

Numerische Methoden

Prof. Dr. Jürgen Koch

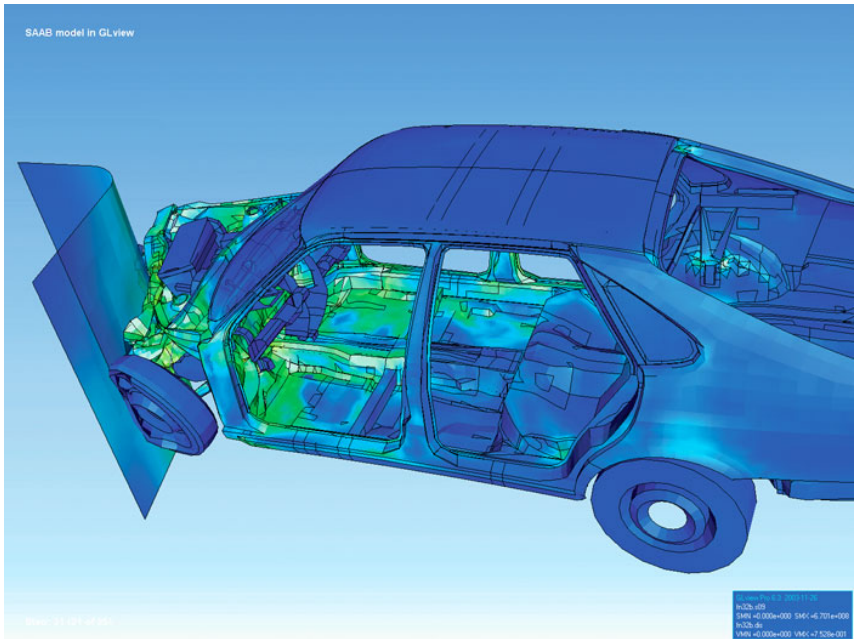
Inhaltsverzeichnis

I. Lineare Gleichungssysteme	1
1. Gauss-Algorithmus	3
1.1. Rundungsfehler	3
1.2. Pivotieren	5
1.3. Nachiteration	6
1.4. Eigenschaften	6
2. Iterative Algorithmen	7
2.1. Iterationsverfahren	8
2.2. Jacobi-Iteration	8
2.3. Gauss-Seidel-Iteration	13
3. Lineare Algebra Software	16
4. Übungen	17
II. Nicht lineare Gleichungen und Gleichungssysteme	21
1. Intervallhalbierung	22
2. Fixpunktiteration	24
3. Newton-Verfahren	31
4. Newton-Verfahren für Systeme	34
5. Global Positioning System (GPS)	43
5.1. Das GPS-Prinzip	43
5.2. Positionsbestimmung	44
5.3. Laufzeitfehler	45

5.4.	Newton–Verfahren für GPS	45
6.	Übungen	48
III.	Interpolation und Approximation	53
1.	Polynomeninterpolation	53
2.	Newton–Schema	56
3.	Hermite–Interpolation	58
4.	Ausgleichspolynome	58
IV.	Numerische Integration	65
1.	Problemstellung	65
2.	Trapezregel	67
3.	Rombergverfahren	70
4.	Übungen	74
V.	Gewöhnliche Differentialgleichungen	77
1.	Numerische Differentiation	77
2.	Anfangswertprobleme	78
3.	Euler–Verfahren	79
4.	Globaler und lokaler Fehler	82
5.	Schrittweitensteuerung	82
6.	Übungen	83

I Lineare Gleichungssysteme

Viele numerische Verfahren, die wir in den folgenden Kapiteln noch genauer kennenlernen werden, basieren auf der Lösung von linearen Gleichungssystemen. Aus China sind Verfahren bekannt, mit denen Gleichungssysteme bereits vor über 2000 Jahren systematisch gelöst wurden. Uns geht es an dieser Stelle aber nicht darum die Lösung für Gleichungssysteme mit zwei oder drei Unbekannten zu bestimmen, sondern Gleichungssysteme mit sehr vielen Unbekannten. Ein typisches Beispiel, bei dem oft mehr als eine Million Gleichungen zu lösen sind, stellt die Simulation eines Crash-Tests in der Automobilindustrie dar.



Zur Beschreibung linearer Gleichungssysteme eignet sich die Matrix-Vektor-Notation

$$\underline{A} \underline{x} = \underline{b}.$$

Bei vorgegebener Matrix \underline{A} und rechter Seite \underline{b} sucht man den Lösungsvektor \underline{x} . Wir beschränken uns bei unserer Betrachtung auf lineare Gleichungssysteme mit einer quadratischen Matrix, die eindeutig lösbar sind.

1. Gauss–Algorithmus

Beispiel I.1 (Gauss–Algorithmus)

Das lineare Gleichungssystem

$$\begin{aligned}3x_1 + 6x_2 &= 6 \\x_1 + 7x_2 + 5x_3 &= 17 \\2x_1 + 4x_2 - 8x_3 &= -12\end{aligned}$$

soll mit dem Gauss–Algorithmus gelöst werden.

Gauss–Algorithmus

```
function x = gauss(A,b)
n = size(A,1);
for i=1:n-1
    b(i+1:n) =b(i+1:n) -A(i+1:n,i)/A(i,i)*b(i);
    A(i+1:n,:) =A(i+1:n,:) -A(i+1:n,i)/A(i,i)*A(i,:);
end
x = zeros(n,1);
for j=n:-1:1
    x(j) = (b(j) - A(j,j+1:n)*x(j+1:n))/A(j,j);
end
```

1.1. Rundungsfehler

Beispiel I.2 (Gauss–Algorithmus)

$$\begin{aligned}
 A &= [10 \ -7 \ 0; -3 \ 2.099 \ 6; \ 5 \ -1 \ 5] \\
 b &= [7; \ 3.901; \ 6] \\
 x &= \text{gauss}(A, b), \text{ pause}
 \end{aligned}$$

Beispiel I.3 (Rundungsfehler beim Gauss-Algorithmus)

$$\begin{array}{ccc|c}
 10.000 & -7.000 & 0.000 & 7.000 \\
 -3.000 & 2.099 & 6.000 & 3.901 \\
 5.000 & -1.000 & 5.000 & 6.000
 \end{array} \Rightarrow x = \begin{pmatrix} -0.000000000000000031 \\ -1.000000000000000044 \\ 1.0000000000000000 \end{pmatrix}$$

Vertauschen der zweiten und dritten Zeile ergibt:

$$\begin{array}{ccc|c}
 10.000 & -7.000 & 0.000 & 7.000 \\
 5.000 & -1.000 & 5.000 & 6.000 \\
 -3.000 & 2.099 & 6.000 & 3.901
 \end{array} \Rightarrow x = \begin{pmatrix} 0.0000000000000000 \\ -1.0000000000000000 \\ 1.0000000000000000 \end{pmatrix}$$

Rundungsfehler beim Gauss-Algorithmus

Aus mathematischer Sicht kann man die Zeilen bei einem linearen Gleichungssystem beliebig vertauschen. Aus Sicht der Numerischen Mathematik hat die Anordnung der Zeilen jedoch Einfluss auf den Rundungsfehler. Das Ziel ist es, den Gauss-Algorithmus so zu verändern, dass der Rundungsfehler möglichst klein bleibt.

1.2. Pivotieren

Beispiel 1.4 (Pivotieren beim Gauss-Algorithmus)

- (A) Suche das betragsmäßig größte Element (C) Vertausche erste und letzte Spalte

$$\begin{array}{ccc|c} x_1 & x_2 & x_3 & b \\ \hline 3 & 6 & 0 & 6 \\ 1 & 7 & 5 & 17 \\ 2 & 4 & -8 & -12 \end{array}$$

$$\begin{array}{ccc|c} x_3 & x_2 & x_1 & b \\ \hline -8 & 4 & 2 & -12 \\ 5 & 7 & 1 & 17 \\ 0 & 6 & 3 & 6 \end{array}$$

- (B) Vertausche erste und letzte Zeile

$$\begin{array}{ccc|c} x_1 & x_2 & x_3 & b \\ \hline 2 & 4 & -8 & -12 \\ 1 & 7 & 5 & 17 \\ 3 & 6 & 0 & 6 \end{array}$$

Pivotieren beim Gauss-Algorithmus

- Die Grundidee beim Pivotieren besteht darin Zeilen und Spalten so zu vertauschen, dass die betragsmäßig grössten Koeffizienten nach Möglichkeit in der Diagonal des Dreieckschemas stehen.
- Pivotieren verringert den Einfluss von Rundungsfehlern.
- Totale Pivotelementsuche erfordert hohen Rechenaufwand, deshalb verwendet man in der Praxis häufig nur Spaltenpivotelementsuche oder Diagonalpivotelementsuche.

1.3. Nachiteration

Beispiel 1.5 (Nachiteration beim Gauss-Algorithmus)

(A) Löse $\underline{A}\underline{x} = \underline{b}$ mit Gauss-Algorithmus

$$\underline{x} = \begin{pmatrix} -0.000000000000031 \\ -1.000000000000044 \\ 1.000000000000000 \end{pmatrix}$$

(C) Löse $\underline{A}\underline{dx} = \underline{r}$ mit Gauss-Algorithmus

$$\underline{dx} = \begin{pmatrix} 0.000000000000031 \\ 0.000000000000044 \\ 0.000000000000000 \end{pmatrix}$$

(B) Berechne Residuum $\underline{r} = \underline{b} - \underline{Ax}$

$$\underline{r} = \begin{pmatrix} 0.000000000000000 \\ 0.000000000000000 \\ 0.000000000000111 \end{pmatrix}$$

(D) Verbesserte Lösung $\tilde{\underline{x}} = \underline{x} + \underline{dx}$

$$\tilde{\underline{x}} = \begin{pmatrix} 0.000000000000000 \\ -1.000000000000000 \\ 1.000000000000000 \end{pmatrix}$$

1.4. Eigenschaften

Eigenschaften des Gauss-Algorithmus

- Funktioniert für beliebige $n \times m$ Matrizen.
- Pivotieren und Nachiteration verkleinern Rundungsfehler.
- Rechenoperationen $\frac{n(n+1)(2n+1)}{6} - n$
- L-R-Zerlegung (Dreieckszerlegung)

$$\underline{P} \underline{A} = \underbrace{\begin{bmatrix} 1 & 0 & \cdots & 0 \\ l_{21} & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \cdots & 1 \end{bmatrix}}_{\underline{L}} \underbrace{\begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ 0 & 0 & \cdots & r_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & r_{nn} \end{bmatrix}}_{\underline{R}}$$

2. Iterative Algorithmen

Einfache mathematische Probleme kann man direkt durch geeignete Formeln lösen. Beispielsweise lassen sich die Lösungen einer quadratischen Gleichung durch die so genannte Mitternachtsformel berechnen.

Für viele mathematische Probleme kann man das Ergebnis nicht in geschlossener Form angeben. In diesen Fällen versucht man die Lösung mit einem iterativen Algorithmus zu berechnen. Bei der Iteration wird ein und die selbe Berechnungsvorschrift wiederholt angewendet. Das Ergebnis des letzten Iterationsschrittes verwendet man dabei als Ausgangswert für den nächsten Iterationsschritt.

Bei jedem einzelnen Verfahren muss man untersuchen, ob das Verfahren im Grenzwert gegen eine Lösung des Problems konvergiert. Für die praktische Bedeutung ist die Geschwindigkeit der Konvergenz ausschlaggebend.

2.1. Iterationsverfahren

Iterations–Verfahren

Ein Iterationsverfahren besteht aus drei Teilen: Startwert, Iterationsvorschrift und Abbruchkriterium. Die Iterationsvorschrift legt fest, wie man aus einem bereits bekannten Näherungswert einen neuen Näherungswert berechnet. Die Iterationsvorschrift wird zunächst auf den Startwert angewendet. Im zweiten Schritt verwendet man das Ergebnis der ersten Berechnung als neuen Startwert. Dieser Ablauf wird wiederholt bis das Abbruchkriterium erfüllt ist.

Bei Anwendungen aus der Praxis treten oft Gleichungssysteme mit sehr vielen Unbekannten auf. In der Regel ist die Matrix dann aber dünn besetzt, d.h. in jeder Zeile sind nur wenige Elemente von Null verschieden und alle Gleichungen haben eine ähnliche Bauart. Solche Gleichungssysteme lassen sich mit Iterativen–Algorithmen wesentlich besser lösen als mit dem Eliminationsverfahren von Gauss.

2.2. Jacobi–Iteration

Die Jacobi–Iteration beruht auf einer einfachen Idee. Wir lösen die erste Gleichung nach x_1 auf, die zweite nach x_2 , usw. und berechnen so iterativ Näherungslösungen.

Jacobi-Iteration (Gesamtschrittverfahren)

Zur Berechnung der Lösung eines linearen Gleichungssystems löste man die i -te Gleichung nach der i -ten Variable auf. Die neuen Näherungswerte für die Lösung berechnet man iterative aus den alten Näherungswerten.

Beispiel 1.6 (Jacobi-Iteration)

Das lineare Gleichungssystem

$$\begin{aligned}3x_1 + 6x_2 &= 6 \\x_1 + 7x_2 + 5x_3 &= 17 \\2x_1 + 4x_2 - 8x_3 &= -12\end{aligned}$$

soll mit der Jacobi-Iteration gelöst werden. Durch Auflösen der ersten Gleichung nach x_1 , der zweiten Gleichung nach x_2 und der dritten Gleichung nach x_3 erhält man die Iterationsvorschrift

$$\begin{aligned}x_1^{(neu)} &= \frac{1}{3} \left(6 - 6x_2^{(alt)} \right), \\x_2^{(neu)} &= \frac{1}{7} \left(17 - x_1^{(alt)} - 5x_3^{(alt)} \right), \\x_3^{(neu)} &= -\frac{1}{8} \left(-12 - 2x_1^{(alt)} - 4x_2^{(alt)} \right).\end{aligned}$$

Wenn wir mit den Werten

$$x_1^{(0)} = 0, \quad x_2^{(0)} = 0, \quad x_3^{(0)} = 0,$$

starten, erhalten wir im ersten Schritt

$$x_1^{(1)} = 2, \quad x_2^{(1)} = \frac{17}{7}, \quad x_3^{(1)} = \frac{3}{2}.$$

Der zweite Schritt liefert

$$x_1^{(2)} = -\frac{20}{7}, \quad x_2^{(2)} = \frac{15}{14}, \quad x_3^{(2)} = \frac{45}{14}.$$

Nach ungefähr 92 Iterationsschritten erhalten wir die Lösung

$$x_1^{(92)} \approx 0, \quad x_2^{(92)} \approx 1, \quad x_3^{(92)} \approx 2,$$

mit zwölfstelliger Genauigkeit.

Iterative-Algorithmen zur Lösung linearer Gleichungssysteme sind in der Regel einfach zu implementieren. Insbesondere bei dünn besetzten Matrizen, die eine einfache Struktur aufweisen, ist auch bei Gleichungssystemen mit vielen Unbekannten eine Implementierung mit ein paar Zeilen Code möglich.

