Skriptum Numerik

Prof. Dr. René Grothmann

2015

Inhaltsverzeichnis

Vorwort

Dieses Skript ist eine Einführung in die numerische Mathematik und umfasst eine Standardvorlesung über "Numerik", wie sie im Bachelor-Studiengang Mathematik üblich ist. Es entstand im Wintersemester 2011/2012 an der Katholischen Universität Eichstätt.

Neben den notwendigen Grundlagen über Rechnerarithmetik, sowie den Sätzen über verschiedene Themen und Algorithmen der numerischen Mathematik, enthält das Skript zahlreiche Beispiele numerischer Rechnungen, die mit der Software "Euler Math Toolbox" (EMT) gerechnet wurden. Das erste Kapitel bringt eine Einführung in dieses Programm.

Selbstverständlich kann auch eine andere Software verwendet werden. Es bietet sich Matlab an, sofern es verfügbar ist, oder eine Kombination aus einem numerischen Klon von Matlab, wie etwa Scilab, und einer freien Computer-Algebra-Software wie Maxima. Der Autor ist der Meinung, dass eine praktische Betätigung mit dem Rechner unbedingt zu einem Modul "Numerik" gehört.

Neben interaktiven Systemen wäre auch eine Programmierung in einer Programmiersprache möglich. Jedoch lenken die Schwierigkeiten einer Programmiersprache möglicherweise zu sehr vom mathematischen Inhalt ab, und sollte auf einzelne Beispiele reduziert werden. Das Skript enthält deswegen auch Beispiele in Java.

Die Darstellung der Sätze und ihrer Beweise ist vollständig. Gelegentlich jedoch werden elementare Beweisschritte als Übung präsentiert. Das Skript enthält außerdem zahlreiche andere, vertiefende Übungsaufgaben.

Viel Erfolg

R. Grothmann
Eichstätt, 28. Mai 2015

Kapitel 1

Euler Math Toolbox

1.1 Numerisches und Symbolisches Rechnen

Euler Math Toolbox (im Folgenden kurz EMT genannt) ist eine Software, die numerische und symbolische Berechnungen mit einer interaktiven Umgebung und sehr guter Grafikausgabe ermöglicht. Sie besteht aus einem numerischen Kern und verwendet für die unendliche Ganzzahl-Arithmetik und für die symbolischen Berechnungen das Computer-Algebra-System Maxima, welches im Hintergrund geladen wird. Die Software läuft unter Windows, oder in Linux emuliert mit Wine. EMT ist unter der Webadresse `www.euler-math-toolbox.de` verfügbar.

In diesem einleitenden Kapitel soll die Software kurz vorgestellt werden, ohne auf die Benutzeroberfläche im Detail einzugehen. Es wird im Folgenden nur ein kurzer Überblick gegeben. Die Oberfläche von EMT besteht

- aus einem Textfenster, in dem Kommandos, Ausgaben, Kommentare und Grafiken in sogenannten "Notebooks" gespeichert werden,
- aus einem Grafikfenster, das die aktuelle Grafik enthält, und das mit der Tabulator-Taste nach vorne geholt werden kann, sofern es verdeckt ist. Euler kann so eingestellt werden, dass das Grafikfenster fehlt. Dann wird die Graphik im Textfenster angezeigt, wenn die Tabulatortaste gedrückt wird, oder wenn es für interaktive Eingaben notwendig ist. Graphiken können auch in den Text eingebunden werden.

Weitere Hilfe über die Oberfläche und die Kommandos von EMT und Maxima erhält man an vielen Stellen in Programm.

- EMT enthält eine vollständige Referenz für alle Kommandos, auch die von Maxima. Man kann diese Referenz im Browser aufrufen, oder nach einzelnen Kommandos im Hilfefenster suchen.
- Darüber hinaus gibt es eine Einführung in Deutsch und viele Einführungsnotebooks in Englisch, sowie viele Beispiel-Notebooks zum Erlernen der Syntax und aller Möglichkeiten.

- Die Statuszeile zeigt alle möglichen Vervollständigungen an, während ein Befehl eingetippt wird. Zum Einfügen verwendet man die Einfg-Taste. Nach dem Tippen der geöffneten Klammer zeigt die Statuszeile eine Parameterübersicht an.
- Ein Doppelklick auf einen Befehl öffnet das Hilfefenster an dieser Stelle.
- Ein Menü fügt Befehle mit Platzhaltern für Parameter ein. Der Pfeil nach rechts springt zum nächsten Platzhalter. Eine Eingabe ersetzt den Platzhalter.

Notebooks sollten reichhaltig Kommentare enthalten, um nach dem Abspeichern auch für andere Nutzer verständlich zu sein. Es gibt folgende Möglichkeiten, Kommentare einzufügen

- Zeichen nach // in einer Zeile werden ignoriert.
- Durch Druck auf F5 öffnet sich ein Kommentarfenster, in dem man einen Kommentar eingeben kann. Der Kommentar erscheint unmittelbar über der aktuellen Zeile. In solche Kommentare können mit Latex gesetzte Formeln und Bilder eingefügt werden.

Notebooks können geladen und gespeichert werden. Idealerweise verwendet man das Verzeichnis `Euler Files`, das in den eigenen Dokumenten bei der Installation angelegt wird. Beim Laden eines Notebooks wird EMT zurückgesetzt. Alle Variablen und die vom Nutzer definierten Funktionen gehen verloren. Maxima wird neu gestartet.

Prinzipiell werden numerische Kommandos in einer Kommandozeile eingegeben, und durch die Eingabetaste wird die numerische Auswertung von EMT gestartet. Mehrere Kommandos können durch Komma oder Semikolon getrennt in einer Zeile stehen. Variablen werden mit := oder mit = deklariert.

Hier einige Beispiele.

```
>10*exp(-5)
 0.0673794699909
>1/3  // Kommando mit Kommentar
 0.333333333333
>longest 1/3  // längeres Ausgabeformat
      0.3333333333333333
>a := 3; short 1/a  // kürzeres Ausgabeformat und Variable
 0.333333
>a*exp(-a^2), a*exp(-a),  // zwei Kommandos mit Ausgabe
 0.00037022941226
 0.149361205104
>sin(45°)  // trigonometrische Befehle im Radians-Modus!
 0.707106781187
>sin(0.5); arcsin(%)  // Verwendung des vorigen Ergebnisses
 0.5
```

Das letzte Beispiel zeigt, wie das vorige Ergebnis im nächsten Kommando verwendet werden kann, ohne auf eine Variable gespeichert zu werden. Man sollte % nur innerhalb einer Kommandozeile verwenden.

