfeldern.

Aufgabe 97: Bearbeiten von Audio-Signalen im Fourier-Raum

Lesen Sie die Datei `klang2.wav` ein und lassen Sie sie abspielen:

```
[X,fs] = audioread('klang2.wav');
n = length(X);
sound(X,fs)
```

Wenden Sie FFT auf das Signal `X` an.

```
Y = fft(X);
```

Sie sollen nun das Signal im Frequenzraum bearbeiten. Das bearbeitete Signal können Sie rücktransformieren und das Ergebnis abhören.

Zuordnung Index k im Y-Vektor $\mapsto$ Frequenz f_k im Frequenzraum:

$$f_k = \frac{k-1}{n} f_s \quad \text{für } k \leq \frac{n}{2}$$

Sie können Komponenten in Y verstärken oder schwächen und damit die entsprechenden Frequenzen im Audiosignal beeinflussen.

Ein Filter-Vektor F multipliziert ein Signal Y im Fourierraum:

$$Y_{\text{gefiltert}} = Y \cdot F$$

Für einen reelllen Signalvektor X und dessen Fouriertransformierte Y müssen die Komponenten $F(k)$ des Filtervektors F für $k > 1$ symmetrisch bezüglich Index $n/2+1$ liegen. In MATLABs Schreibweise:

```
F(n/2+2:n) = F(n/2:-1:2);
```

Definieren Sie dazu verschiedene Filter-Funktionen.

- Tiefpass-Filter: lässt nur Frequenzen bis zu einer bestimmten Grenzfrequenz f_{cut} durch; höhere Frequenzen werden unterdrückt. Hier zum Beispiel ein Tiefpass-Filtervektor mit Grenzfrequenz 200 Hz:

  ```
  Filt = ones(size(Y));
  fcut=200;
  ncut=round(fcut/fs*n)+1; % Index im Y-Vektor
  Filt(ncut:n/2) = 0;
  % Filter muss symmetrisch um n/2+1 liegen
  Filt(n/2+2:n) = Filt(n/2:-1:2);
  ```

  Wenden Sie diesen Filter an, transformieren Sie das Signal zurück vom Frequenz- in den Zeitbereich und hören Sie sich den Unterschied an:

  ```
  Y1 = Y.*Filt;
  X1 = ifft(Y1);
  sound(X1,fs)
  ```

Das Original-Signal war der Klang einer tiefen A-Saite auf der Gitarre, Grundton 110 Hz. Ein Tiefpass-Filter mit 200 Hz schneidet sämtliche Obertöne ab. Das gefilterte Signal klingt deswegen etwas dumpf und leer.

Probieren Sie aus: wenn Sie Grenzfrequenz 100 Hz wählen, hören Sie gar keinen Ton mehr, weil nun auch die Grundfrequenz weggefiltert wird. Wenn Sie höhere Grenzfrequenzen wählen, klingt das gefilterte Signal immer ähnlicher dem Ausgangssignal.

- Hochpass-Filter: lässt nur Frequenzen ab einer bestimmten Grenzfrequenz f_{cut} durch; tiefere Frequenzen werden unterdrückt.

  Probieren Sie aus: Je höher Sie die Grenzfrequenzen wählen, desto schärfer klingt das gefilterte Signal.

- Bandpass-Filter: lässt nur Frequenzen innerhalb eines Bereiches $f_{\text{min}} < f < f_{\text{max}}$ durch.

- Sie können auch wildere Filter-Funktionen wählen. Achten Sie aber darauf, dass der Filter-Vektor symmetrisch um `n/2+1` liegen muss: `Filt(n/2+2:n) = Filt(n/2:-1:2);`. (Andernfalls enthält das rücktransformierte Signal imaginäre Anteile!) Auch müssen Filter- und Datenvektor die gleiche Gestalt (beide jeweils Zeilen- oder Spalten-Vektoren) sein; andernfalls führt die Anwendung der Filterfunktion `Y.*Filt` zu Fehlermeldungen.

- Wenn Ihnen das Erstellen eines symmetrische Filter-Vektors zu kompliziert ist, können Sie der Einfachheit halber beim rücktransformierten Signal den Imaginärteil abschneiden. Sie schreiben dann statt `X1 = ifft(Y1);` eben `X1 = real(ifft(Y1));` Das ist zwar eine grobe Vorgangsweise, aber sie funktioniert.

  Das Signal von Datei `klang2.wav` ist allerdings nicht sehr komplex. Besser hören Sie den Einfluss verschiedener Filter zum Beispiel, wenn Sie die Aufnahme einer menschlichen Stimme bearbeiten.

Aufgabe 98: Bearbeiten von Bild-Dateien im Fourier-Raum

Die folgenden Befehle lesen eine Bilddatei ein und zeigen eine Schwarzweiß-Version davon an:

```
% Einlesen einer Bilddatei
photoarray = imread('katzekatze.jpg');

% Eine Standard-Formel rechnet RGB-Werte auf Grautöne um
redpix = single(photoarray(:,:,1))/255;
greenpix = single(photoarray(:,:,2))/255;
bluepix = single(photoarray(:,:,3))/255;
graypix = 0.299*redpix + 0.587*greenpix + 0.114*bluepix;
graypix = rot90(graypix,2); % so wird Foto seitenrichtig angezeigt...

figure(1)
imagesc(graypix)
colormap(gray), axis off image

% Fouriertransformierte
Y = fft2(graypix);
```

Der Befehl `fft2` führt eine 2-dimensionale Fouriertransformation des Datenfeldes durch. Sie können nun im Fourier-Raum gezielt Frequenzen verstärken oder abschwächen. Wenn Sie nur die nierigsten Frequenz-Anteile behalten (Tiefpass-Filter), wird das Bild unscharf und weichgezeichnet:

```
% Filter Tiefpass
% Aanzahl Fouriermoden, die bleiben
% n=8;  % fast nicht erkennbar
% n=16; %  erkennbar, sehr unscharf
 n=30; % unscharf
% n=50; % unscharf
% n=100 % maessig gut
% n=200 % ganz gut!

 Y(n:end-n,:) = 0;
 Y(:,n:end-n) = 0;

y = real(ifft2(Y));
figure(3)
imagesc(y)
colormap(gray), axis off image
```

Wenn Sie Fourier-Komponenten ändern, enthält das rücktransformierte Signal imaginäre Anteile. Nur wenn die Filter-Funktion gewisse Symmetrie-Eigenschaften hat, ist auch das rücktransformierte Signal reell. Bei der eindimensionalen FFT des Audio-Signals haben wir das berücksichtigt. Im zweidimensionalen Fall ist die Symmetrie-Bedingung komplizierter. Wir schneiden deswegen der Einfachheit halber beim rücktransformierten Signal den Imaginärteil ab. (Das ist eine *Hau-Ruck*-Methode; sie funktioniert, aber die Signal-Energie insgesamt wird um den Anteil der abgeschnittenen Imaginärteile verringert.)

Sie können auch wildere Filter-Funktionen ausprobieren und sehen, wie sich Änderungen im Fourier-Raum auf das Bild auswirken.