Beispiel 1.7 (Jacobi-Iteration Implementierung)

```
void main()
{
    double xalt[] = {0.0, 0.0, 0.0};
    double xneu[3];

    for (int i=0; i < 100; ++i)
    {
        xneu[0] = ( 6-6*xalt[1])/3.0;
        xneu[1] = ( 17- xalt[0]-5*xalt[2])/7.0;
        xneu[2] = (-12-2*xalt[0]-4*xalt[1])/(-8.0);
        print(i, xneu);
        copy(xneu, xalt);
    }
}
```

Zwei Fragen sind bisher noch nicht geklärt: Bei welchen linearen Gleichungssystemen konvergiert die Jacobi-Iteration gegen die Lösung des linearen Gleichungssystems und wie schnell konvergiert das Verfahren?

Bei vielen praktischen Problemen kann man die Konvergenzfrage mit dem Begriff der Diagonaldominanz einer Matrix zufriedenstellend klären.

Definition 1.1 (Diagonaldominante Matrix) *Eine Matrix A mit den Elementen a_{ki} heisst Diagonal dominant, falls die Summe der Beträge der nicht Diagonalelemente in jeder Zeile echt kleiner ist als der Betrag des Diagonalelementes*

$$\sum_{i \neq k} |a_{ki}| < |a_{kk}|.$$

Das linearer Gleichungssysteme

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

kann man mit der Jacobi-Iteration lösen, falls die Matrix \underline{A} Diagonal dominant ist. Der Umkehrschluss gilt aber nicht. Es gibt durchaus lineare Gleichungssysteme, bei denen die Matrix \underline{A} nicht Diagonal dominant ist, aber die Jacobi-Iteration trotzdem konvergiert.

Für die mathematisch exakte Beantwortung der Konvergenzfrage verwendet man die Zerlegung der Matrix \underline{A} in Dreiecksmatrizen.

Dreieckszerlegung

Jede Matrix \underline{A} lässt sich in eine untere Dreiecksmatrix \underline{L} , eine Diagonalmatrix \underline{D} und eine obere Dreiecksmatrix \underline{U} zerlegen:

$$\underbrace{\begin{bmatrix} 0 & 0 & \cdots & 0 \\ a_{21} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & 0 \end{bmatrix}}_{\underline{L}} + \underbrace{\begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{bmatrix}}_{\underline{D}} + \underbrace{\begin{bmatrix} 0 & a_{12} & \cdots & a_{1n} \\ 0 & 0 & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}}_{\underline{U}}.$$

Mit Hilfe der Dreieckszerlegung kann man die Jacobi-Iteration in Matrixform formulieren. Dazu schreiben wir das lineare Gleichungssystem in der Form

$$(\underline{L} + \underline{D} + \underline{U}) \underline{x} = \underline{b}.$$

Nun lösen wir diese Gleichung nach den Diagonalelementen auf

$$\underline{D} \underline{x}^{(\text{neu})} = \underline{b} - (\underline{L} + \underline{U}) \underline{x}^{(\text{alt})}.$$

Falls alle Elemente auf der Diagonale der Matrix \underline{A} ungleich Null sind, kann man die inverse der Diagonalmatrix \underline{D}^{-1} aus den Kehrwerten der Diagonalelemente berechnen

$$\underline{x}^{(\text{neu})} = \underline{D}^{-1} (\underline{b} - (\underline{L} + \underline{U}) \underline{x}^{(\text{alt})}).$$

Ausschlaggebend für die Konvergenz der Iteration ist der sogenannte Spektralradius der Matrix

$$-\underline{D}^{-1} (\underline{L} + \underline{U}).$$

Als Spektralradius bezeichnet man den betragsmäßig größten Eigenwert der Matrix. Auch die Konvergenzgeschwindigkeit lässt sich mit Hilfe dieses

Spektralradius bestimmen. Um so kleiner der Spektralradius ist, um so schneller konvergiert die Iteration.

Eigenschaften der Jacobi-Iteration

- Die Lösung des linearen Gleichungssystems

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

kann man mit der Jacobi-Iteration berechnen, falls die Matrix \underline{A} Diagonal dominant ist.

- In Matrixform lautet die Jacobi-Iteration

$$\underline{x}^{(\text{neu})} = \underline{D}^{-1} (\underline{b} - (\underline{L} + \underline{U}) \underline{x}^{(\text{alt})}).$$

2.3. Gauss-Seidel-Iteration

Die Gauss-Seidel-Iteration ist eine naheliegende Verbesserung der Jacobi-Iteration. Die bereits neu berechneten Näherungswerte werden sofort wieder bei der Berechnung verwendet.

Gauss-Seidel-Iteration (Einzelschrittverfahren)

Genau wie beim Jacobi-Verfahren löst man zur Berechnung der Lösung eines linearen Gleichungssystems die i -te Gleichung nach der i -ten Variable auf. Im Gegensatz zum Jacobi-Verfahren verwendet man bei der Berechnung der neuen Näherungswerte alle bereits neu berechneten Näherungswerte.

Beispiel 1.8 (Gauss–Seidel–Iteration)

Das lineare Gleichungssystem

$$\begin{aligned} 3x_1 + 6x_2 &= 6 \\ x_1 + 7x_2 + 5x_3 &= 17 \\ 2x_1 + 4x_2 - 8x_3 &= -12 \end{aligned}$$

soll mit der Gauss–Seidel gelöst werden. Durch Auflösen der ersten Gleichung nach x_1 , der zweiten Gleichung nach x_2 und der dritten Gleichung nach x_3 erhält man die Iterationsvorschrift

$$\begin{aligned} x_1^{(\text{neu})} &= \frac{1}{3} \left(6 - 6x_2^{(\text{alt})} \right), \\ x_2^{(\text{neu})} &= \frac{1}{7} \left(17 - x_1^{(\text{neu})} - 5x_3^{(\text{alt})} \right), \\ x_3^{(\text{neu})} &= -\frac{1}{8} \left(-12 - 2x_1^{(\text{neu})} - 4x_2^{(\text{neu})} \right). \end{aligned}$$

Im Gegensatz zum Jacobi–Verfahren verwendet die Gauss–Seidel–Iteration bei der Berechnung von x_2 bereits den neuen Wert von x_1 und bei der Berechnung von x_3 bereits die neuen Wert von x_1 und x_2 . Wenn wir mit den Werten

$$x_1^{(0)} = 0, \quad x_2^{(0)} = 0, \quad x_3^{(0)} = 0,$$

starten, erhalten wir im ersten Schritt

$$x_1^{(1)} = 2, \quad x_2^{(1)} = \frac{15}{7}, \quad x_3^{(1)} = \frac{43}{14}.$$

Der zweite Schritt liefert

$$x_1^{(2)} = -\frac{15}{7}, \quad x_2^{(2)} = \frac{55}{98}, \quad x_3^{(2)} = \frac{237}{196}.$$

Nach ungefähr 65 Iterationsschritten erhalten wir die Lösung

$$x_1^{(65)} \approx 0, \quad x_2^{(65)} \approx 1, \quad x_3^{(65)} \approx 2,$$

mit zwölfstelliger Genauigkeit. Im Vergleich zum Jacobi-Verfahren benötigt das Gauss-Seidel-Verfahren also weniger Iterationsschritte.

Im Gegensatz zur Jacobi-Iteration arbeitet die Gauss-Seidel-Iteration ohne temporäre Werte. Eine Unterscheidung in $x^{(\text{alt})}$ und $x^{(\text{neu})}$ ist deshalb bei der Implementierung nicht notwendig. Deshalb lässt sich die Gauss-Seidel-Iteration sehr einfach implementieren.

Beispiel 1.9 (Gauss-Seidel-Iteration Implementierung)

```
void main()
{
    double x[] = {0.0, 0.0, 0.0};
    for (int i=0; i < 100; ++i)
    {
        x[0] = ( 6.0-6.0*x[1]) /3.0;
        x[1] = ( 17.0-      x[0]-5.0*x[2]) /7.0;
        x[2] = (-12.0-2.0*x[0]-4.0*x[1]) /(-8.0);
        print(i, x);
    }
}
```

Die Darstellung der Gauss-Seidel-Iteration in Matrixform unterscheidet sich von der Jacobi-Iteration nur dadurch, dass die Elemente in der unteren Dreiecksmatrix \underline{L} mit den neuen Werten multipliziert werden

$$\underline{x}^{(\text{neu})} = \underline{D}^{-1} \left(\underline{b} - \underline{L}\underline{x}^{(\text{neu})} + \underline{U}\underline{x}^{(\text{alt})} \right).$$

Genau wie bei der Jacobi-Iteration ist bei einer Diagonal dominanten Matrix \underline{A} Konvergenz sichergestellt.

Eigenschaften der Gauss–Seidel–Iteration

- Die Lösung des linearen Gleichungssystems

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

kann man mit der Gauss–Seidel–Iteration berechnen, falls die Matrix \underline{A} Diagonal dominant ist.

- In Matrixform lautet die Gauss–Seidel–Iteration

$$\underline{x}^{(\text{neu})} = \underline{D}^{-1} \left(\underline{b} - \underline{L} \underline{x}^{(\text{neu})} + \underline{U} \underline{x}^{(\text{alt})} \right).$$

- Die Gauss–Seidel–Iteration arbeitet „in place“ und lässt sich deshalb besonders einfach und effizient implementieren.

3. Lineare Algebra Software

Effiziente Algorithmen zur Lösung von linearen Gleichungssystemen sind seit Jahrzehnten Gegenstand von Forschungsprojekten. Es gibt eine ganze Reihe von frei verfügbaren Softwarepaketen, in denen die grundlegenden Algorithmen effizient realisiert sind. Weitere Informationen findet man unter den Begriffen BLAS (Basic Linear Algebra Subprograms), LAPACK (Linear Algebra Package) bzw. LPACK++.

4. Übungen

Übung I.1 (Jacobi- und Gauss-Seidel-Iteration)

Das lineare Gleichungssystem $\underline{A}x = \underline{b}$ mit

$$\underline{A} = \begin{bmatrix} 3 & 2 & 1 \\ 1 & 3 & 2 \\ 2 & 1 & 3 \end{bmatrix} \quad \text{und} \quad \underline{b} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

soll iterativ gelöst werden.

- Führen Sie einen Schritt der Jacobi-Iteration und dem Startvektor $[0; 0; 0]$ durch.
- Führen Sie einen Schritt der Gauss-Seidel-Iteration und dem Startvektor $[0; 0; 0]$ durch.
- Ist auf Grund der Theorie sichergestellt, dass die Iterationen konvergieren? Ermitteln Sie experimentell, ob die Jacobi- und die Gauss-Seidel-Iteration konvergieren.

Übung I.2 (Jacobi- und Gauss-Seidel-Iteration)

Das lineare Gleichungssystem $\underline{A}x = \underline{b}$ mit

$$\underline{A} = \begin{bmatrix} 0 & -1 & 2 \\ 2 & 0 & -1 \\ -1 & 2 & 0 \end{bmatrix} \quad \text{und} \quad \underline{b} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

soll iterativ gelöst werden.

- Warum kann man die Jacobi-Iteration und die Gauss-Seidel-Iteration nicht auf die Matrix \underline{A} anwenden?
- Vertauschen Sie die Zeilen des linearen Gleichungssystems so, dass die Jacobi-Iteration und die Gauss-Seidel-Iteration konvergieren.

- c) Führen Sie mit der modifizierten Matrix \underline{A} und dem modifizierten Vektor \underline{b} einen Schritt der Jacobi-Iteration zum Startvektor $[0; 0; 0]$ durch.
- d) Führen Sie mit der modifizierten Matrix \underline{A} und dem modifizierten Vektor \underline{b} einen Schritt der Gauss-Seidel-Iteration zum Startvektor $[0; 0; 0]$ durch.
- e) Verwenden Sie geeignete Programme für die Jacobi-Iteration und die Gauss-Seidel-Iteration und vergleichen Sie die Anzahl der Iterationsschritte.

Übung I.3 (Jacobi-Iteration)

Das lineare Gleichungssystem $\underline{A}x = \underline{b}$ mit

$$\underline{A} = \begin{bmatrix} 0 & -1 & 4 & -1 \\ -1 & 0 & -1 & 4 \\ 4 & -1 & 0 & -1 \\ -1 & 4 & -1 & 0 \end{bmatrix} \quad \text{und} \quad \underline{b} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

soll iterativ gelöst werden.

- a) Vertauschen Sie die Zeilen des linearen Gleichungssystems so, dass man die Jacobi-Iteration anwenden kann und das Verfahren konvergiert.
- b) Führen Sie mit der modifizierten Matrix \underline{A} und dem modifizierten Vektor \underline{b} einen Schritt der Jacobi-Iteration und dem Startvektor $[0; 0; 0; 0]$ durch.

Übung I.4 (Iterative Löser)

Das lineare Gleichungssystem $\underline{A}x = \underline{b}$ soll gelöst werden. Dabei ist \underline{A} eine $n \times n$ Matrix und \underline{b} ein Vektor mit n Elementen. Die Struktur von \underline{A} und

von \underline{b} ist jeweils gleich, z.B. ergibt sich für $n = 9$ die folgende Matrix und der folgende Vektor:

$$\underline{A} = \begin{bmatrix} 0 & -1 & 4 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 4 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 4 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 4 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 4 & -1 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 4 \\ 4 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ -1 & 4 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \underline{b} = \begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \\ 1 \\ -1 \\ 1 \\ -1 \\ 1 \end{bmatrix}$$

- Warum kann man die Jacobi-Iteration und die Gauss-Seidel-Iteration nicht auf die Matrix \underline{A} anwenden?
- Vertauschen Sie die Zeilen der Matrix \underline{A} so, dass man die Jacobi-Iteration und die Gauss-Seidel-Iteration anwenden kann.
- Begründen Sie, warum die Jacobi-Iteration und die Gauss-Seidel-Iteration nach geeignetem Vertauschen der Zeilen konvergiert.
- Führen Sie einen Schritt der Jacobi-Iteration zum Startwert $\underline{x}_0 = [0; 0; 0; 0; 0; 0; 0; 0; 0]$ durch.
- Verwenden Sie ein Programm für die Jacobi-Iteration und die Gauss-Seidel-Iteration und vergleichen Sie die Anzahl der Iterationsschritte, die zur Lösung des linearen Gleichungssystems für $n = 10^4$ mit einer Genauigkeit von 8 Nachkommastellen notwendig sind.
- Wieviele elementare Operationen würde der Gauss-Algorithmus zur Lösung des linearen Gleichungssystems für $n = 10^4$ benötigen.

II Nicht lineare Gleichungen und Gleichungssysteme

Zwischen linearen und nicht linearen Problemstellungen bestehen in der Mathematik gravierende Unterschiede.

Lineare und nicht lineare Gleichungssysteme

Für lineare Gleichungssysteme existieren Methoden um alle wesentlichen Fragen systematisch zu beantworten:

- (A) Gibt es überhaupt eine Lösung (Existenz)?
- (B) Wieviele Lösungen gibt es (Eindeutigkeit)?
- (C) Wieviele Rechenoperationen sind notwendig (Komplexität)?
- (D) Welchen Einfluss haben Rundungsfehler (Stabilität)?

Bei nicht linearen Gleichungssystemen sind diese Fragestellungen oft mit ungelösten mathematischen Problemen verbunden.