Kommandos können sich über mehrere Zeilen erstrecken, die mit ... enden. In diesem Fall werden die Zeilen durch einen einzigen Druck auf die Eingabetaste in einer beliebigen Zeile

ausgeführt. Außerdem können alle Zeilen im internen Editor bearbeitet werden, wenn man F9 in der ersten Zeile drückt.

Um solche Zeilen einfach zu erzeugen, drückt man Strg-Return am Ende der Zeile, oder innerhalb einer Zeile, wenn man die Zeile aufspalten will.

```
>g := 9.81; ...
 t := 22; ...
 1/2*g*t^2  // alle Kommandos werden auf einmal ausgeführt
 2374.02
```

Die Kommandos werden ansonsten in der Reihenfolge der Eingabe ausgeführt. Soll ein Notebook ganz neu berechnet werden, so kann man die Eingabe in die erste Zeile positionieren und mit Str-R alle Kommandos ausführen. Shift-Eingabetaste führt die Befehle des aktuellen Abschnitts erneut aus. Ein Abschnitt ist durch eine Überschrift im Kommentar oder durch eine leere Kommandozeile gekennzeichnet.

EMT kann Einheiten umrechnen, aber nicht speichern. Es wird angenommen, dass die gespeicherten Variablen die internationalen Standardeinheiten haben. Zur einfachen Umrechnung existiert eine spezielle Syntax.

```
>23*PS$  // simple Verwendung von PS
 16916.47125
>23PS  // vereinfachte Syntax für Einheiten
 16916.47125
>1e14/ly$  // simple Umrechnung von Metern in Lichtjahre
 0.0105700083402
>23PS -> kW  // spezielle Syntax
 16.91647125
>100in^2 -> " cm^2"  // zur Ausgabe
 645.16 cm^2
>1200h -> " d"  // Stunden in Tage
 50 d
>1d -> " min"
 1440 min
```

Die bevorzugte Art symbolisch zu rechnen verwendet symbolische Ausdrücke, die mit & eingeleitet werden. Damit kann man etwa eine "unendliche" Ganzzahl-Arithmetik verwenden oder in höherer Genauigkeit rechnen.

```
>&30!/10!  // Langzahlarithmetik

                      730965773291972714496000000

>&bfloat(1/347)

                 2.8818443804034582132564841498559b-3
```

Symbolische Ausdrücke werden von Maxima ausgewertet und ausgegeben, falls das Kommando nicht mit einem Semikolon endet. Für EMT stellt das Ergebnis lediglich eine spezielle Zeichenkette dar.

```
>&expand((1+x)^3) // symbolischer Ausdruck, ausmultipliziert

                          3      2
                         x  + 3 x  + 3 x + 1

>&factor(x^2+2*x+1)  // Faktorisieren

                                    2
                               (x + 1)
```

Symbolische Variablen werden mit &= deklariert. Sie werden bei der Deklaration von Maxima als Gleichung ausgegeben, sofern das Kommando nicht mit einem Semikolon abgeschlossen wurde.

```
>abl &= diff(x^3+sin(x),x)  // Ableiten, Ergebnis in Variable

                                     2
                            cos(x) + 3 x

>&integrate(abl,x)  // Integrieren der symbolischen Variable

                                    3
                             sin(x) + x

>&integrate(x^8*exp(x),x,-1,1)  // bestimmtes Integral

                                          - 1
                       14833 E - 109601 E
```

Es ist aber auch möglich, Maxima auf andere Art zu verwenden. Mit `maximamode` werden sogar alle Kommandos an Maxima gesendet. Zu unterscheiden ist der direkte Modus, in dem die Maxima-Syntax unverändert gilt, und der Kompatibitäts-Modus, in dem die Syntax an EMT angepasst wurde.

Der direkte Modus ist anfangs nützlich, um sich in Maxima-Dokumentationen zurecht zu finden. Für symbolische Ausdrücke sind die Unterschiede nicht relevant.

```
>maximamode
 Maxima mode is on (compatibility mode)
>a:=3; a^2  // Syntax genau wie in Euler

                                  9

>diff(x^2,x)  // symbolisches Rechnen

                                 2 x

>euler 1/3 // Euler-Kommando in diese Modus
 0.333333333333
>maximamode off
 Maxima mode is off
>maximamode direct
 Maxima mode is on (direct mode)
>a:3$ a^2;  // Originalsyntax von Maxima

                                  9
```

```
>maximamode off
 Maxima mode is off
```

Als dritte Möglichkeit kann eine komplette Zeile in Maxima ausgewertet werden, ebenfalls in beiden Modi. Die Zeile muss dann mit :: bzw. ::: beginnen, einschließlich einem Leerzeichen.

```
>:: diff(x^x,x); integrate(%,x)  // kompatibler Modus

                                  x log(x)
                                 E

>::: g:diff(x^x,x)$ integrate(g,x);  // direkter Modus

                                  x log(x)
                                 E
```

1.2 Numerische Algorithmen

In EMT sind sehr viele von den numerische Algorithmen von Hause aus implementiert, die in diesem Skriptum definiert, erläutert und bewiesen werden. Daher eignet sich der numerische Teil von EMT besonders für eine Einführung in die numerische Mathematik. Die meisten Probleme der Praxis sind im Unterschied zu Schulaufgaben nicht exakt zu lösen, so dass numerische Rechnungen notwendig werden. Oft ist aber zur Unterstützung bei der Herleitung der Algorithmen symbolisches Rechnen hilfreich. Wir werden das in diesem Skriptum deutlich sehen.

Beispiele für numerische Algorithmen sind das Lösen von Gleichungen oder Gleichungssysteme, das numerische Integrieren und Differenzieren, oder Approximation und Interpolation von Funktionen.

Die in EMT implementierten Algorithmen lassen sich auf Funktionen anwenden, die entweder als Funktionen mit ihrem Namen oder als Ausdrücke in einer Zeichenkette eingegeben werden. In diesem Fall gilt die Konvention, dass die Hauptvariable im Ausdruck x ist.

Wir beginnen mit der Demonstration von Ausdrücken. Ausdrücke können auf Variablen in gewohnter Weise gespeichert werden. Sie sind für EMT Zeichenketten. Sie können aber wie Funktionen ausgewertet werden.

```
>expr := "x^x";  // numerischer Ausdruck in Variable gespeichert
>expr(1.5)  // numerische Auswertung in x=1.5
 1.83711730709
>integrate(expr,1,2)  // numerisches Integral
 2.05044623453
>solve(expr,1,y=2)  // Löse x^x=2 numerisch, Schätzwert ist 1
 1.55961046946
>"a^2+a*b+b^2"(a=1,b=2)  // Verwendung anderer Variablen als x
 7
>solve("x^2-2",1)  // Löse x^2-2=0 numerisch
 1.41421356237
```

Man beachte, dass x^x sich nicht exakt integrieren lässt. Maxima scheitert am Integral.

```
>&integrate(x^x,x)

                                  /
                                  [  x
                                  I x  dx
                                  ]
                                  /
```

Natürlich hat auch Maxima numerische Algorithmen implementiert. Für große Probleme ist jedoch zu beachten, dass EMT um Größenordnungen schneller ist. Auch hat EMT viel mehr numerische Algorithmen implementiert.