1. Intervallhalbierung

Die Problemstellung eine nicht lineare Gleichung zu lösen kann man stets als Nullstellenproblem formulieren.

Beispiel II.1 (Nullstellenproblem)

Wenn wir alle Lösungen der Gleichung

$$\cos x = x$$

suchen, dann können wir genau so gut alle Nullstellen der Funktion

$$f(x) = \cos x - x = 0$$

berechnen.

Eine einfache Methode zur Bestimmung der Nullstellen einer Funktion $f(x)$ beruht auf dem Zwischenwertsatz stetiger Funktionen.

Zwischenwertsatz

Eine Funktion, die auf dem Intervall $[a; b]$ stetig ist, nimmt in diesem Intervall alle Funktionswerte zwischen $f(a)$ und $f(b)$ mindestens einmal an. Insbesondere hat somit eine Funktion mit unterschiedlichem Vorzeichen an den Stellen a und b mindestens eine Nullstelle im Intervall $[a; b]$.

Intervallhalbierung

Bei einer stetigen Funktion $f(x)$ mit $f(a) \cdot f(b) < 0$, kann man eine Nullstelle durch fortgesetztes Unterteilen des Intervalls $[a; b]$ näherungsweise bestimmen. Falls der Funktionswert in der Intervallmitte $f(\frac{a+b}{2})$ das selbe Vorzeichen wie $f(a)$ hat wird das Intervall $[\frac{a+b}{2}; b]$ weiter untersucht, ansonsten setzt man die Suche auf $[a; \frac{a+b}{2}]$ fort.

Intervallhalbierung

```
function x = izero(f,a,b)
fa = feval(f,a);
fb = feval(f,b);
if (fa*fb >= 0)
    error(['f(a)*f(b) >= 0 !']);
else
    while(1)
        m = (a+b)/2, pause
        if (abs(b-a) < 10^(-12))
            x = m;
            break;
        else
            fm = feval(f,m);
            if (fa*fm < 0)
                b = m;
                fb = fm;
            else
                a = m;
                fa = fm;
            end
        end
    end
end
```

Intervallhalbierung ist ein einfaches und robustes Verfahren, das allerdings sehr langsam konvergiert. Außerdem benötigt man zwei Startwerte, bei denen die Funktionswerte unterschiedliches Vorzeichen haben. Leider ist die Idee nicht direkt auf nicht lineare Gleichungssysteme übertragbar.

2. Fixpunktiteration

Bei der sogenannten Fixpunktiteration geht man so ähnlich wie bei der Jacobi- oder Gauss-Seidel-Iteration vor. Die Gleichung

$$f(x) = 0$$

wir auf irgend eine Art nach der Variablen x aufgelöst. Dadurch ergibt sich die Fixpunktgleichung

$$x = g(x).$$

Wenn wir nun einen geeigneten Startwert x_0 haben, dann ist zu hoffen, dass die Iteration

$$x^{(k+1)} = g(x^{(k)})$$

gegen eine gesuchte Lösung x^* konvergiert.

Beispiel II.2 (Fixpunktiteration)

Aus den Schaubildern der beiden Funktionen $f_1(x) = \cos x$ und $f_2(x) = x$ erkennen wir, dass in der Nähe von $x^{(0)} = 0.7$ ein Schnittpunkt vorliegt. Die Fixpunktiteration

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

liefert mit dem Startwert $x_0 = 0.7$ folgende Werte:

k	$x^{(k)}$	k	$x^{(k)}$
1	0.76484218728449	11	0.73958442869535
2	0.72149163959753	12	0.73874870966209
3	0.75082132883945	13	0.73931171033801
4	0.73112877257336	14	0.73893248916970
5	0.74442118362716	15	0.73918794746955
6	0.73548020040599	16	0.73901587239041
7	0.74150865166004	17	0.73913178636711
8	0.73745045315018	18	0.73905370628650
9	0.74018528539676	19	0.73910630240736
10	0.73834361035100	20	0.73907087322704

Tatsächlich konvergieren die Zahlenwerte in Beispiel 11.2 gegen einen Grenzwert, allerdings ist die Konvergenzgeschwindigkeit dabei langsam. Außerdem ist zu beobachten, dass der jeweils neu berechnete Wert zwischen den beiden letzten Näherungswerten liegt. Man spricht hier von alternierender Konvergenz.

Definition 11.1 (Alternierende Konvergenz) Wenn bei einer konvergenten Zahlenfolge der nachfolgende Wert $x^{(k+1)}$ zwischen den beiden Vorgängerwerten $x^{(k-1)}$ und $x^{(k)}$ liegt

$$\begin{array}{cccccc}
 x^{(0)} & \leq & x^{(2)} & \leq & x^{(1)} & & x^{(1)} & \leq & x^{(2)} & \leq & x^{(0)} \\
 x^{(2)} & \leq & x^{(3)} & \leq & x^{(1)} & & x^{(1)} & \leq & x^{(3)} & \leq & x^{(2)} \\
 x^{(2)} & \leq & x^{(4)} & \leq & x^{(3)} & \text{oder} & x^{(3)} & \leq & x^{(4)} & \leq & x^{(2)} \\
 x^{(4)} & \leq & x^{(5)} & \leq & x^{(3)} & & x^{(3)} & \leq & x^{(5)} & \leq & x^{(4)} \\
 \vdots & & \vdots & & \vdots & & \vdots & & \vdots & & \vdots
 \end{array}$$

dann bezeichnet man die Konvergenz als alternierend.

Im Gegensatz zur alternierenden Konvergenz ist es durchaus auch möglich, dass die Werte monoton gegen einen Grenzwert streben.

Definition 11.2 (Monotone Konvergenz) Bei einer konvergenten Zahlenfolge spricht man von monoton fallender Konvergenz, wenn die nachfolgenden Werte stets kleiner sind als die Vorgängerwerte

$$x^{(k+1)} \leq x^{(k)}$$

und von monoton wachsender Konvergenz, wenn die nachfolgenden Werte stets größer sind als die Vorgängerwerte

$$x^{(k+1)} \geq x^{(k)}.$$

Bei der Fixpunktiteration können wir nur dann mit Konvergenz rechnen, wenn der neue Wert $x^{(k+1)}$ näher an der gesuchten Lösung x^* als der alte Wert $x^{(k)}$, d.h.

$$|x^{(k+1)} - x^*| < |x^{(k)} - x^*|.$$

Wenn wir dabei $x^{(k+1)} = g(x^{(k)})$ und $g(x^*) = x^*$ berücksichtigen, erhalten wir

$$|g(x^{(k)}) - g(x^*)| < |x^{(k)} - x^*|,$$

bzw.

$$\underbrace{\left| \frac{g(x^{(k)}) - g(x^*)}{x^{(k)} - x^*} \right|}_{\rightarrow g'(x^*)} < 1.$$

Das Konvergenzverhalten der Fixpunktiteration

$$x^{(k+1)} = g(x^{(k)})$$

ist also Abhängig von den Werten der Ableitung $g'(x)$ im betrachteten Bereich.

Im wesentlichen sind dabei vier Fälle zu unterscheiden.

Beispiel II.3 (Konvergenz der Fixpunktiteration)

Zur Konvergenzuntersuchung der Fixpunktiteration

$$x^{(k+1)} = \cos x^{(k)}, \quad x^{(0)} = 0.7,$$

betrachten wir den maximalen Wert der Ableitung über dem bei der Iteration vorkommenden Zahlenbereich

$$L = \max_{x \in [0.7; 0.765]} |-\sin x| \leq \sin(0.765) < 0.7.$$

Da die Ableitung stets negativ und $L < 1$ ist, liegt alternierende Konvergenz vor.

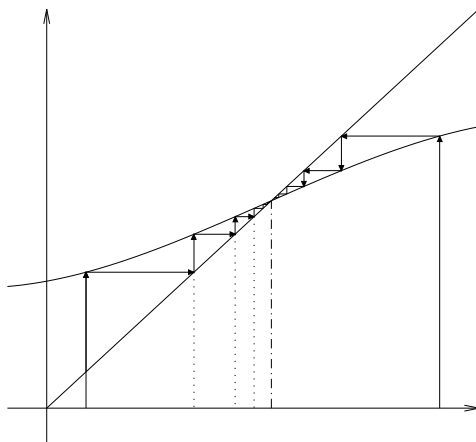


Abbildung II.1.: Fixpunktiteration für eine schwach steigende Funktion ($0 < g'(x) < 1$).

Konvergenz der Fixpunktiteration

Das Konvergenzverhalten der Fixpunktiteration

$$x^{(k+1)} = g(x^{(k)})$$

hängt von der Steigung der Funktion $g(x)$ in der Nähe des Fixpunktes ab. Wenn der Betrag der Ableitung über dem bei der Iteration vorkommenden Zahlenbereich

$$L = \max |g'(x)| .$$

kleiner als 1 ist, dann ist die Konvergenz der Fixpunktiteration sichergestellt. L bezeichnet man als Lipschitz-Konstante.

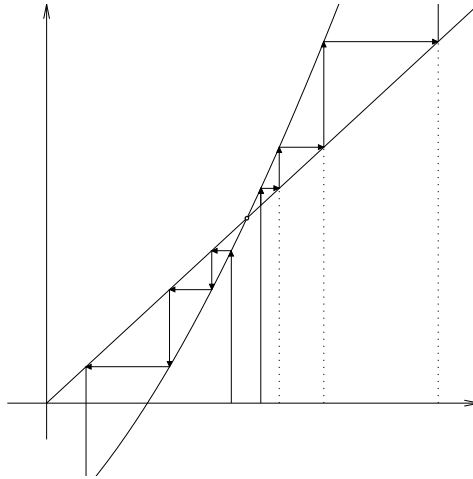


Abbildung II.2.: Fixpunktiteration für eine stark steigende Funktion ($1 < g'(x)$).

„A priori“-Fehlerabschätzung

Bei der Fixpunktiteration

$$x^{(k+1)} = g(x^{(k)})$$

lässt sich der maximale Fehler von $x^{(k)}$ mit Hilfe der Differenz der beiden ersten Iterationswerte und der Lipschitz-Konstante L abschätzen:

$$|x^* - x^{(k)}| < \frac{L^k}{1 - L} |x^{(1)} - x^{(0)}|.$$

Beispiel II.4 („A priori“-Fehlerabschätzung)

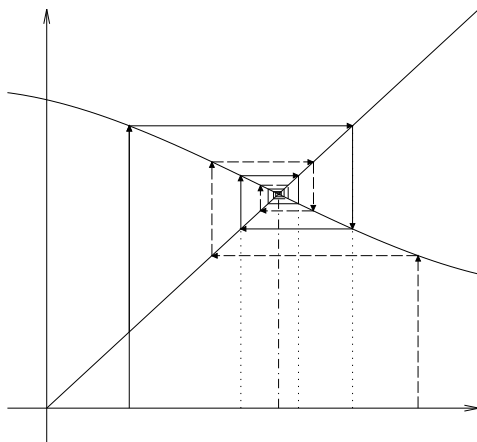


Abbildung II.3.: Fixpunktiteration für eine schwach fallende Funktion ($-1 < g'(x) < 0$).

Bei der der Fixpunktiteration

$$x^{(k+1)} = \cos x^{(k)}, \quad x^{(0)} = 0.7,$$

haben wir bereits in Beispiel II.3 die Lipschitzkonstante $L = 0.7$ ermittelt. Daher ergibt sich aus den beiden ersten Iterationswerten

$$|x^* - x^{(k)}| < \frac{L^k}{1 - L} |x^{(1)} - x^{(0)}| < \frac{0.7^k}{0.3} |0.76484218728449 - 0.7|.$$

Wenn wir einen maximalen Fehler von 10^{-4} fordern, erhalten wir daraus

$$k \geq \dots$$

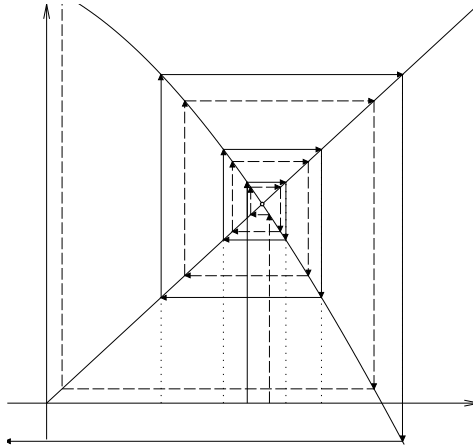


Abbildung II.4.: Fixpunktiteration für eine stark fallende Funktion ($g'(x) < -1$).

„A posteriori“–Fehlerabschätzung

Bei der Fixpunktiteration

$$x^{(k+1)} = g(x^{(k)})$$

lässt sich der maximale Fehler von $x^{(k)}$ mit Hilfe der Differenz der beiden letzten Iterationswerte und der Lipschitz–Konstante L abschätzen:

$$|x^* - x^{(k)}| < \frac{L}{1-L} |x^{(k)} - x^{(k-1)}|.$$

Beispiel II.5 („A posteriori“–Fehlerabschätzung)

Bei der der Fixpunktiteration

$$x^{(k+1)} = \cos x^{(k)}, \quad x^{(0)} = 0.7,$$

haben wir bereits in Beispiel II.3 die Lipschitzkonstante $L = 0.7$ ermittelt. Daher ergibt sich nach zwanzig Iterationsschritten aus den beiden letzten Iterationswerten

$$|x^* - x^{(k)}| < \frac{L}{1-L} |x^{(20)} - x^{(19)}| < \frac{0.7}{0.3} |0.739071 - 0.739106| < 10^{-4}.$$

Mit der „A posteriori“-Fehlerabschätzung erkennen wir also, dass bereits der zwanzigste Iterationsschritt für eine Genauigkeit von 10^{-4} genügt.

Bei praktischen Problemen ist die Bestimmung einer Lipschitzkonstanten oft sehr aufwändig, deshalb begnügt man sich oft mit dem Abbruchkriterium

$$|x^{(k)} - x^{(k-1)}| < \text{tol},$$

dabei ist tol eine der Problemstellung angepasste vorgegebene Toleranz.

3. Newton–Verfahren

Die Fixpunktiteration lässt noch einige Fragen offen. Konvergenz ist nur gesichert, falls die Lipschitzkonstante kleiner als 1 ist. Die Konvergenzgeschwindigkeit der Fixpunktiteration ist oftmals sehr langsam.

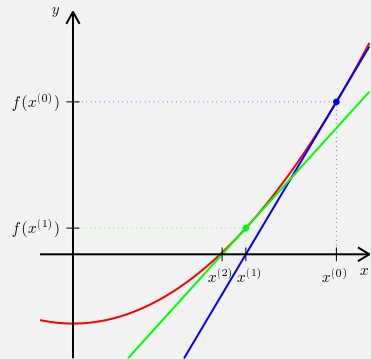
Das bekannteste Näherungsverfahren zur Bestimmung von Nullstellen von Funktionen geht auf den englischen Mathematiker Sir Isaac Newton (1643–1727) zurück. Die Grundidee beruht darauf, die Funktion an geeigneten Punkten zu linearisieren und dann die Nullstelle der Linearisierung zu bestimmen. Die Linearisierung einer Funktion in einer Veränderlichen ist die Tangente.

Newton'sches Näherungsverfahren (Grafisch)

Zur näherungsweise Berechnung einer Nullstelle einer Funktion wählt man zunächst einen geeigneten Startwert.

Dann wiederholt man die folgenden Schritte:

- (1) Bestimme die Tangente der Funktion im Startpunkt.
- (2) Der neue Startpunkt ist der Schnittpunkt der Tangente mit der x -Achse.



Aus dem grafischen Verfahren lassen sich geeignete Formeln für die Iterationsvorschrift ableiten. Die Gleichung der Tangente im Startpunkt $x^{(0)}$ ist gegeben durch

$$y = f(x^{(0)}) + f'(x^{(0)})(x - x^{(0)}).$$

Die Schnittbedingung mit der x -Achse $y = 0$ liefert den gesuchten x -Wert

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

Diesen x -Wert wählen wir nun als neuen Startwert $x^{(1)}$ für den nächsten Schritt. Wenn alles gut läuft, dann nähern wir uns so Schritt für Schritt einer Nullstelle.

Newton'sches Näherungsverfahren

Zur Berechnung einer Lösung der Gleichung

$$f(x) = 0$$

bestimmt man zunächst einen geeigneten Näherungswert $x^{(0)}$. Dann berechnet man Schritt für Schritt Näherungswerte $x^{(1)}, x^{(2)}, \dots$ aus der Iterationsvorschrift

$$x^{(n+1)} = x^{(n)} - \frac{f(x^{(n)})}{f'(x^{(n)})}, \quad n = 0, 1, 2, \dots,$$

solange bis die gewünschte Genauigkeit erreicht ist.

Beispiel II.6 (Newton'sches Näherungsverfahren)

Wir wollen die Nullstellen von $x - \cos x = 0$ bestimmen. Die Schaubilder der beiden Funktionen $f_1(x) = x$ und $f_2(x) = \cos x$ schneiden sich in einem einzigen Punkt. Als Startwert für das Newton'sche Iterationsverfahren wählen wir $x_0 = 0.7$. Die weiteren Näherungswerte ergeben sich aus der Formel

$$x^{(n+1)} = x^{(n)} - \frac{f(x^{(n)})}{f'(x^{(n)})} = x^{(n)} - \frac{x^{(n)} - \cos x^{(n)}}{1 + \sin x^{(n)}}.$$

Der Tabelle kann man die einzelnen Zahlenwerte entnehmen.

k	$x^{(k)}$
0	0.70000000000000
1	0.73943649784806
2	0.73908516046511
3	0.73908513321516
4	0.73908513321516

Bereits der Näherungswert $x^{(3)}$ ist bis zur 14. Nachkommastelle exakt.

Bei der Implementierung verwendet man die Iterationsvorschrift gerne in der Form

$$\Delta x = x^{(n+1)} - x^{(n)} = -\frac{f(x^{(n)})}{f'(x^{(n)})}.$$

Dadurch kann man die Veränderung Δx von $x^{(n)}$ zu $x^{(n+1)}$ im Abbruchkriterium verwenden. Typischerweise wird die Iteration abgebrochen, falls sich im Rahmen der Maschinengenauigkeit keine Verbesserung mehr ergibt.

Newton–Verfahren

```
double newton(double x)
{
    double delta;
    do
    {
        delta = -f(x)/fp(x);
        x += delta;
    } while (abs(delta) >= DBL_EPSILON);
    return x;
}
```

4. Newton–Verfahren für Systeme

Die Problemstellung nicht lineare Gleichungssysteme zu lösen ist äquivalent zur Bestimmung der gemeinsamen Nullstellen von Funktionen.

Beispiel II.7 (Nicht lineares Gleichungssystem)

Wir suchen alle x_1 - und x_2 -Werte, so dass die beiden Gleichungen

$$\begin{aligned}x_1^2 + 2x_2^2 &= 8, \\x_1^3 &= 4x_2,\end{aligned}$$

erfüllt sind. Durch Umformung erhalten wir

$$\begin{aligned}f_1(x_1, x_2) &= x_1^2 + 2x_2^2 - 8 = 0, \\f_2(x_1, x_2) &= x_1^3 - 4x_2 = 0.\end{aligned}$$

Unsere gesuchten x_1 - und x_2 -Werte sind also gemeinsame Nullstellen der beiden Funktionen $f_1(x_1, x_2)$ und $f_2(x_1, x_2)$.

Fassen wir die Variablen x_1, x_2, \dots, x_n zu einem Vektor \underline{x} und die Funktionen $f_1(x_1, x_2, \dots, x_n), f_2(x_1, x_2, \dots, x_n), \dots, f_n(x_1, x_2, \dots, x_n)$ zu einem Vektor $\underline{f}(\underline{x})$ zusammen

$$\underline{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}, \quad \underline{f}(\underline{x}) = \begin{pmatrix} f_1(x_1, x_2, \dots, x_n) \\ f_2(x_1, x_2, \dots, x_n) \\ \vdots \\ f_n(x_1, x_2, \dots, x_n) \end{pmatrix},$$

so können wir das nicht lineare Gleichungssystem in der Form

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

schreiben. Genau wie bei linearen Gleichungssystemen betrachten wir nur den Fall, dass die Anzahl der Gleichungen der Anzahl der Unbekannten entspricht. Allerdings kann man bei nicht linearen Gleichungssystemen nicht erwarten, dass es immer eine eindeutige Lösung gibt. Die Frage nach der Existenz von Lösungen und nach der Anzahl der Lösungen führt auf mathematische Problemstellungen, die bis heute teilweise nicht gelöst sind.

Bei naturwissenschaftlichen und technischen Anwendungen sind die Funktionen in der Regel so komplex, dass eine exakte Berechnung der Nullstellen durch Formeln nur in Ausnahmefällen möglich ist. Zur Berechnung der Nullstellen kann man jedoch iterative Algorithmen verwenden.

Das Newton–Verfahren lässt sich problemlos von einer Funktion in einer Veränderlichen auf Gleichungssysteme mit mehreren Funktionen in mehreren Veränderlichen übertragen. Dazu benötigen wir die Linearisierung einer Funktion in mehreren Veränderlichen.

Definition II.3 (Jacobi–Matrix) Die Jacobi–Matrix der Funktion $\underline{f}(\underline{x})$ enthält die partiellen Ableitungen aller Funktionen nach allen Variablen

$$\underline{J} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \dots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \dots & \frac{\partial f_n}{\partial x_n} \end{pmatrix}.$$

Beispiel II.8 (Jacobi–Matrix)

Die Jacobi–Matrix der Funktion

$$\underline{f}(\underline{x}) = \begin{pmatrix} x_1^2 + 2x_2^2 - 8 \\ x_1^3 - 4x_2 \end{pmatrix}$$

ist

$$\underline{J} = \begin{pmatrix} 2x_1 & 4x_2 \\ 3x_1^2 & -4 \end{pmatrix}.$$

Mit Hilfe der Jacobi–Matrix lässt sich die Iterationsvorschrift

$$\underline{x}^{(n+1)} = \underline{x}^{(n)} - \frac{\underline{f}(\underline{x}^{(n)})}{\underline{f}'(\underline{x}^{(n)})},$$

verallgemeinern

$$\underline{x}^{(n+1)} = \underline{x}^{(n)} - \underline{J}^{-1} \cdot \underline{f}(\underline{x}^{(n)}).$$

Das Teilen durch die Ableitung entspricht bei Funktionen in mehreren Veränderlichen der Multiplikation mit der inversen Jacobi–Matrix \underline{J}^{-1} .

Newton–Verfahren für Systeme

Zur Berechnung einer Lösung der Gleichung

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

bestimmt man zunächst einen geeigneten Näherungswert $\underline{x}^{(0)}$. Dann berechnet man Schritt für Schritt Näherungswerte $\underline{x}^{(1)}, \underline{x}^{(2)}, \dots$ aus der Iterationsvorschrift

$$\underline{x}^{(n+1)} = \underline{x}^{(n)} - \underline{J}^{-1} \cdot \underline{f}(\underline{x}^{(n)}).$$

solange bis die gewünschte Genauigkeit erreicht ist. Dabei ist \underline{J}^{-1} die Inverse der Jacobi–Matrix an der Stelle $\underline{x}^{(n)}$.

Beispiel II.9 (Newton–Verfahren für Systeme)

Wir bestimmen eine Lösung des nicht linearen Gleichungssystems

$$\begin{aligned} f_1(x_1, x_2) &= x_1^2 + 2x_2^2 - 8 = 0, \\ f_2(x_1, x_2) &= x_1^3 - 4x_2 = 0, \end{aligned}$$

mit dem Newton–Verfahren. Als Startwert wählen wir

$$x_1 = 1, \quad x_2 = 1.$$

Der Funktionswert am Startwert ist

$$\underline{f}(1, 1) = \begin{pmatrix} -5 \\ -3 \end{pmatrix}.$$

Die Jacobi-Matrix der Funktion

$$\underline{J} = \begin{pmatrix} 2x_1 & 4x_2 \\ 3x_1^2 & -4 \end{pmatrix}$$

haben wir bereits in Beispiel II.8 bestimmt. Wie man leicht nachprüfen kann, gilt für die Inverse einer 2×2 Matrix die Formel

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}^{-1} = \frac{1}{ad - bc} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}.$$

Für unsere Jacobi-Matrix erhalten wir damit

$$\underline{J}^{-1} = \frac{1}{-8x_1 - 12x_1^2x_2} \begin{pmatrix} -4 & -4x_2 \\ -3x_1^2 & 2x_1 \end{pmatrix}$$

und an der Stelle $x_1 = 1$ und $x_2 = 1$

$$\underline{J}^{-1} = \frac{-1}{20} \begin{pmatrix} -4 & -4 \\ -3 & 2 \end{pmatrix}.$$

Dadurch erhalten wir den ersten Näherungswert

$$\underline{x}^{(1)} = \begin{pmatrix} 1 \\ 1 \end{pmatrix} + \frac{1}{20} \begin{pmatrix} -4 & -4 \\ -3 & 2 \end{pmatrix} \cdot \begin{pmatrix} -5 \\ -3 \end{pmatrix} = \begin{pmatrix} 2.6 \\ 1.45 \end{pmatrix}.$$

Unter Zuhilfenahme geeigneter Datenstrukturen für Vektoren und Matrizen unterscheidet sich die Implementierung des Newton-Verfahrens für Systeme kaum von der Implementierung in 3.

Newton–Verfahren für Systeme

```
Vector newton(Vector x)
{
    Vector delta;
    do
    {
        delta = -Jinv(x)*f(x);
        x += delta;
    } while (maxabs(delta) >= DBL_EPSILON);
    return x;
}
```

Bisher haben wir uns noch keine Gedanken gemacht unter welchen Umständen das Newton–Verfahren überhaupt gegen eine Lösung des Problems konvergiert. Konvergenzuntersuchungen beim Newton–Verfahren sind mit vielen interessanten mathematischen Problemen verbunden, die bis heute teilweise noch nicht gelöst sind. Die Visualisierung der Problemstellungen mit Hilfe leistungsfähiger Computer hat zu verblüffenden Ergebnissen geführt.

Beispiel II.10 (Newton–Fraktal)

Wir bestimmen die Lösungen des nicht linearen Gleichungssystems

$$\begin{aligned}f_1(x_1, x_2) &= x_1^3 - 3x_1x_2^2 + 1 = 0, \\f_2(x_1, x_2) &= 3x_1^2x_2 - x_2^3 = 0,\end{aligned}$$

mit dem Newton–Verfahren. Die Lösungen des Systems kann man recht einfach durch komplexe Rechnung bestimmen. Die Gleichung

$$z^3 = (x + jy)^3 = -1$$

hat genau drei komplexe Lösungen

$$z_1 = -1, \quad z_2 = \frac{1}{2} + \frac{\sqrt{3}}{2}j, \quad z_3 = \frac{1}{2} - \frac{\sqrt{3}}{2}j.$$

Andererseits gilt

$$(x + jy)^3 = x^3 + 3x^2yj - 3xy^2 - y^3j,$$

somit beschreibt unser Gleichungssystem den Realteil und den Imaginärteil der komplexen Gleichung $z^3 = -1$. Da wir alle drei Lösungen kennen, interessiert uns die Frage, welcher Startwert zu welcher Lösung konvergiert? Zur Visualisierung des Problems ordnen wir jeder Lösung eine bestimmte Farbe zu

$$z_1 \rightarrow \text{blau}, \quad z_2 \rightarrow \text{grün}, \quad z_3 \rightarrow \text{rot}.$$

In Abbildung 11.5 wurden die Startpunkte mit der Farbe des entsprechenden Lösungspunktes markiert.

Abbildung 11.5 bezeichnet man als Julia-Menge oder Newton-Fraktal. Solche selbstähnlichen Bilder sind typisch für die Konvergenzbereiche des Newton-Verfahrens.

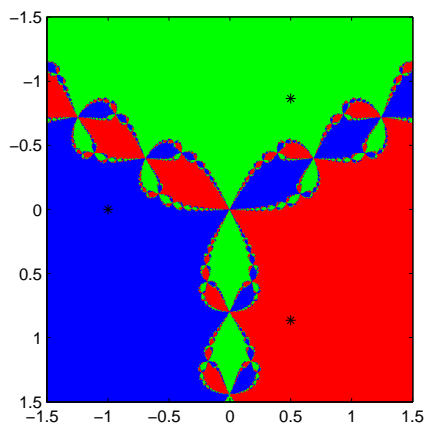


Abbildung II.5.: Newton-Fraktal zur Gleichung $z^3 + 1 = 0$.

Eigenschaften des Newtonschen Näherungsverfahrens

- Wenn der Startwert hinreichend nahe bei einer Lösung liegt, dann konvergiert das Newtonsche Näherungsverfahren in der Regel sehr schnell, dabei verdoppelt sich die Anzahl der gültigen Ziffern pro Schritt.
- Bei mehrfachen Nullstelle konvergiert das Verfahren langsam oder gar nicht.
- Ausgehend von einem bestimmten Startwert findet das Newton–Verfahren höchstens eine Nullstelle. Zur Bestimmung mehrere Nullstellen, kann man die Iteration mit unterschiedlichen Startwerten durchführen.
- Es gibt Situationen, in denen die Newton–Iteration nicht gegen eine Nullstelle konvergiert.

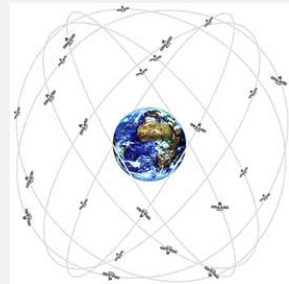
Bei praktischen Problemen ist es oft sehr schwierig oder sogar unmöglich die Ableitung einer Funktion analytisch zu bestimmen. Man kann sich damit behelfen, die Ableitung mit Hilfe von Differenzen numerisch anzunähern. Im Fall einer Funktion in einer Veränderlichen stellt die Regula Falsi ein ableitungsfreies Newton–Verfahren dar.