Numerisches Differenzieren ist, wie wir noch sehen werden, nicht mit voller Genauigkeit durchzuführen. Hier ist Maxima überlegen. Wir demonstrieren das an der Funktion x^x. Das Ergebnis von Maxima ist exakter. Wir zeigen nebenbei, wie man in einem symbolischen Ausdruck mit `with` einen Wert einsetzt.

```
>&diff(x^x,x) with x=1.5  // numerische Rechnung in Maxima

                           2.582004274612949

>longestformat; diff("x^x",1.5)  // numerisches Differenzieren in Euler
       2.582004274613638
```

1.3 Einfache Funktionen

Die numerischen Algorithmen von EMT akzeptieren auch Funktionen. Wir können einfache Funktionen numerisch mit dem Schlüsselwort `function` und ":=" und symbolisch mit "&=" definieren. Der Ausdruck bei der Definition einer symbolischen Funktion wird zunächst mit Maxima ausgewertet, und erst dann wird die Funktion definiert.

Im Beispiel definieren wir die Funktion und ihre Ableitung symbolisch. Symbolische Funktionen stehen auch in numerischen Ausdrücken zur Verfügung. Das Newton-Verfahren benötigt zum Beispiel die Funktion und ihre Ableitung.

```
>function f(x) &= x^x  // symbolische Funktion

                                    x
                                   x

>&f(t^2+1)  // Verwendung von f in symbolischen Ausdrücken

                                      2
                               2     t  + 1
                             (t  + 1)

>f(1.5)  // Verwendung von f in numerischen Ausdrücken
 1.83711730709
```

```
>function df(x) &= diff(f(x),x)  // symbolische Ableitungsfunktion

                            x
                           x  (log(x) + 1)

>newton("f(x)","df(x)",1,y=2)  // Löse x^x=2 mit Newton
 1.55961046946
```

Die Funktion im folgenden Beispiel ist nur numerisch und implementiert die Umkehrfunktion von x^x. Wir testen die Lösung von $x^x = 5$.

```
>function f(a) := solve("x^x",1,y=a)
>f(5), %^%  // Lösung von x^x=5
 2.12937248276
 5
```

Numerische Algorithmen akzeptieren den Namen von Funktionen in Anführungszeichen, symbolische Ausdrücke oder auch nur den Namen einer symbolischen Funktion.

```
>function f(x) &= x^(a*x)

                                  a x
                                 x

>a:=1.2; solve("f",1,y=2)  // Name der Funktion statt Ausdruck
 1.47813165187
>a:= 1.4; solve(f,1,y=2)  // geht nur bei symbolischen Funktionen
 1.41790533445
>solve(&x^x,1,y=2)  // symbolischer Ausdruck direkt
 1.55961046946
```

Wichtig zu wissen ist, dass symbolische und numerische Funktionen, die in einer Zeile definiert werden, globale Variablen verwenden können.

1.4 Die Matrixsprache

Die Matrixsprache von EMT ist ähnlich zu der Matrixsprache von Matlab, jedoch konsequenter umgesetzt und nicht kompatibel.

Wesentliches Merkmal einer Matrixsprache ist, dass alle Funktionen und Operatoren elementweise auf Matrizen wirken. Eine Funktion wird auf alle Elemente der Matrix oder des Vektors separat angewendet, ein Operator verknüpft die Elemente von zwei Matrizen oder Vektoren paarweise.

Wir erzeugen im folgenden Beispiel einen einfachen Vektor mit den Zahlen 1 bis 10, und verknüpfen diesen Vektor mit Funktionen und Operatoren.

```
>shortformat; v:=1:10
 [ 1  2  3  4  5  6  7  8  9  10 ]
>sqrt(v)
 [ 1  1.41421  1.73205  2  2.23607  2.44949  2.64575  2.82843  3  3.16228 ]
>v^2
 [ 1  4  9  16  25  36  49  64  81  100 ]
>1+v
 [ 2  3  4  5  6  7  8  9  10  11 ]
>3*v
 [ 3  6  9  12  15  18  21  24  27  30 ]
```

Die letzten beiden Kommandos zeigen, dass ein Skalar mit einem Vektor ebenfalls elementweise verknüpft wird.

Matrizen werden zeilenweise eingegeben, wobei die Zeilen durch Semikolon getrennt werden, die Elemente einer Zeile durch Kommas. Unvollständige Zeilen werden durch 0 ergänzt. Die Matrixsprache bedeutet, dass auch Matrizen elementweise verknüpft werden.

```
>A=[1,2,3;4,5,6;7]
             1             2             3
             4             5             6
             7             0             0
>A^2
             1             4             9
            16            25            36
            49             0             0
>exp(A)
       2.71828       7.38906       20.0855
       54.5982       148.413       403.429
       1096.63             1             1
```

Diese elementweise Multiplikation ist verschieden vom Matrixprodukt.

```
>A*A
             1             4             9
            16            25            36
            49             0             0
>A.A  // Matrixprodukt
            30            12            15
            66            33            42
             7            14            21
```

Falls ein Spaltenvektor mit einer Matrix verknüpft wird, wirken seine Elemente auf die Zeilen der Matrix. Ein Zeilenvektor wird auf die Spalten. Ein Zeilenvektor verknüpft mit einem Spaltenvektor ergibt eine Matrix.

```
>x := 0:0.2:1  // Zeilenvektor
 [ 0  0.2  0.4  0.6  0.8  1 ]
>n := (0:3)'  // Spaltenvektor
         0
         1
         2
         3
>shortest x^n  // Matrix x[i]^j
         1         1         1         1         1         1
         0       0.2       0.4       0.6       0.8         1
         0      0.04      0.16      0.36      0.64         1
         0     0.008     0.064     0.216     0.512         1
```