5. Global Positioning System (GPS)

5.1. Das GPS-Prinzip

GPS-Prinzip

- Das GPS besteht aus Bodenstationen, Satelliten und Empfänger.
- Der GPS-Empfänger ermittelt aus der Zeitdifferenz zwischen Sendezeit am Satelliten und Empfangszeit am Empfänger die Entfernung des Empfängers zum jeweiligen Satelliten.
- Aus den Entfernungen zu mindestens 3 (4) Satelliten lässt sich die genaue Position des Empfängers ermitteln.



- 24 Satelliten in 6 Ebenen.
- Umlaufzeit 12 Stunden.

GPS–Satelliten

- GPS–Satelliten sind nicht geostationäre Satelliten, deren Umlaufbahnen sich in einer Höhe von rund 20.000 Kilometer befinden.
- Ein GPS–Satellit sendet Datenpakete, die die Position des jeweiligen Satelliten zu einer bestimmten Uhrzeit enthalten.
- Die Koordinaten der Satelliten werden bezogen auf ein Koordinatensystem, das mit der Erde rotiert „Earth–Centered Earth–Fixed (ECEF)“ angegeben.
- Aufgrund der Lichtgeschwindigkeit ($c \approx 299\,792\,458$ Meter pro Sekunde) ergeben sich Signallaufzeiten zwischen den GPS–Satelliten und dem GPS–Empfänger von ungefähr 0.07 Sekunden.

5.2. Positionsbestimmung

3 Satelliten

Aus den Koordinaten von 3 Satelliten

$$S_1(x_1/y_1/z_1), \quad S_2(x_2/y_2/z_2), \quad S_3(x_3/y_3/z_3)$$

und den entsprechenden Signallaufzeiten t_1, t_2, t_3 , kann man durch Lösen des nicht linearen Gleichungssystems

$$\sqrt{(x - x_1)^2 + (y - y_1)^2 + (z - z_1)^2} = c \cdot t_1,$$

$$\sqrt{(x - x_2)^2 + (y - y_2)^2 + (z - z_2)^2} = c \cdot t_2,$$

$$\sqrt{(x - x_3)^2 + (y - y_3)^2 + (z - z_3)^2} = c \cdot t_3,$$

die unbekannte Position $P(x/y/z)$ des GPS–Empfängers berechnen.

5.3. Laufzeitfehler

Laufzeitfehler

Auf Grund der Lichtgeschwindigkeit haben kleine Ungenauigkeiten in der Laufzeit große Auswirkung auf die Genauigkeit der Ortsbestimmung. Eine Abweichung von einer Mikrosekunde (10^{-6} Sekunden) erzeugt eine Ortsungenauigkeit von rund 300 Metern.

4 Satelliten

Aus den Koordinaten von 4 Satelliten

$$S_1(x_1/y_1/z_1), \quad S_2(x_2/y_2/z_2), \quad S_3(x_3/y_3/z_3), \quad S_4(x_4/y_4/z_4)$$

und den entsprechenden Signallaufzeiten t_1, t_2, t_3, t_4 , kann man durch Lösen des nicht linearen Gleichungssystems

$$\sqrt{(x - x_1)^2 + (y - y_1)^2 + (z - z_1)^2} = c(t_1 - \Delta t)$$

$$\sqrt{(x - x_2)^2 + (y - y_2)^2 + (z - z_2)^2} = c(t_2 - \Delta t)$$

$$\sqrt{(x - x_3)^2 + (y - y_3)^2 + (z - z_3)^2} = c(t_3 - \Delta t)$$

$$\sqrt{(x - x_4)^2 + (y - y_4)^2 + (z - z_4)^2} = c(t_4 - \Delta t)$$

die Position $P(x/y/z)$ und die Laufzeitkorrektur Δt berechnen.

5.4. Newton–Verfahren für GPS

Gleichungssystem

$$(x - x_1)^2 + (y - y_1)^2 + (z - z_1)^2 - c^2 (t_1 - \Delta t)^2 = 0$$

$$(x - x_2)^2 + (y - y_2)^2 + (z - z_2)^2 - c^2 (t_2 - \Delta t)^2 = 0$$

$$(x - x_3)^2 + (y - y_3)^2 + (z - z_3)^2 - c^2 (t_3 - \Delta t)^2 = 0$$

$$(x - x_4)^2 + (y - y_4)^2 + (z - z_4)^2 - c^2 (t_4 - \Delta t)^2 = 0$$

Jacobi-Matrix

$$\underline{J} = 2 \begin{pmatrix} x - x_1 & y - y_1 & z - z_1 & c^2 (t_1 - \Delta t) \\ x - x_2 & y - y_2 & z - z_2 & c^2 (t_2 - \Delta t) \\ x - x_3 & y - y_3 & z - z_3 & c^2 (t_3 - \Delta t) \\ x - x_4 & y - y_4 & z - z_4 & c^2 (t_4 - \Delta t) \end{pmatrix}$$

Startwert für Newton-Iteration

Beim GPS kann man einen Startwert für die gesuchte Position \underline{x} durch lösen eines linearen Gleichungssystem

$$\underline{A} \cdot \underline{x} = \underline{b}$$

berechnen. Dabei ist

$$\underline{A} = 2 \begin{pmatrix} x_2 - x_1 & y_2 - y_1 & z_2 - z_1 \\ x_3 - x_2 & y_3 - y_2 & z_3 - z_2 \\ x_4 - x_3 & y_4 - y_3 & z_4 - z_3 \end{pmatrix}$$

und

$$\underline{b} = \begin{pmatrix} x_2^2 - x_1^2 + y_2^2 - y_1^2 + z_2^2 - z_1^2 + c^2(t_2^2 - t_1^2) \\ x_3^2 - x_2^2 + y_3^2 - y_2^2 + z_3^2 - z_2^2 + c^2(t_3^2 - t_2^2) \\ x_4^2 - x_3^2 + y_4^2 - y_3^2 + z_4^2 - z_3^2 + c^2(t_4^2 - t_3^2) \end{pmatrix}.$$

6. Übungen

Übung II.1 (Intervallhalbierungsmethode)

Bestimmen Sie eine Lösung der nichtlinearen Gleichung

$$\sin(x) = 1 - x$$

mit der Intervallhalbierungsmethode.

Übung II.2 (Fixpunktiteration)

Die Lösung der nichtlinearen Gleichung

$$f(x) = x - \frac{1}{1 + x^2} = 0$$

soll numerisch berechnet werden.

- a) Berechnen Sie einen Näherungswert mit Hilfe einer Fixpunktiteration.
- a) Für welche Startwerte $x^{(0)}$ konvergiert das Verfahren? Handelt es sich dabei um monotone oder alternierende Konvergenz?

Übung II.3 (Fixpunktiteration)

Bestimmen Sie Startwerte für die Fixpunktiterationen

$$\text{a) } x^{(\text{neu})} = e^{-x^{(\text{alt})}} \qquad \text{b) } x^{(\text{neu})} = \ln(x^{(\text{alt})}) + \pi$$

Konvergiert die Fixpunktiteration? Handelt es sich dabei um monotone oder alternierende Konvergenz? Führen Sie jeweils ein paar Iterationsschritte durch.

Übung II.4 (Newton–Verfahren)

Gesucht sind die Schnittpunkte der beiden Kreise

$$f_1(x, y) = (x - 1)^2 + y^2 - 0.9 = 0,$$

$$f_2(x, y) = x^2 + (y - 1)^2 - 1.1 = 0.$$

a) Bestimmen Sie geeignete Startwerte für das Newton–Verfahren.

b) Wie lautet die Jacobi–Matrix \underline{J} beim Newton–Verfahren?

$$\begin{bmatrix} x^{(k+1)} \\ y^{(k+1)} \end{bmatrix} = \begin{bmatrix} x^{(k)} \\ y^{(k)} \end{bmatrix} - \underline{J}^{-1} \begin{bmatrix} f_1(x^{(k)}, y^{(k)}) \\ f_2(x^{(k)}, y^{(k)}) \end{bmatrix}$$

c) Führen Sie einen Schritt der Newton–Iteration mit einem Startwert aus a) durch.

Übung II.5 (Newton–Verfahren)

Die Funktion in zwei Veränderlichen

$$f(x, y) = x e^{-(x^2+y^2)}$$

besitzt genau ein absolutes Minimum und Maximum. Die Koordinaten der Extremwerte sollen berechnet werden. Dazu löst man das System von zwei nichtlinearen Gleichungen

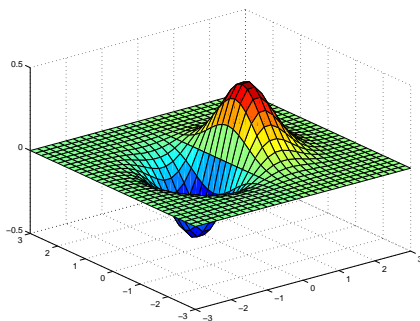
$$f_x(x, y) = (1 - 2x^2) e^{-(x^2+y^2)} = 0$$

$$f_y(x, y) = -2xy e^{-(x^2+y^2)} = 0$$

mit dem Newton–Verfahren.

a) Berechnen Sie die Jacobi–Matrix \underline{J} für das Newton–Verfahren

$$\begin{bmatrix} x^{(neu)} \\ y^{(neu)} \end{bmatrix} = \begin{bmatrix} x^{(alt)} \\ y^{(alt)} \end{bmatrix} - \underline{J}^{-1} \begin{bmatrix} f_x(x^{(alt)}, y^{(alt)}) \\ f_y(x^{(alt)}, y^{(alt)}) \end{bmatrix}.$$



- b) Bestimmen Sie geeignete Startwerte und führen Sie mit MATLAB ein paar Schritte der Newton-Iteration durch.

Hinweis Der Ausdruck $e^{-(x^2+y^2)}$ ist für alle Werte von x und y positiv. Dadurch vereinfachen sich die Berechnungen so, dass man die exakten Lösungen zur Kontrolle von Hand berechnen kann.

Übung II.6 (Newton-Fraktal)

Das System von zwei Gleichungen mit zwei Unbekannten

$$\begin{aligned} f_1(x, y) &= x^3 - 3xy^2 + 1 = 0 \\ f_2(x, y) &= 3x^2y - y^3 = 0 \end{aligned}$$

soll mit dem Newton-Verfahren gelöst werden.

- a) Wie lautet die Jacobi-Matrix \underline{J} beim Newton-Verfahren?

$$\begin{bmatrix} x^{(k+1)} \\ y^{(k+1)} \end{bmatrix} = \begin{bmatrix} x^{(k)} \\ y^{(k)} \end{bmatrix} - \underline{J}^{-1} \begin{bmatrix} f_1(x^{(k)}, y^{(k)}) \\ f_2(x^{(k)}, y^{(k)}) \end{bmatrix}$$

- b) Führen Sie einen Newton-Schritt mit dem Startwert $(1, 1)$ durch.
c) Es ist bekannt, dass das Gleichungssystem genau drei Lösungen

$$(x_1, y_1) = (-1, 0), \quad (x_2, y_2) = \left(\frac{1}{2}, \frac{1}{2}\sqrt{3}\right), \quad (x_3, y_3) = \left(\frac{1}{2}, -\frac{1}{2}\sqrt{3}\right)$$

besitzt. Die Frage ist nun, bei welchen Startwerten die Newton-Iteration konvergiert und welcher Startwert zu welcher der drei Lösungen führt. Diese Fragestellung führt zu mathematisch komplexen Problemen. Experimentell kann man jedoch einfach so vorgehen, dass man die Startwerte durch ein Programm testet. Das Fraktal in Abbildung II.5 wurde durch Testen von 500×500 Startwerten im Bereich von $[-1, 1] \times [-1, 1]$ erzeugt. Dabei wird die Iteration abgebrochen, sobald sich der Näherungswert um weniger als 0.0001 von

einer der drei Lösungen unterscheidet. Startwerte, die gegen (x_1, y_1) konvergieren werden rot gefärbt, entsprechend werden Startwerte, die gegen (x_2, y_2) bzw. (x_3, y_3) konvergieren grün bzw. blau gefärbt. Startwerte, die nach 50 Newton-Schritten gegen keine der drei Lösungen konvergieren, werden schwarz gefärbt. Lassen Sie das Fraktal in Abbildung II.5 durch ein Programm erzeugen.

Übung II.7 (GPS)

Bei der Startwertsuche für das GPS wird der Schnittpunkt von Kugeln durch ein lineares Gleichungssystem bestimmt. Dieser Trick lässt sich auch zur Berechnung der Schnittpunkte zweier Kreise anwenden. Falls sich die beiden Kreise mit den Mittelpunkten (x_1/y_1) , (x_2/y_2) und den Radien r_1 , r_2 schneiden, so liegen die beiden gesuchten Schnittpunkte auf einer Geraden g , der sogenannten Polare. Eine Gleichung der Geraden g erhält man durch Subtraktion der beiden Gleichungen

$$\begin{aligned}(x - x_1)^2 + (y - y_1)^2 &= r_1^2 \\ (x - x_2)^2 + (y - y_2)^2 &= r_2^2.\end{aligned}$$

- a) Bestimmen Sie eine Gleichung der Geraden g .
- b) Berechnen Sie die Polare g und die Schnittpunkte für die beiden Kreise mit den Mittelpunkten $(-1/-1)$ bzw. $(1/1)$ und den Radien 2 bzw. 1.

Übung II.8 (GPS)

Ein GPS-Empfänger empfängt von 4 Satelliten die folgenden Koordinaten (Einheit 10^6 m)

$$\begin{aligned}S_1(14.5160, 7.3963, 21.1552), & \quad S_2(16.0362, 2.2537, 21.1042), \\ S_3(20.2137, -1.0594, 17.1669), & \quad S_4(21.0161, -15.8368, -1.9786)\end{aligned}$$

und die Signallaufzeiten (Einheit 10^{-3} s)

$$t_1 = 68.3984, \quad t_2 = 67.5427, \quad t_3 = 67.9003, \quad t_4 = 81.9250.$$

Berechnen Sie die Koordinaten des GPS-Empfänger mit Hilfe des Newton-Verfahrens. Auf welchem Längen- und Breitenkreis befindet sich der GPS-Empfänger?

III Interpolation und Approximation

1. Polynomeninterpolation

Polynominterpolation

Bei der Polynominterpolation sucht man ein Polynom

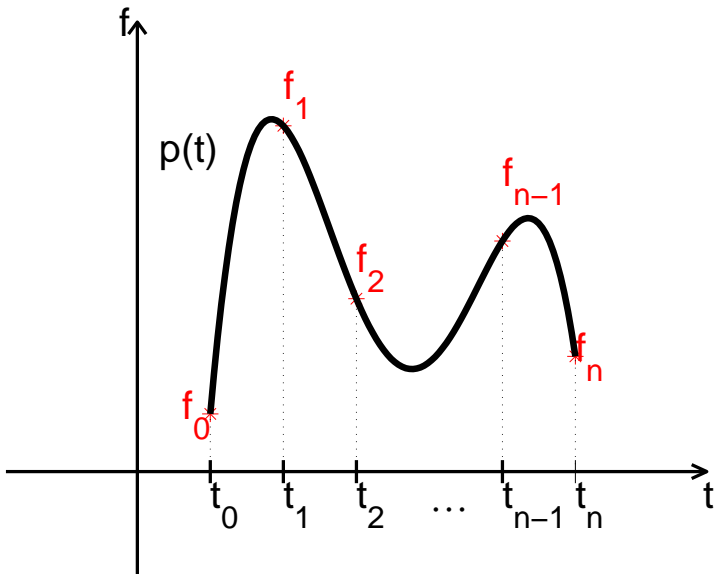
$$p(t) = a_0 + a_1t + a_2t^2 + \dots + a_nt^n,$$

das die Wertepaare

$$(t_0, f_0), \quad (t_1, f_1), \quad (t_2, f_2), \quad \dots, \quad (t_m, f_m),$$

interpoliert, d.h.

$$p(t_i) = f_i, \quad i = 0, 1, 2, \dots, m.$$



Polynominterpolation als lineares Gleichungssystem

Die Interpolationsbedingungen

$$p(t_i) = a_0 + a_1 t_i + \dots + a_n t_i^n, \quad i = 0, 1, 2, \dots, m$$

stellen ein lineares Gleichungssystem mit $m + 1$ Gleichungen für die $n + 1$ Unbekannten a_0, a_1, \dots, a_n dar

$$\begin{array}{cccccc} a_0 & + & a_1 t_0 & + & \dots & + & a_n t_0^n & = & y_0 \\ a_0 & + & a_1 t_1 & + & \dots & + & a_n t_1^n & = & y_1 \\ \vdots & & \vdots & & \ddots & & \vdots & = & \vdots \\ a_0 & + & a_1 t_m & + & \dots & + & a_n t_m^n & = & y_m \end{array}$$

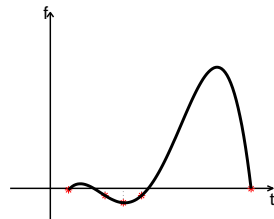
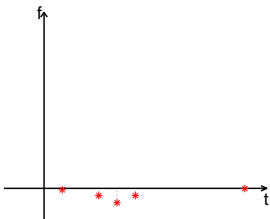
Vandermondsche Matrix

Die Koeffizienten a_0, a_1, \dots, a_n des Interpolationspolynoms $p(t)$ kann man durch lösen eines linearen Gleichungssystems berechnen.

$$\underbrace{\begin{bmatrix} 1 & t_0 & \dots & t_0^n \\ 1 & t_1 & \dots & t_1^n \\ \vdots & \vdots & \ddots & \vdots \\ 1 & t_m & \dots & t_m^n \end{bmatrix}}_{\text{Vandermondsche Matrix}} \cdot \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_m \end{bmatrix}$$

Das Gleichungssystem ist genau dann eindeutig lösbar, wenn $n = m$ ist und alle Parameterwerte t_i paarweise verschieden sind. Somit ist dann das Interpolationspolynom eindeutig bestimmt.

Ein Nachteil bei der Polynominterpolation besteht darin, dass aus der Lage der Interpolationspunkte oft der Verlauf des Interpolationspolynoms nicht klar vorhersehbar ist.



2. Newton–Schema

Idee beim Newton–Schema

Die Idee beim Newton–Schema besteht darin, eine Darstellung für das Interpolationspolynom zu wählen, die bereits die Parameterwerte enthält

$$p(t) = a_0 + a_1(t-t_0) + a_2(t-t_0)(t-t_1) + \dots + a_n(t-t_0)(t-t_1) \cdots (t-t_{n-1}).$$

dadurch lassen sich die Koeffizienten $a_0, a_1, a_2, \dots, a_n$ direkt ohne Lösen eines linearen Gleichungssystems berechnen.

Newton–Schema

$$\begin{array}{r|l}
 -3 & \frac{1}{10} \\
 -1 & \frac{1}{2} \quad \frac{1}{5} \\
 & \quad \frac{1}{2} \quad \frac{1}{10} \quad -\frac{15}{100} \\
 0 & 1 \quad -\frac{1}{2} \quad -\frac{1}{2} \quad \frac{7}{100} \quad \frac{22}{1000} \\
 & \quad -\frac{1}{2} \quad \frac{3}{50} \\
 1 & \frac{1}{2} \quad -\frac{2}{25} \\
 7 & \frac{1}{50}
 \end{array}$$

$$\begin{aligned}
 p(t) &= \frac{1}{10} + \frac{1}{5}(t+3) + \frac{1}{10}(t+3)(t+1) - \frac{15}{100}(t+3)(t+1)t \\
 &\quad + \frac{22}{1000}(t+3)(t+1)t(t-1)
 \end{aligned}$$

Newton–Schema (Teilschema)

$$\begin{array}{r|rrrrrr}
 -3 & \frac{1}{10} & & & & & \\
 -1 & \frac{1}{2} & \frac{1}{5} & & & & \\
 0 & 1 & \frac{1}{2} & \frac{1}{10} & -\frac{15}{100} & & \\
 1 & \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & \frac{7}{100} & \frac{22}{1000} & \\
 1 & \frac{1}{2} & -\frac{2}{25} & \frac{3}{50} & & & \\
 7 & \frac{1}{50} & & & & &
 \end{array}$$

$$p(t) = 1 - \frac{1}{2}t + \frac{3}{50}t(t-1)$$

$$p(0) = 1, \quad p(1) = \frac{1}{2}, \quad p(7) = \frac{1}{50}.$$

Newton–Schema (Reihenfolge)

Reihenfolge

Die Werte im Differenzenschema können in beliebiger Reihenfolge angeordnet werden.

$$\begin{array}{r|rrr}
 0 & 1 & & \\
 1 & \frac{1}{2} & -\frac{1}{2} & \\
 -1 & \frac{1}{2} & 0 & -\frac{1}{2}
 \end{array}$$

$$p(t) = 1 - \frac{1}{2}t - \frac{1}{2}t(t-1)$$

$$\begin{array}{r|rrr}
 1 & \frac{1}{2} & & \\
 -1 & \frac{1}{2} & 0 & -\frac{1}{2} \\
 0 & 1 & \frac{1}{2} &
 \end{array}$$

$$p(t) = \frac{1}{2} - \frac{1}{2}(t-1)(t+1)$$

3. Hermite–Interpolation

Newton–Schema (Ableitungen)

Daten für $p(t) = t^3$

t_i	0	2	3
f_i	0	8	27
f'_i		12	27
f''_i		12	

0	0				
2	8	4			
2	8	$\frac{12}{1!}$	1		
2	8	$\frac{12}{2!}$	$\frac{12}{2!}$	0	
2	8	$\frac{12}{1!}$	7	0	0
		19	1		
3	27	8			
		$\frac{27}{1!}$			
3	27				

$$p(t) = 0 + 4t + 4t(t - 2) + 1t(t - 2)^2 + 0t(t - 2)^3 + 0t(t - 2)^3(t - 3)$$

4. Ausgleichspolynome

Beispiel III.1 (Ausgleichsgerade)

Wir suchen eine Gerade

$$p(t) = a_0 + a_1t,$$

die möglichst gut zu den Punkten

$$(0, 0), \quad (1, 1), \quad (2, 4)$$

passt. Zunächst müssen wir einen Weg finden, das Problem mathematisch zu formulieren. Dazu betrachten wir die Fehler zwischen der Geraden und den gewünschten Funktionswerten

$$e_0 = p(0) - 0, \quad e_1 = p(1) - 1, \quad e_2 = p(2) - 4.$$

Natürlich könnten wir die Gerade so bestimmen, dass jeweils zwei dieser Fehler Null werden, allerdings wird dadurch der Fehler am dritten Punkt relativ groß. Bei der Methode der kleinsten Fehlerquadrate bestimmt man die Gerade so, dass die Summe der Fehlerquadrate

$$e = e_0^2 + e_1^2 + e_2^2 \rightarrow \min$$

minimal wird. Der Gesamtfehler e ist eine Funktion, die nur noch von a_0 und a_1 abhängt

$$e(a_0, a_1) = a_0^2 + (a_0 + a_1 - 1)^2 + (a_0 + 2a_1 - 4)^2.$$

Die Bedingungen

$$\begin{aligned} e_{a_0}(a_0, a_1) &= 2a_0 + 2(a_0 + a_1 - 1) + 2(a_0 + 2a_1 - 4) = 0 \\ e_{a_1}(a_0, a_1) &= 2(a_0 + a_1 - 1) + 4(a_0 + 2a_1 - 4) = 0 \end{aligned}$$

sind notwendig für ein Minimum und bilden ein lineares Gleichungssystem

$$\begin{aligned} 6a_0 + 6a_1 &= 10, \\ 6a_0 + 10a_1 &= 18, \end{aligned}$$

das die eindeutige Lösung

$$a_0 = -\frac{1}{3}, \quad a_1 = 2$$

hat. Die gesuchte Gerade ist somit

$$p(t) = -\frac{1}{3} + 2t.$$

Die Vorgehensweise aus Beispiel III.1 lässt sich auf Polynome vom Grad n

$$p(t) = a_0 + a_1 t + a_2 t^2 + \dots + a_n t^n$$

verallgemeinern. Zu den Daten

$$(t_0, f_0), \quad (t_1, f_1), \quad (t_2, f_2), \quad \dots, \quad (t_m, f_m)$$

bestimmen wir das Polynom so, dass die Summe der Fehlerquadrate

$$e(a_0, a_1, \dots, a_n) = (p(t_0) - f_0)^2 + (p(t_1) - f_1)^2 + \dots + (p(t_n) - f_n)^2$$

minimal wird. Die einzelnen Fehler lassen sich mit Hilfe der Vandermond-schen Matrix sehr elegant darstellen

$$\underbrace{\begin{bmatrix} e_0 \\ e_1 \\ \vdots \\ e_n \end{bmatrix}}_{\underline{e}} = \underbrace{\begin{bmatrix} 1 & t_0 & \dots & t_0^n \\ 1 & t_1 & \dots & t_1^n \\ \vdots & \vdots & \ddots & \vdots \\ 1 & t_m & \dots & t_m^n \end{bmatrix}}_{\underline{A}} \cdot \underbrace{\begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix}}_{\underline{a}} - \underbrace{\begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_n \end{bmatrix}}_{\underline{f}}.$$

Die Summe der Fehlerquadrate lässt sich auch in Form eines Matrizenproduktes darstellen

$$e_0^2 + e_1^2 + \dots + e_n^2 = \underbrace{[e_0 \quad e_1 \quad \dots \quad e_n]}_{\underline{e}^T} \cdot \begin{bmatrix} e_0 \\ e_1 \\ \vdots \\ e_n \end{bmatrix}.$$

Dabei bezeichnet \underline{e}^T die transponierte Matrix von \underline{e} . Insgesamt ergibt sich somit die Darstellung

$$\begin{aligned} e(\underline{a}) &= \underline{e}^T \cdot \underline{e} \\ &= (\underline{A} \cdot \underline{a} - \underline{f})^T \cdot (\underline{A} \cdot \underline{a} - \underline{f}) \\ &= (\underline{a}^T \cdot \underline{A}^T - \underline{f}^T) \cdot (\underline{A} \cdot \underline{a} - \underline{f}) \\ &= \underline{a}^T \cdot \underline{A}^T \cdot \underline{A} \cdot \underline{a} - \underline{a}^T \cdot \underline{A}^T \cdot \underline{f} - \underline{f}^T \cdot \underline{A} \cdot \underline{a} + \underline{f}^T \cdot \underline{f} \\ &= \underline{a}^T \cdot \underline{A}^T \cdot \underline{A} \cdot \underline{a} - 2\underline{a}^T \cdot \underline{A}^T \cdot \underline{f} + \underline{f}^T \cdot \underline{f}. \end{aligned}$$

In dieser Form lassen sich die partiellen Ableitungen berechnen

$$\frac{\partial e(\underline{a})}{\partial \underline{a}} = 2\underline{A}^T \cdot \underline{A} \cdot \underline{a} - 2\underline{A}^T \cdot \underline{f}$$

und die notwendigen Bedingungen für ein Minimum

$$\frac{\partial e(\underline{a})}{\partial \underline{a}} = \underline{0}$$

liefern die sogenannten Normalgleichungen

$$\underline{A}^T \cdot \underline{A} \cdot \underline{a} = \underline{A}^T \cdot \underline{f}.$$

Normalgleichungen

Zur Approximation der Daten

$$(t_0, f_0), \quad (t_1, f_1), \quad (t_2, f_2), \quad \dots, \quad (t_m, f_m)$$

durch eine Ausgleichspolynom

$$p(t) = a_0 + a_1 t + a_2 + \dots + a_n t^n$$

berechnet man die Koeffizienten $a_0, a_1, a_2, \dots, a_n$ aus den Normalgleichungen

$$\underline{A}^T \cdot \underline{A} \cdot \underline{a} = \underline{A}^T \cdot \underline{f}.$$

Dabei bezeichnet \underline{A} die Vandermondsche Matrix, \underline{a} den Vektor der gesuchten Polynomkoeffizienten und \underline{f} den Vektor mit den Funktionswerten.

Beispiel III.2 (Normalgleichungen einer Ausgleichsgerade)

Wir betrachten nochmals die Problemstellung aus Beispiel III.1. Das über-

bestimmt lineare Gleichungssystem lautet

$$\underbrace{\begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \end{bmatrix}}_A \cdot \underbrace{\begin{bmatrix} a_0 \\ a_1 \end{bmatrix}}_a = \underbrace{\begin{bmatrix} 0 \\ 1 \\ 4 \end{bmatrix}}_f.$$

Wir multiplizieren dieses System mit A^T

$$\underbrace{\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 2 \end{bmatrix}}_{A^T} \cdot \underbrace{\begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \end{bmatrix}}_A \cdot \underbrace{\begin{bmatrix} a_0 \\ a_1 \end{bmatrix}}_a = \underbrace{\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 2 \end{bmatrix}}_{A^T} \cdot \underbrace{\begin{bmatrix} 0 \\ 1 \\ 4 \end{bmatrix}}_f$$

und erhalten das selbe lineare Gleichungssystem

$$\underbrace{\begin{bmatrix} 3 & 3 \\ 3 & 5 \end{bmatrix}}_{A^T \cdot A} \cdot \underbrace{\begin{bmatrix} a_0 \\ a_1 \end{bmatrix}}_a = \underbrace{\begin{bmatrix} 5 \\ 9 \end{bmatrix}}_{A^T \cdot f}$$

wie in Beispiel III.1.

Übung III.1 (Newton-Schema)

Gegeben sind die Punkte

$$P_0(-1/-1), \quad P_1(0/1), \quad P_2(2/-1).$$

- Berechnen Sie das Polynom, das diese Punkte interpoliert, mit dem Newton-Schema.
- Erweitern Sie das Newton-Schema aus Teil a) so, dass der Punkt P_2 mit der Steigung 1 interpoliert wird.
- Skizzieren Sie die Schaubilder der beiden Polynome aus dem a) und b) Teil.

Übung III.2 (Newton–Schema)

Die Punkte

$$P_0(-3/0.1), P_1(-1/0.5), P_2(0/1), P_3(1/0.5), P_4(1/0.02)$$

sollen so interpoliert werden, dass das Interpolationspolynom im Punkt P_4 eine waagrechte Tangente besitzt.