Die Matrixsprache funktioniert auch für Bedingungen, die in EMT die Werte `0=false` oder `1=true` zurück geben. Der Vergleich wird wie ein Operator elementweise durchgeführt. Mit `nonzeros` lassen sich die Indizes aller Elemente eines Vektors, die nicht gleich 0 sind herausfinden.

```
>(1:10)<=5
 [ 1  1  1  1  1  0  0  0  0  0 ]
>nonzeros((1:10)>5)
 [ 6  7  8  9  10 ]
>n := 2|(3:2:100); n[nonzeros(isprime(n))]
 [ 2  3  5  7  11  13  17  19  23  29  31  37  41  43  47  53  59  61  67
 71  73  79  83  89  97 ]
>t := (1.01)^(1:100);
>min(nonzeros(t>2))  // minimaler Index mit 1.01^n > 2
 70
>ceil(log(2)/log(1.01))  // dasselbe
 70
```

Weitere boolsche Operatoren sind && für "und" und || für "oder", sowie ! für "nicht". Diese Operatoren wirken ebenfalls elementweise und interpretieren nur 0 als falsch.

```
>v=1:10; v>=4 && v<=6, nonzeros(%)
 [0,  0,  0,  1,  1,  1,  0,  0,  0,  0]
 [4,  5,  6]
```

Es gibt viele weitere Funktionen, die Matrizen erzeugen.

```
>shortestformat;
>random(1,20)  // 20 in [0,1] gleichverteilte Zufallszahlen
 [ 0.636  0.151  0.252  0.334  0.761  0.41  0.559  0.711  0.0843  0.32
 0.353  0.433  0.425  0.0221  0.839  0.429  0.136  0.21  0.793  0.637 ]
>intrandom(1,20,6)  // 20 Würfelzahlen
 [ 2  4  3  3  5  5  5  3  5  4  1  1  5  5  3  2  3  2  1  6 ]
>shuffle(1:10)  // Zufalls-Permumation
 [ 2  9  10  3  4  8  7  5  6  1 ]
>sort(random(1,10))  // Sortieren
 [ 0.0373  0.0665  0.124  0.221  0.385  0.45  0.487  0.522  0.679  0.703 ]
>id(4)
         1         0         0         0
         0         1         0         0
         0         0         1         0
         0         0         0         1
>ones(2,3)
         1         1         1
         1         1         1
```

Weiter gibt es verschiedene Operatoren, die Matrizen verknüpfen.

```
>v := 1:3; A := dup(v,3)
         1         2         3
         1         2         3
         1         2         3
>A_0
         1         2         3
         1         2         3
         1         2         3
         0         0         0
```

```
>A|A
        1         2         3         1         2         3
        1         2         3         1         2         3
        1         2         3         1         2         3
```

Der Zugriff auf die Matrixelemente erfolgt durch Indizierung. Man kann ein oder zwei Indizes Verwenden. Verwendet man nur einen Index bei echten Matrizen, so wird die Zeile zurück gegeben.

```
>shortestformat; A := [1,2,3;4,5,6;7,8,9]
        1         2         3
        4         5         6
        7         8         9
>A[1]
 [ 1  2  3 ]
>A[2,2]
 5
>A[:,3]  // alle Zeilen, dritte Spalten
        3
        6
        9
>A[[3,2,1],3]  // 3., 2. und 1. Zeile, 3. Spalte
        9
        6
        3
>A[1:2,1:2] := 0
        0         0         3
        0         0         6
        7         8         9
```

Wie man sieht, ist es auch möglich, Vektoren von Indizes zu verwenden. Auch kann eine ganze Untermatrix gleich einer passenden Matrix, oder gleich einem Skalar gesetzt werden.

Für die lineare Algebra ist es wichtig zu wissen, dass das Matrixprodukt mit dem Punkt gerechnet wird. Die transponierte Matrix wird durch ein Hochkomma erzeugt. Ein Gleichungssystem wird mit \ gelöst.

```
>A := [1,2,3;3,2,1;4,5,4]
             1             2             3
             3             2             1
             4             5             4
>b := sum(A)
             6
             6
            13
>x := A\b
             1
             1
             1
>A.x
             6
             6
            13
```

Es gibt verschiedene numerische Funktionen für Berechnungen in der linearen Algebra, einschließlich numerischer Berechnung der Eigenwerte oder der singulären Werte.

```
>shortformat;
>A := [1,2,3;4,5,6;7,8,9];
>det(A)  // numerische Determinante
 0
>b := [1,1,1]';
>x := fit(A,b)  // minimiert ||Ax-b||
            -1
             1
             0
>norm(A.x-b)  // tatsächlich ist das lösbar
 0
>kernel(A)  // numerischer Kern von A
             1
            -2
             1
>sort(real(eigenvalues(A)))  // numerische Eigenwerte
 [ -1.11684  0  16.1168 ]
>fracprint(svdsolve(A,[3,4,1]'))  // Fit mit Singulärwertzerlegung
    -41/18
      -1/9
     37/18
```

Aufgrund der Zeilensyntax von EMT ist es auch möglich den Gauß-Algorithmus per Hand durchzurechnen. Dabei ist es nützlich, die vorige Zeile mit Strg-Pfeil-Hoch auf die aktuelle Zeile zu kopieren.

```
>shortformat;
>A := [5,2;3,4]; b := [1,1]';
>M := A|b
             5             2             1
             3             4             1
>M[1] := M[1]/5; fracprint(M)
         1       2/5       1/5
         3         4         1
>M[2] := M[2]-3*M[1]; fracprint(M)
         1       2/5       1/5
         0      14/5       2/5
>M[2] := M[2]/M[2,2]; fracprint(M)
         1       2/5       1/5
         0         1       1/7
>M[1] := M[1]-M[1,2]*M[2]; fracprint(M)
         1         0       1/7
         0         1       1/7
```

Es gibt auch symbolische Matrizen.

```
>A &= [1,a;a,2]

                                [ 1  a ]
                                [      ]
                                [ a  2 ]

>&determinant(A)

                                      2
                                 2 - a

>&eigenvalues(A)
```

```
                2              2
  3 - sqrt(4 a  + 1)  sqrt(4 a  + 1) + 3
[[------------------, ------------------], [1, 1]]
          2                   2
```

Matrixberechnungen wie beim Gauß-Algorithmus sollten allerdings in einer Maxima-Zeile erfolgen.

```
>:: A[1] := A[1]-A[2]*a; A

                          [        2        ]
                          [ 1 - 2 a   - 3 a ]
                          [                 ]
                          [    a        2   ]
```