- Welchen Grad n würden Sie für das Interpolationspolynom wählen? Begründen Sie Ihre Wahl.
- Berechnen Sie das Interpolationspolynom $p_n(x)$.

Übung III.3 (Newton–Schema)

Berechnen Sie das Polynom, das in $(0/0)$ einen Wendepunkt mit Steigung 1 hat. Ausserdem soll in $(-1/-1)$ ein Tiefpunkt und in $(1/1)$ ein Hochpunkt liegen.

Übung III.4 (Ausgleichsparabel)

Bestimmen Sie die Koeffizienten der Ausgleichsparabel

$$p_2(x) = a_0 + a_1x + a_2x^2$$

zu den Messwerten

x	1.00	2.00	3.00	4.00	5.00	6.00	7.00	8.00	9.00	10.00
y	5.95	11.23	19.60	29.48	41.89	55.76	71.45	89.01	109.82	131.44

IV Numerische Integration

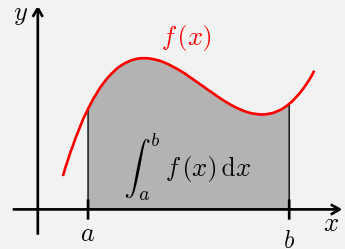
Es gibt unterschiedliche Gründe, die eine Annäherung von Integralen durch numerische Werte erfordern. Beispielsweise existieren elementare Funktionen, wie z.B. e^{x^2} , für die die Stammfunktion nicht durch elementare Funktionen darstellbar ist. Bei Problemen aus der Praxis ist man oftmals schon deshalb auf numerische Integrationsmethoden angewiesen, weil man keine analytische Darstellung der Funktion selbst kennt. In diesen Fällen kann man lediglich Funktionswerte für bestimmte Parameterwerte bestimmen und daraus versuchen einen möglichst guten Näherungswert für das Integral zu erzeugen.

1. Problemstellung

Numerische Integration

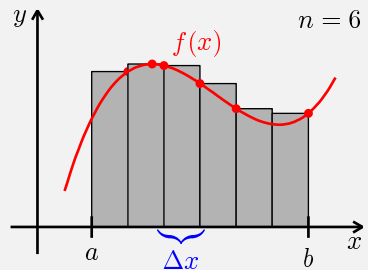
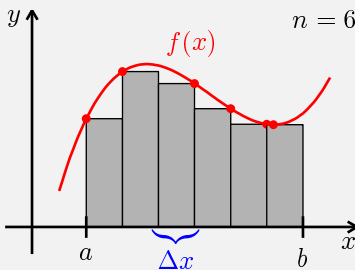
Numerische Integrationsverfahren verwendet man zur Berechnung eines Näherungswertes des bestimmten Integrals

$$A = \int_a^b f(x) dx.$$



Unter- und Obersummen

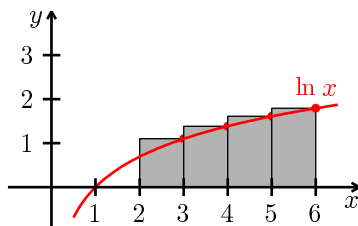
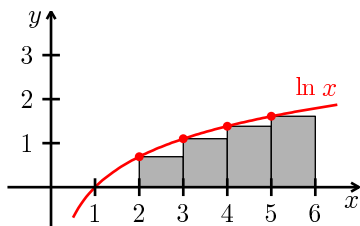
Die Fläche unter einer stetigen positiven Funktion kann durch Untersumme und Obersumme angenähert werden. Der tatsächliche Wert der Fläche ist sicherlich nicht kleiner als die Untersumme und sicherlich auch nicht größer als die Obersumme.



Beispiel IV.1 (Unter- und Obersumme)

Die Fläche unter der Funktion $f(x) = \ln x$ für x -Werte zwischen 2 und

6 soll mit Hilfe von Unter- und Obersumme näherungsweise berechnet werden.



Eine grobe Abschätzung erhält man mit $n = 4$ Rechtecken deren Grundseiten die Länge $\Delta x = 1$ haben. Die Untersumme ergibt

$$U = 1 \cdot \ln 2 + 1 \cdot \ln 3 + 1 \cdot \ln 4 + 1 \cdot \ln 5 = \ln(2 \cdot 3 \cdot 4 \cdot 5) = \ln 120 \approx 4.7875$$

und die Obersumme

$$O = 1 \cdot \ln 3 + 1 \cdot \ln 4 + 1 \cdot \ln 5 + 1 \cdot \ln 6 = \ln(3 \cdot 4 \cdot 5 \cdot 6) = \ln 360 \approx 5.8861.$$

Der Wert des bestimmten Integrals muss somit zwischen den beiden berechneten Werten liegen,

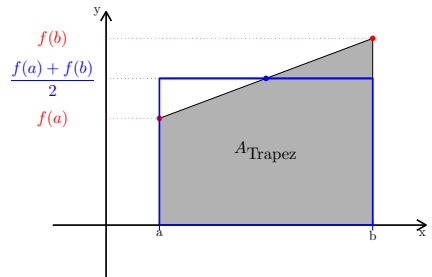
$$\ln 120 \leq \int_2^6 \ln x \, dx \leq \ln 360.$$

2. Trapezregel

Eine deutliche Verbesserung bei der numerischen Berechnung des bestimmten Integrals erzielt man, in dem man anstelle von Rechtecken Trapeze verwendet.

Die Fläche eines einzelnen Trapezes ist genau gleich groß wie die Fläche des Rechtecks, das die selbe Grundseite hat und dessen Höhe gerade dem Mittelwert der Höhen des Trapezes entspricht,

$$A_{\text{Trapez}} = (b - a) \cdot \frac{f(a) + f(b)}{2}.$$



Wenn man das Intervall $[a; b]$ in n gleichlange Teilintervalle unterteilt, dann hat jedes Trapez eine Grundseite der Länge

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

und die Funktion wird an insgesamt $n + 1$ Stellen ausgewertet

$$f(a), f(a + h), f(a + 2h), f(a + 3h), \dots, f(b - h), f(b).$$

Die Summe aller n Trapezflächen ergibt dann

$$\begin{aligned} A &= h \cdot \frac{f(a) + f(a + h)}{2} \\ &+ h \cdot \frac{f(a + h) + f(a + 2h)}{2} \\ &+ h \cdot \frac{f(a + 2h) + f(a + 3h)}{2} \\ &+ \dots \\ &+ h \cdot \frac{f(b - h) + f(b)}{2}. \end{aligned}$$

Alle Funktionswerte außer dem ersten und dem letzten liefern einen Beitrag zu zwei Trapezen. Dadurch kann man die Formel noch etwas vereinfachen.

Trapezregel

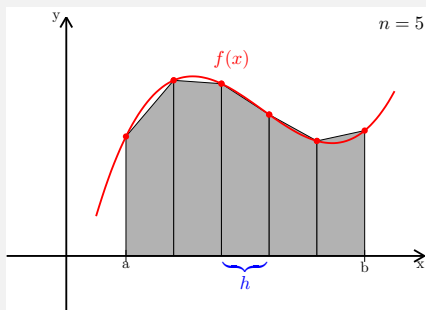
Das bestimmte Integral einer Funktion $f(x)$ im Intervall $[a; b]$

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

kann man durch eine Summe von n Trapezflächen annähern. Die Trapeze haben eine Grundseite der Länge $h = \frac{b-a}{n}$.

Die Funktionswerte müssen an $n+1$ Stellen berechnet werden, die Formel zur Berechnung der Summe der Trapezflächen lautet

$$T(h) = h \left(\frac{1}{2} f(a) + f(a+h) + f(a+2h) + \dots + f(b-h) + \frac{1}{2} f(b) \right).$$



Beispiel IV.2 (Trapezregel für $\ln x$)

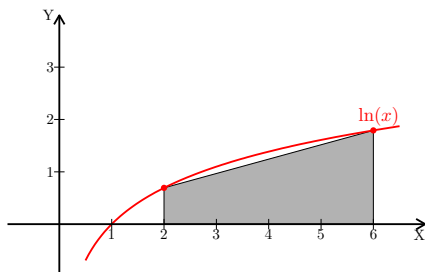
Die Fläche unter der Funktion $f(x) = \ln x$ für x -Werte zwischen 2 und 6 soll mit Hilfe der Trapezregel näherungsweise berechnet werden.

Für $n = 1$ ergibt sich

$$h = \frac{b-a}{1} = \frac{6-2}{1} = 4$$

und die Formel liefert

$$T(4) = 4 \cdot \left(\frac{1}{2} \ln 2 + \frac{1}{2} \ln 6 \right) \approx 4.9698.$$

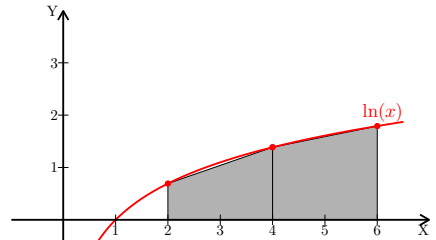


Bei $n = 2$ liefert die Schrittweite

$$h = \frac{b - a}{2} = \frac{6 - 2}{2} = 2$$

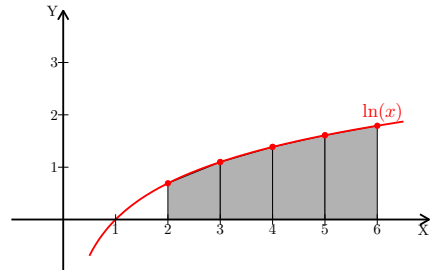
den Näherungswert

$$T(2) = 2 \cdot \left(\frac{1}{2} \ln 2 + \ln 4 + \frac{1}{2} \ln 6 \right) \approx 5.2013.$$



Für $n = 4$ ist aus der Grafik kaum noch ein Unterschied zwischen der Originalfläche und den Trapezen zu erkennen. Die Grundseite der Trapeze haben alle die Länge

$$h = \frac{b - a}{4} = \frac{6 - 2}{4} = 1.$$



Die Formel ergibt

$$T(4) = 1 \cdot \left(\frac{1}{2} \ln 2 + \ln 3 + \ln 4 + \ln 5 + \frac{1}{2} \ln 6 \right) \approx 5.3368.$$

Der Näherungswert der Trapezregel resultiert in einem deutlich besseren Ergebnis als die Annäherung durch Unter- und Obersumme

$$\int_2^6 \ln x \, dx \approx 5.3368.$$

3. Rombergverfahren

Das Rombergverfahren verbessert die Ergebnisse der Trapezregel, dabei werden zwei grundlegende Prinzipien der Mathematik verwendet. Zum

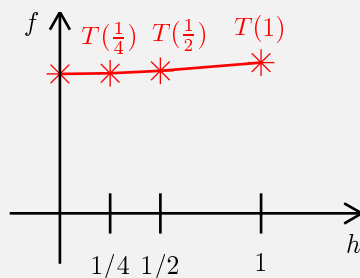
einen basiert das Rombergverfahren auf einer Extrapolationsidee und zum andern werden die Parameterwerte an denen man die Funktion auswertet geschickt gewählt.

Wenn man bei der Trapezregel von Schritt zu Schritt n verdoppelt, dann kann man alle zuletzt berechneten Werte „recyclen“

n	$f(a)$		$f(a + \frac{b-a}{n})$		\dots	$f(b)$
$2n$	$f(a)$	$f(a + \frac{b-a}{2n})$	$f(a + 2\frac{b-a}{2n})$	$f(a + 3\frac{b-a}{2n})$	\dots	$f(b)$

Extrapolation

Der Grundgedanke bei der Extrapolation besteht darin, aus einer Sequenz von Näherungswerten den exakten Wert zu extrapolieren.



Fehlerentwicklung der Trapezregel

Für genügend oft differenzierbare Funktionen $f(x)$ besitzt die Trapezregel eine quadratische Fehlerentwicklung bezüglich der Schrittweite h , d.h.

$$\int_a^b f(x) \, dx - T(h) = c_2 h^2 + c_4 h^4 + c_6 h^6 + \dots$$

Dabei sind c_2, c_4, c_6, \dots Konstanten, die zwar von der Funktion $f(x)$ aber nicht von der Schrittweite h abhängen.

Da wir die asymptotische Fehlerentwicklung kennen, lässt sich die Idee der Extrapolation gewinnbringend auf die Trapezregel anwenden. Aus den beiden Werten $T(2h)$ und $T(h)$ erhalten wir

$$\begin{aligned} \int_a^b f(x) \, dx - T(2h) &= 4 \cdot c_2 h^2 + 16 \cdot h^4 + \dots, \\ \int_a^b f(x) \, dx - T(h) &= c_2 h^2 + h^4 + \dots \end{aligned}$$

Wenn wir nun die zweite Gleichung mit dem Faktor 4 multiplizieren und die erste Gleichung abziehen ergibt sich

$$3 \int_a^b f(x) \, dx - (4T(h) - T(2h)) = \tilde{c}_4 h^4 + \dots$$

Mit anderen Worten

$$\frac{4T(h) - T(2h)}{3}$$

ist eine Näherung, bei der der Fehler nur noch Terme der Ordnung 4 oder höher enthält.

Beispiel IV.3 (Trapezregel für $\frac{1}{x}$)*Problemstellung*

$$I = \int_1^2 \frac{1}{x} dx$$

Trapezregel für $n = 1, n = 2, n = 4$:

$$n = 1, \quad h = \frac{2-1}{1} = 1, \quad T(1) = 1\left(\frac{1}{2} \cdot \frac{1}{1} + \frac{1}{2} \cdot \frac{1}{2}\right) = 0.75$$

$$n = 2, \quad h = \frac{2-1}{2} = \frac{1}{2}, \quad T\left(\frac{1}{2}\right) = \frac{1}{2}\left(\frac{1}{2} \cdot \frac{1}{1} + \frac{1}{1.5} + \frac{1}{2} \cdot \frac{1}{2}\right) = 0.7083$$

$$n = 4, \quad h = \frac{2-1}{4} = \frac{1}{4}, \quad T\left(\frac{1}{4}\right) = \frac{1}{4}\left(\frac{1}{2} \cdot \frac{1}{1} + \frac{1}{1.25} + \frac{1}{1.5} + \frac{1}{1.75} + \frac{1}{2} \cdot \frac{1}{2}\right) = 0.6970$$

Romberg-Tableau:

h	$T(h)$	$k = 1$	$k = 2$
1	0.7500		
$\frac{1}{2}$	0.7083	0.6944	0.6931
$\frac{1}{4}$	0.6970	0.6932	

Bemerkenswert ist, dass der beste Näherungswert in der ersten Spalte nur mit zwei Nachkommastellen mit dem exakten Wert $\ln 2 \approx 0.6931$ übereinstimmt. Das Ergebnis des Romberg-Tableau liefert jedoch vier korrekte Nachkommastellen.

Allgemein verwendet man zur Berechnung der verbesserten Näherungswerte in der k -ten Spalte die Formel für die Gewichte

$$\text{neuer Wert} = \underbrace{\frac{-1}{4^k - 1}}_{<0} \cdot \text{„schlechterer“ Wert} + \underbrace{\frac{4^k}{4^k - 1}}_{>1} \cdot \text{„besserer“ Wert.}$$

Die beiden Gewichte summieren zu 1, wobei das erste Gewicht negativ ist und das zweite deshalb größer als 1.

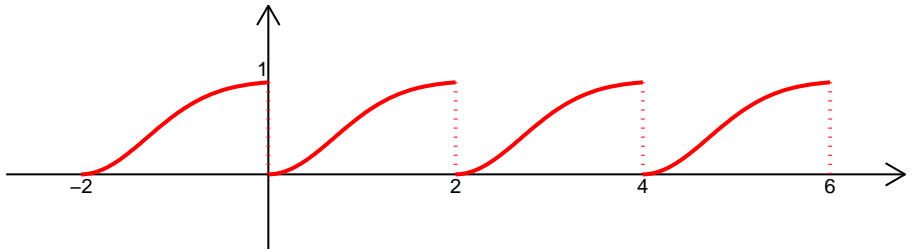
4. Übungen

Übung IV.1 (Romberg–Verfahren)

Die Funktion

$$f(t) = 1 - e^{-t^2}, \quad t \in [0, 2],$$

wird zu einer Funktion mit Periode 2 für alle reellen Zahlen fortgesetzt.



Entsprechend kann man die Fourier–Koeffizienten aus den Formeln

$$a_0 = \int_0^2 (1 - e^{-t^2}) dt, \quad a_k = \int_0^2 (1 - e^{-t^2}) \cos(k\pi t) dt, \quad b_k = \int_0^2 (1 - e^{-t^2}) \sin(k\pi t) dt$$

für $k = 1, 2, 3, \dots$ berechnen.

- Berechnen Sie den Mittelwert $\frac{a_0}{2}$ mit dem Romberg–Verfahren bis auf 4 Stellen nach dem Komma.
- Lassen Sie a_1, a_2, \dots, a_5 mit Hilfe des Romberg–Verfahrens durch ein MATLAB Programm berechnen und stellen Sie die Approximation

$$s_5(t) = \frac{a_0}{2} + a_1 \cos(\pi t) + a_2 \cos(2\pi t) + \dots + a_5 \cos(5\pi t),$$

grafisch dar.

Übung IV.2 (Romberg-Verfahren)

Der Wert des Integrals

$$I = \int_0^2 e^{-x^2} dx$$

soll numerisch berechnet werden.

- a)** Berechnen Sie einen groben Näherungswert $T(2)$ mit der Trapezregel zur Schrittweite $h = 2$.
- b)** Verwenden Sie das Romberg-Verfahren um das Integral bis auf vier Nachkommastellen numerisch zu bestimmen.
- c)** Vergleichen Sie das Romberg-Verfahren mit dem Integrationsverfahren `quad` in MATLAB.

V Gewöhnliche Differentialgleichungen

1. Numerische Differentiation

Bevor wir uns mit numerischen Lösungsverfahren für gewöhnliche Differentialgleichungen beschäftigen, betrachten wir die numerische Berechnung von Ableitungen. Die Ableitung einer Funktion $f(x)$ an einem Wert x_0 ist definiert als Grenzwert

$$f'(x_0) = \lim_{h \rightarrow 0} \frac{f(x_0 + h) - f(x_0)}{h}.$$

Es liegt also sehr nahe ein geeignetes h festzulegen und damit einen Näherungswert

$$\tilde{f}'(x_0) = \frac{f(x_0 + h) - f(x_0)}{h}.$$

zu bestimmen. Die Methode klingt zwar relativ simpel, bei praktischen Anwendungen sind dabei aber ein paar Schwierigkeiten zu überwinden. Je kleiner man das h wählt um so näher liegen in der Theorie der Näherungswert $\tilde{f}'(x_0)$ und der exakte Wert $f'(x_0)$ beieinander. Allerdings kann man h nur so klein wählen, dass der Rundungsfehler keinen zu großen Einfluß

hat. Schließlich erfordert die Berechnung die Division von zwei Zahlen, die nahe bei Null liegen.

Optimale Schrittweite

Eine zu große Schrittweite h verschlechtert den Verfahrensfehler $V(h)$ und eine zu kleine Schrittweite verschlechtert den Rundungsfehler $R(h)$. Das Ziel der Numerik ist es, eine der Problemstellung optimal angepasste Schrittweite h so zu bestimmen, dass der Gesamtfehler

$$G(h) = V(h) + R(h)$$

minimal wird.

Beispiel V.1 (Optimale Schrittweite)

Der Wert der Ableitung der Funktion $f(x) = e^x$ an der Stelle $x_0 = 0$ soll mit der Formel

$$\tilde{f}'(0) = \frac{e^h - e^0}{h}$$

numerisch ermittelt werden.

2. Anfangswertprobleme

Man spricht zwar von numerischen Lösungsverfahren für gewöhnliche Differentialgleichungen, genau genommen meint man aber numerische Verfahren zur Lösung von Anfangswertproblemen. Man sucht also diejenige Lösung $x(t)$ einer Differentialgleichung, die zu gegebenem Anfangszeitpunkt t_0 den Anfangswert $x(t_0) = x_0$ hat.

Außerdem benötigen numerische Verfahren für alle Größen in der Differentialgleichung konkrete Werte. Abhängigkeiten der Lösung von bestimmten Parameterwerten lassen sich nicht unmittelbar erkennen.

Ein numerisches Verfahren liefert als Lösung eines Anfangswertproblems eine Folge von Näherungswerten. In der Praxis fordert man, dass diese Näherungswerte bis auf eine vorgegebene Toleranz mit der mathematisch exakten Lösung übereinstimmen. Im folgenden verwenden wir für Näherungswerte stets das Symbol $\tilde{}$.

3. Euler–Verfahren

Herleitung

$$\dot{x}(t) \approx \frac{x(t+h) - x(t)}{h}$$

Polygonzugverfahren von Euler

Mit dem Polygonzugverfahren von Euler kann man für das Anfangswertproblem

$$\dot{x}(t) = f(t, x(t)), \quad x(t_0) = x_0$$

mit der Iterationsvorschrift

$$\begin{aligned}\tilde{x}_{k+1} &= \tilde{x}_k + h \cdot f(t_k, \tilde{x}_k) \\ t_{k+1} &= t_k + h\end{aligned}$$

Schritt für Schritt Näherungswerte \tilde{x}_k für die exakte Lösung $x(t_0 + k \cdot h)$ berechnen. Dabei hat die gewählte Schrittweite h großen Einfluß auf den Fehler der Näherungswerte.

Beispiel V.2 (Euler–Verfahren)

$$\dot{x}(t) = x(t), \quad x(0) = 1.$$

Beispiel V.3 (Fadenpendel)

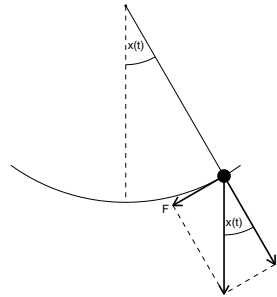
$$\text{Bogenlänge} \quad s(t) = l \cdot x(t)$$

$$\text{Geschwindigkeit} \quad v(t) = l \cdot \dot{x}(t)$$

$$\text{Beschleunigung} \quad a(t) = l \cdot \ddot{x}(t)$$

$$\text{Newton} \quad \vec{F} = m \cdot a(t)$$

$$\text{Reibungskraft} \quad \vec{F}_\delta = -\delta \cdot v(t)$$



Rücktreibende Kraft

$$\vec{F}_R = -m \cdot g \cdot \sin x(t)$$

Differentialgleichung

$$m \cdot l \cdot \ddot{x}(t) + \delta \cdot \dot{x}(t) + m \cdot g \cdot \sin x(t) = 0$$

Zustandsvariablen

Differentialgleichungen höherer Ordnung kann man mit Hilfe von Zustandsvariablen

$$z_1(t) = x(t)$$

$$z_2(t) = \dot{x}(t)$$

$$\vdots \quad \quad \quad \vdots$$

in ein äquivalentes System von Differentialgleichungen erster Ordnung

$$\underline{\dot{z}}(t) = \underline{f}(t, \underline{z}(t))$$

überführen. Dabei enthält der Vektor $\underline{z}(t)$ alle Zustandsvariablen und der Vektor $\underline{f}(t, \underline{z}(t))$ alle Funktionen.

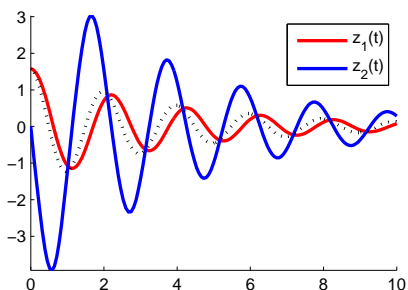
Beispiel V.4 (Euler-Verfahren für das Fadenpendel)

$m = 1\text{kg}$, $l = 1\text{m}$, $g = 10\frac{\text{m}}{\text{s}^2}$, $\delta = 1/2$:

$$\begin{aligned} \dot{z}_1(t) &= z_2(t), & z_1(0) &= \pi/2, \\ \dot{z}_2(t) &= -1/2 z_2(t) - 10 \sin z_1(t), & z_2(0) &= 0. \end{aligned}$$

Euler-Verfahren ($h = 0.1$)

t	\tilde{z}_1	\tilde{z}_2
0.0	1.57	0.00
0.1	1.57	-1.00
0.2	1.47	-1.95
\vdots	\vdots	\vdots



```
function dydt = pendel(t,y)
% function dydt = pendel(t,y)
%
% DGL des Fadenpendels
%
% m*l*x''(t)+delta*x'(t)+m*g*sin(x(t))=0
m = 1; % [kg]
l = 1; % [m]
g = 10; % [m/s^2]
delta = 0.5;
dydt = [y(2,:);
        -delta/m/l*y(2,:)-g/l*sin(y(1,:))];
```

```
function [t,y] = euler_h(f,tspan,y0,h)
% function [t,y] = euler_h(f,tspan,y0,h)
%
% Euler-Verfahren
%
t0 = tspan(1);
n = (tspan(2)-tspan(1))/h;
t = t0:h:tspan(2);
y = [y0'; zeros(n,2)];
for i = 1:n
    y0 = y0 + h*feval(f,t0,y0);
    t0 = t0 + h;
    y(i+1,:) = y0';
end
```

```
%
% Demo für Euler-Verfahren
%
h = 0.001;
[t,y] = euler_h('pendel',[0;10],[pi/2;0],h);
clf;
title(['h = ' num2str(h)]);
plot(t,y(:,1),'.r',t,y(:,2),'.b');
```

4. Globaler und lokaler Fehler

5. Schrittweitensteuerung

6. Übungen

Übung V.1 (Numerisches Differenzieren)

Die Ableitung der Funktion

$$f(x) = \frac{1}{1+x^2}$$

an der Stelle $x_0 = 1$ soll mit Hilfe des Differenzenquotienten

$$f'(x) \approx \frac{f(x_0 + h) - f(x_0)}{h}$$

numerisch berechnet werden.

- a) *Bestimmen Sie experimentell die optimale Schrittweite h für die Formel*

$$f'(x) \approx \frac{\frac{1}{1+(1+h)^2} - \frac{1}{2}}{h}.$$

- b) *Stellen Sie die Formel so um, dass der Rundungsfehler für kleine h -Werte verbessert wird.*

Übung V.2 (Differentialgleichung des menschlichen Herzens)

Komplizierte biologische Abläufe versucht man mit Hilfe mathematischer Modelle besser zu verstehen. Die Differentialgleichung

$$\ddot{x}(t) = -100 (\dot{x}(t)x^2(t) - B\dot{x}(t) + x(t) - L_0)$$

beschreibt ein sehr vereinfachtes Modell des menschlichen Herzens. Dabei bezeichnet $x(t)$ die Veränderung der Länge des Herzmuskels in Abhängigkeit der Zeit t , L_0 die Länge des Herzmuskels im Ruhezustand und B den Blutdruck.

- a) Überführen Sie die Differentialgleichung durch Einführen geeigneter Zustandsgrößen in ein äquivalentes Differentialgleichungssystem erster Ordnung.
- b) Realisieren Sie Differentialgleichung in Form einer MATLAB-Funktion:

```
function dzdt = ekg(t, z)
% function dzdt = ekg(t, z)
%
% Differentialgleichung für EKG
%
```

- c) Simulieren Sie den Herzschlag mit Hilfe des Euler-Verfahrens mit der Schrittweite $h = 0.001$ bei den Parameterwerten $B = 1$ und $L_0 = 1/2$. Wählen Sie dazu geeignete Anfangswerte.
- d) Wie ändert sich das "Elektrokardiogramm (EKG)" bei den Parameterwerten $B = 0.76$, $B = 0.75$, $B = 0.74$, $B = 0.5$, $B = 1.5$ und $B = 2$?

Übung V.3 (Van der Polsche Differentialgleichung)

Die Van der Polsche Differentialgleichung

$$\ddot{x}(t) + \underbrace{\mu(x^2(t) - 1)}_{\delta(x(t))} \dot{x}(t) + x(t) = f(t),$$

beschreibt ein schwingungsfähiges System. Wir betrachten nur den einfachen Fall ohne äußere Erregung, d.h. mit $f(t) = 0$. Im Gegensatz zur linearen homogenen Schwingungsdifferentialgleichung

$$\ddot{x}(t) + \delta \dot{x}(t) + x(t) = 0,$$

hängt die Dämpfung δ bei der Van der Polsche Differentialgleichung von $x(t)$ ab. Für kleine x -Werte ist der Dämpfungsfaktor δ negativ und die

Schwingung wird dadurch angeregt. Für große x -Werte ist der Dämpfungsfaktor positiv und die Schwingung wird dadurch gedämpft. Durch dieses Zusammenspiel stirbt das System für $t \rightarrow \infty$ unabhängig von den Anfangsbedingungen in einen sogenannten Grenzyklus. Da man die Lösungen der Van der Polsche Differentialgleichung nicht in geschlossener Form angeben kann, wollen wir den Grenzyklus für $\mu = 1$ mit den Anfangswerten $x(0) = 1$, $\dot{x}(0) = 0$ im Zeitintervall $t \in [0, 100]$ numerisch bestimmen.

- a) Überführen Sie die Van der Polsche Differentialgleichung durch Einführen geeigneter Zustandsgrößen in ein äquivalentes Differentialgleichungssystem erster Ordnung.
- b) Realisieren Sie die Van der Polsche Differentialgleichung in Form einer MATLAB-Funktion:

```
function dzdt = vdp(t, z)
% function dzdt = vdp(t, z)
%
% Van der Polsche DGL
%
```

- c) Berechnen Sie numerische Näherungswerte für den Grenzyklus mit dem Euler-Verfahren zur Schrittweite $h = 0.1$ und visualisieren Sie die Näherungslösung in der Phasenebene.

Abbildungsverzeichnis

II.1. Fixpunktiteration für eine schwach steigende Funktion ($0 < g'(x) < 1$).	27
II.2. Fixpunktiteration für eine stark steigende Funktion ($1 < g'(x)$).	27
II.3. Fixpunktiteration für eine schwach fallende Funktion ($-1 < g'(x) < 0$).	28
II.4. Fixpunktiteration für eine stark fallende Funktion ($g'(x) < -1$).	28
II.5. Newton-Fraktal zur Gleichung $z^3 + 1 = 0$	41