1.5 Plots

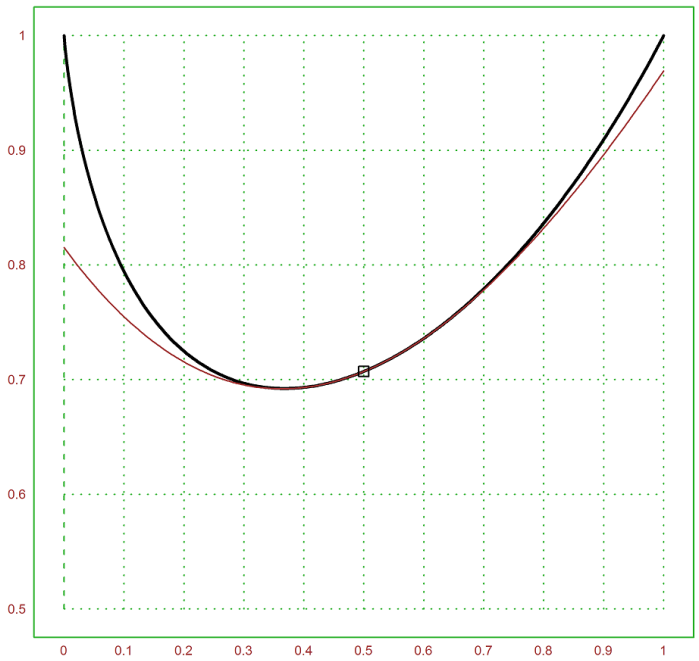

Abbildung 1.1: Taylorpolynom vom Grad 3 zu x^x

Plots von Funktionen und Daten sind flächig oder räumlich möglich. Die entsprechenden Funktionen sind `plot2d` bzw. `plot3d`. Diese Funktionen sind sehr vielfältig und akzeptieren plotbare Funktionen, Ausdrücke oder Datenvektoren. Darüber hinaus gibt es implizite Plots von Ausdrücken mit zwei oder drei Variablen. Räumliche Plots lassen sich als Rot-Grün-Anaglyphen stereographisch anzeigen.

Plots werden im Grafikfenster angezeigt. Falls das Fenster verdeckt ist oder Euler so eingestellt ist, dass Grafiken im Textfenster erscheinen, kann die Grafik mit der Tabulator-Taste nach vorne geholt werden. Per Default ist die Grafik exakt quadratisch.

Grafiken können mit dem Kommando `insimg;` in das Textfenster eingefügt werden, oder einfach, indem man die Kommandozeile mit einem Doppelpunkt abschließt. Sie werden dann mit dem Notebook gespeichert. Per Default wird ein Unterverzeichnis `images` für die Bilder verwendet.

```
>plot2d("sin(x)",0,2pi):  // Grafik erscheint unter dieser Zeile
```

Im folgenden Beispiel zeichnen wir eine Funktion und ihre Taylorreihe in denselben Plot (siehe die obige Abbildung). Wir verwenden einmal einen Ausdruck, und das andere Mal einen symbolischen Ausdruck in der Variable x. Die Funktion `plot2d` akzeptiert Ausdrücke, Funktionen und symbolische Ausdrücke genau wie die numerischen Algorithmen von EMT.

Die Grenzen des Plotbereichs werden mit `a=...` etc. gesetzt. Alternativ kann man auch mit `r=...` einen quadratischen Bereich um 0 wählen. Der Parameter `add=true` (oder kurz `>add`) sorgt dafür, dass der alte Plot nicht gelöscht wird.

Alle Kommandos wurden in einem Mehrzeilenkommando zusammengefasst. Bei der Definition der Funktion ist dies nur möglich, wenn sie mit einem Semikolon abgeschlossen wird.

```
>function f(x) &= x^x; ...
 plot2d("f(x)",a=0,b=1,c=0.5,d=1,color=black,thickness=2); ...
 plot2d(&taylor(f(x),x,1/2,3),>add,color=red); ...
 plot2d(0.5,f(0.5),>points,>add):
```

Das letzte Kommando fügt schließlich noch den Entwicklungspunkt hinzu. Es ist ein Beispiel für einen Plot von Daten. Wenn `plot2d` Datenvektoren statt Funktionen als Parameter erhält, so werden die Daten als x- und y-Koordinaten von Punkten interpretiert.

```
>x := 0:10;
>y := bin(10,x);  // bin(10,0) bis bin(10,10)
>plot2d(x,y);  // Plot als Streckenzug
>plot2d(x,y,>points,style="o",>add);  // Plot als Punkte
```

Verwendet man einen x-Vektor und eine Matrix für y, so wird jede Zeile der Matrix als Funktion aufgefasst. Wenn x auch eine Matrix ist, so werden für jede Zeile der beiden Matrizen eine Funktion gezeichnet. Damit lassen sich sehr elegant mehrere parameterabhängige Funktionen oder Kurven in einen Plot zeichnen.

```
>r := linspace(0,1,10);  // Radius r von 0 bis 1 (Zeile)
>phi := linspace(0,2pi,40)';  // Winkel phi von 0 bis 2pi (Spalte)
>X := r*cos(phi); Y := r*sin(phi);  // X und Y von den Kreisen
>plot2d(X,Y,r=1.2);  // plotte radiale Strecken
>plot2d(X',Y',>add);  // plotte Kreise
```

Die Gitterdarstellung kann mit dem Parameter `grid` auf verschiedene Weisen verändert werden. Wir müssen dazu auf die Referenz zu `plot2d` verweisen.

Es gibt eine ganze Reihe von weiteren Möglichkeiten, Funktionen und Daten zu plotten.

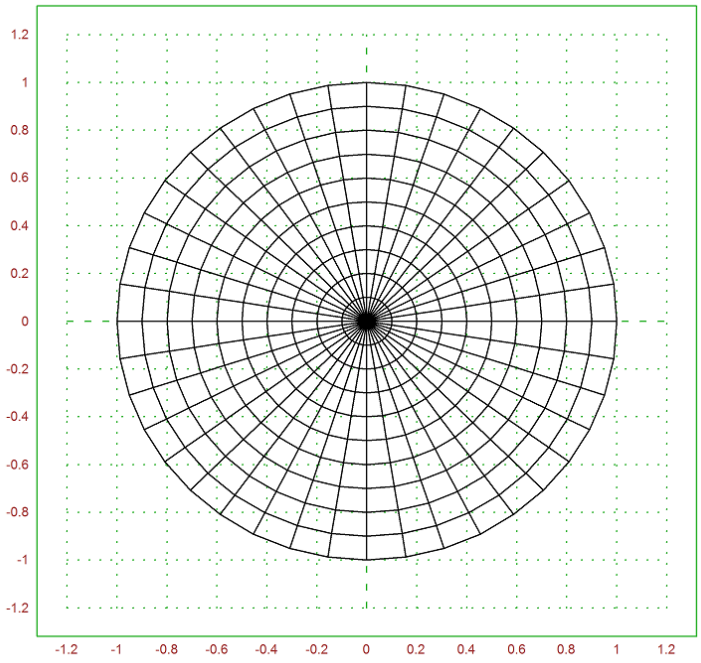

Abbildung 1.2: $r\,(\cos(\phi), \sin(\phi))$

- Balkenplots lassen sich mit `>bar` in verschiedenen Stilen ausgeben. Dabei sollte der x-Vektor um eins länger sein, da er die Begrenzungen der Intervalle angibt.
- Implizite Plots können mit `level=v` von Funktionen oder Ausdrücken in den zwei Variablen x und y gezeichnet werden. Dabei ist v ein Skalar und gibt den Wert an, für den $f(x, y) = v$ gelöst werden soll. Wenn v ein Vektor ist, werden mehrere implizite Kurven gezeichnet.
- Für Polygonkurven kann man einen Vektor x und einen Vektor y verwenden. Alternativ kann man für die beiden Koordinaten Funktionen oder Ausdrücke in x angeben. Dabei wird mit `xmin` und `xmax` der Laufbereich der Variablen angegeben. Geschlossene Polygone lassen sich mit `>filled` in verschiedenen Stilen plotten.

Es gibt auch eine Vielzahl von elementaren Möglichkeiten, ohne die Funktion `plot2d` den Plot zu erzeugen. Auch dazu müssen wir auf die Referenz verweisen. Die Grundidee ist, mit `hold on` die Grafik vor dem löschen zu bewahren, dann die einzelnen Plotelemente übereinander zu legen, und schließlich mit `hold off` den Plot wieder freizugeben.

Beachten Sie, dass die Größe des Plotfensters veränderbar ist. Der Plot wird dabei neu gezeichnet. Es gibt einen internen Puffer für die Grafik. Das Fenster kann mit einem Menüeintrag für skalierte Plots, die in beiden Richtungen die gleiche Skala verwenden sollen, eingerichtet werden.

Plots können exportiert werden.

- Für Druckausgaben eignet sich eine Ausgabe als PNG-Export mit 100 Pixeln auf 1 cm. Für Latex verwendet dieses Skript 1000 Pixel, wobei die Grafiken eine Breite von 10 cm haben. Für Webseiten stellt man die exakt gewünschte Größe ein. Ein Nachteil dieser Vorgehensweise ist immer, dass die Beschriftungen und Labels zu klein geraten können, insbesondere auf Webseiten mit sehr kleinen Grafiken. Eventuell ist es nötig, das Kommando `setfont` zu verwenden. Wir verweisen dazu auf die Beispiel-Notebooks und die Referen.
- Die Ausgabe von SVG-Grafiken ist auch möglich, und ergibt gut aussehende Grafiken, wenn der verwendete SVG-Renderer gut arbeitet. Der Vorteil ist die Skalierbarkeit. Eventuell ist auch hier `setfont` nötig.
- Der Export über die Zwischenablage ist als Windows Metafile oder als Bitmap-Grafik möglich. Ein direkter Druck auf einem Windows-Drucker ist ebenfalls möglich, jedoch nicht empfehlenswert.

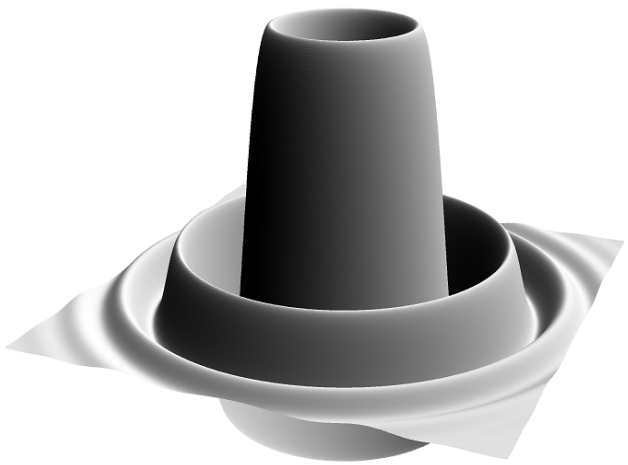

Abbildung 1.3: $\sin(4r)\exp(-r^2)$

EMT erzeugt auch gut aussehende 3D-Plots. Die Funktion `plot3d` akzeptiert dazu Funktionen und Ausdrücke in zwei Variablen x und y. Das letzte Beispiel ist in der Abbildung zu sehen.

```
>plot3d("x^2+y^2*x",r=2);  // simpler Plot einer Funktion
>plot3d("x*y",>contour);  // mit Höhenlinien
>plot3d("sin(4*(x^2+y^2))*exp(-x^2-y^2)",  ...
>  r=2,n=500,>hue,frame=false);  // ohne Rahmen, mit Beleuchtung
```

Der Plot lässt sich vielseitig gestalten. Für die Details verweisen wir auf die Referenz von `plot3d`. Eine interessante Möglichkeit sind Anaglyphen, die mit Rot/Cyan-Brille dreidimensional erscheinen. Der Effekt ist beeindruckend.

```
>plot3d("x*y",>polar,>wire,>anaglyph);  // 3D-Drahtgitter
```

Plots von Oberflächen werden mit drei Parameterfunktionen oder mit drei Matrizen erzeugt. Im folgenden Beispiel erzeugen wir die Koordinaten für eine Fläche, die ensteht, wenn man eine Funktion um die z-Achse dreht (siehe die folgende Abbildung).

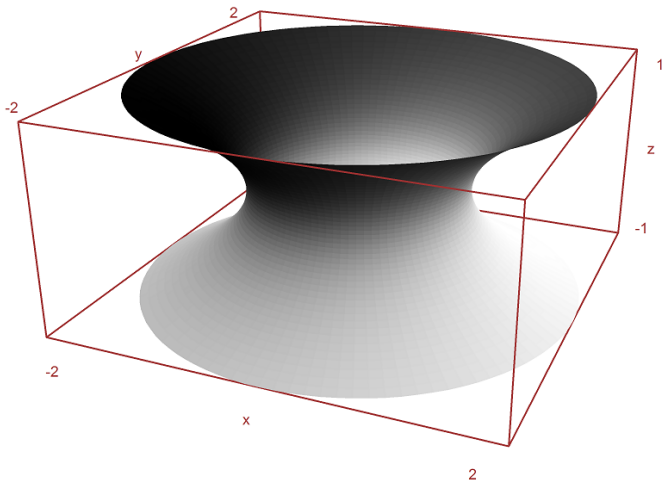

Abbildung 1.4: $X = f(z)\cos(\phi)$, $Y = f(z)\cos(\phi)$, $Z = z$

```
>function f(z) := z^2+1  // zu drehende Funktion
>phi := linspace(0,2pi,100);  // Winkel als Zeilenvektor
>Z := linspace(-1,1,50)';  // z-Werte als Spaltenvektor
>plot3d(f(Z)*cos(phi),f(Z)*sin(phi),Z,>hue);
```

Man beachte dabei die Verwendung der Matrixsprache von EMT. Der Spaltenvektor Z wird mit den Zeilenvektoren $\cos(\phi)$ und $\sin(\phi)$ verknüpft und ergibt eine Matrix aus (x, y, z)-Werten. Der Spaltenvektor Z wird von `plot3d` korrekt als Matrix verwendet.

Kombinierte Plots mit implizit berechneten Höhenlinien in einer Ebene unterhalb des Funktionsgraphen sind ebenfalls möglich. Der Parameter `>cp` (contour plane) erzeugt einen solchen Plot. Für die Details verweisen wir auf die Dokumentation und die Tutorien.

Die Funktionen `plot2d` und `plot3d` lassen auch Benutzeraktionen zu, wenn `>user` eingestellt wurde.

- Die dreidimensionalen Plots lassen sich dann mit den Maustasten drehen oder mit $+/-$ zoomen.
- Die Taste a erzeugt einen Anaglyphen-Plot von der aktuellen Ansicht.
- Wenn `hue` oder `contour` gewählt ist, so kann die Lichtquelle verschoben werden. Die Lichtquelle und das Umgebungslicht kann aber auch mit Parametern eingestellt werden. Dazu verweisen wir auf die Referenz zu `plot3d`.

- Die Leertaste stellt den Plot auf die ursprüngliche Ansicht zurück.
- Auch der Punkt, auf den die Ansicht gerichtet ist, kann interaktiv oder per Voreinstellung verschoben werden.

1.6 Exakte Arithmetik

EMT besitzt eine Intervall-Arithmetik. In diesem Skript werden verschiedene numerische Verfahren vorgestellt, die Intervalle nutzen. Das Ziel ist, garantierte enge Einschließung für Lösungen zu finden. Der Aufwand dafür ist natürlich größer als bei einfachen Iterationsverfahren. Zusammen mit dem exakten Skalarprodukt wird damit ein neue Art des "wissenschaftlichen Rechnens" möglich.

Die Grundidee der Intervallarithmetik ist, dass zwei Intervalle verknüpft werden, indem man den Operator auf alle Elemente beider Intervalle anwendet, und eine Einschließung aller Ergebnisse zurück liefert. Also mit dem Beispiel der Multiplikation von X und Y

$$X \cdot Y = \{x \cdot y : x \in X,\, y \in Y\} \subseteq Z$$

wobei das Rechenergebnis Z möglichst eng gefasst wird. Bei stetigen Operatoren ist $X \cdot Y$ in der Tat ein Intervall. Der Rechner liefert aber meist ein geringfügig größeres Intervall.

Für Funktionen gilt analog

$$f(X) = \{f(x) : x \in X\} \subseteq Z$$

mit möglichst engem Z.

Man beachte zum Beispiel

$$[0,1] = [-1,1]^2 \neq [-1,1] \cdot [-1,1] = [-1,1].$$

Dieser und andere Effekte sorgen dafür, dass das Ergebnis eines längeren Ausdrucks meist überschätzt wird. EMT bietet eine Unterteilung des Ausgangsinterval an, um ein möglichst enges Ergebnis zu erreichen.

```
>X := ~-1,1~  // Intervall in Euler
 ~-1,1~
>X^2
 ~0,1~
>X*X
 ~-1,1~
>X*exp(X)-X^2+X  // überschätzte Auswertung
 ~-4.7,3.7~
>ieval("x*exp(x)-x^2+x",X,30)  // bessere Auswertung
 ~-2.4,2.8~
```

Intervalle können auch in einer Notation mit ± eingegeben werden. Das Zeichen lässt sich mit Hilfe der Taste F8 eingeben. Für Beispiele dazu verweisen wir auf das Tutorial über Intervallrechnung.

Eine garantierte Einschließung der Lösung einer Gleichung ist schon mit einem intervallmäßig gerechneten Bisektionsalgortihmus möglich. Schneller ist das Intervall-Newton-Verfahren. Wir berechnen den Punkt, in dem $x^{\sin(x)}$ ein Minimum annimmt, indem wir die Nullstelle der Ableitung einschließen.

```
>function f(x) &= diff(x^sin(x),x)

                     sin(x)  sin(x)
                    x       (------ + cos(x) log(x))
                              x

>ibisect(f,0.1,1)
 ~0.3522153992831407,0.3522153992831472~
>inewton(f,&diff(f(x),x),0.36)
 ~0.35221539928314327,0.3522153992831441~
```

Es gibt auch Intervall-Matrizen. Damit lassen sich selbst schlecht konditionierte Gleichungssysteme sehr gut lösen. Es wird ein numerisches Iterationsverfahren verwendet, das wir in diesem Skript besprechen werden. Die Garantie der Einschließung kommt von einem Fixpunktsatz.

```
>H := hilbert(10); b := sum(H);
>ilgs(H,b)
  ~0.99999999999999978,1.0000000000000002~
  ~0.99999999999999956,1.0000000000000004~
  ~0.99999999999999822,1.0000000000000018~
  ~0.99999999999999334,1.0000000000000067~
   ~0.99999999999995237,1.000000000000048~
    ~0.9999999999996404,1.000000000000359~
      ~0.999999999999152,1.00000000000085~
    ~0.9999999999995126,1.000000000000486~
     ~0.9999999999997816,1.00000000000022~
  ~0.99999999999996947,1.0000000000000302~
```

1.7 Komplexe Zahlen

EMT besitzt eine Arithmetik für komplexe Zahlen. Die Zahlen werden mit der komplexen Einheit `I` oder in der Form $a + ib$ eingegeben. Der Logarithmus und die Wurzelfunktion liefert den Zweig in der mit der negativen x-Achse geschlitzten Ebene.

```
>(1+I)^2
 0+2i
>exp(pi*I)+1
 0+0i
>log(3+4i)
 1.60943791243+0.927295218002i
```

Komplexe Vektoren und Matrizen sind ebenfalls möglich. Im folgenden Beispiel zeichnen wir das Bild des Einheitskreises unter der Funktion $\exp(x)$. Die Funktion `plot2d` zeichnet die Punkte der Bildmatrix automatisch als Gitter. Mit `cgrid=n` lässt sich erreichen, dass nur ein Teil der Gitterlinien gezeichnet werden.

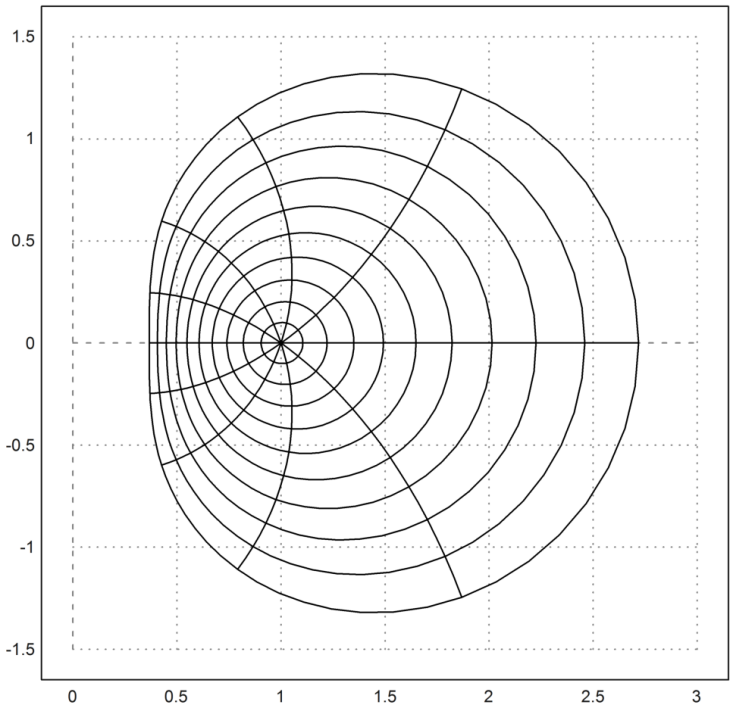

Abbildung 1.5: Bild des Einheitskreises unter exp

```
>r := linspace(0,1,20); phi := linspace(0,2pi,80)';
>z := r*exp(I*phi);
>plot2d(exp(z),a=0,b=3,c=-1.5,d=1.5,cgrid=10);
```

1.8 Programmierung

Die Programmiersprache von EMT ist ein sehr ausgereifter Basic-Dialekt. In der Tat sind große Teile von EMT in dieser Programmiersprache geschrieben. Maxima besitzt ebenfalls eine Programmiersprache, auf die wir hier nicht eingehen.

Wir haben bisher nur Einzeilen-Funktionen kennen gelernt. Um komplexere Funktionen einzugeben, verwendet man am einfachsten den Funktionseditor. Schreiben Sie dazu `function f(x)` in eine Zeile, und drücken Sie F9. Die Funktion lässt sich dort bearbeiten, und mit OK in EMT einfügen. Alternativ kann man auch Strg-Eingabe nach der Kopfzeile drücken und die Funktion direkt eingeben. Später kann man in eine Funktion klicken und die Funktion bearbeiten.

```
>function f(x) ...
$  if x>0 then return x^3;
$  else return x^2;
$  endif;
$endfunction
>[f(-2), f(0), f(2)]
 [ 4  0  8 ]
```

Jede Funktion beginnt mit `function` und endet mit `endfunction`. Der Funktioneneditor fügt `endfunction` automatisch ein. Die drei Punkte ... am Ende der ersten Zeile bedeuten, dass die Funktion mit einem einzigen Druck auf die Eingabezeile eingegeben werden kann. Auch sie werden vom Funktionseditor automatisch eingefügt.

Diese Funktion hat Zweige, je nachdem welches Vorzeichen x hat. Die Verzweigung wird mit `if ... endif` realisiert. Es kann außer dem `else` auch mehrere Zweige mit `elseif` geben. Das `then` ist nicht notwendig, aber empfehlenswert.

Die obige Version der Funktion funktioniert nicht für Vektoren und Matrizen. Um die Matrix-Sprache einzuhalten, genügt es `map` einzufügen.

```
>function map f(x) ...
$  if x>0 then return x^3;
$  else return x^2;
$  endif;
$endfunction
>f([-2,0,2])
 [ 4  0  8 ]
```

Bedingungen für `if` können mit `and`, `or` und `not` verknüpft werden. Es gilt, dass etwa bei `and` die zweite Bedingung nicht mehr ausgewertet wird, wenn schon die erste falsch ist.

Soll eine Bedingung für einen ganzen Vektor wahr sein, so kann man `all` verwenden. `any` prüft, ob die Bedingung für irgendein Element des Vektors erfüllt ist.

```
>any(isprime(31399:2:31467))
 0
>all(!isprime(31399:2:31467))
 1
```

Natürlich besitzt EMT auch Schleifen. Die unendliche Schleife verwendet `repeat ... end`, und kann mit `while`, `until`, oder `break`, und natürlich mit `return` abgebrochen werden.

```
>function cositer (x) ...
$  repeat
$    xn=cos(x);
$    if abs(xn-x)<1e-15 then return xn; endif;
$    x=xn;
$  end;
$endfunction
>cositer(1)
 0.739085133216
```

Alternativen sind `break` in einer `if`-Konstruktion, oder `until` und `while`, die überall in der Schleife, auch mehrfach, stehen können. Falls die Bedingung zutrifft bzw. nicht mehr zutrifft, wird die Schleife abgebrochen.

```
>function cositer (x) ...
$  repeat
$    xn=cos(x);
$    until abs(xn-x)<1e-15;
$    x=xn;
$  end;
$  return xn;
$endfunction
```

Natürlich gibt es auch einfache Schleifen über einen Zahlbereich, oder auch über die Elemente eines Vektors. Hier ein Beispiel für die Iteration über einen Vektor.

```
>function sumtest (v) ...
$  sum=0;
$  for x=v;
$  sum=sum+x;
$  end;
$  return sum
$endfunction
>sumtest(1:10)
 55
>sum(1:10)  // In EMT schon vorprogrammiert
 55
```

Eine andere Möglichkeit ist die `for`-Schleife mit einem Zahlbereich, die einen Index von einem Wert bis zu einem Wert iteriert. Die Schrittweite ist normalerweise 1, kann aber mit `step` eingestellt werden. Der Index ist eine Gleitkommazahl. Für ganzzahlige Loops verwendet man `loop ... end`. Details finden Sie in der Referenz.

Hier ein Beispiel, bei dem eine Matrix erzeugt per Iteration erzeugt wird.

```
>function cositer (x,n) ...
$  v=zeros(1,n); v[1]=x;
$  for i=2 to n;
$    x=cos(x); v[i]=x;
$  end;
$  return v;
$endfunction
>cositer(1,10)
 [ 1  0.540302305868  0.857553215846  0.654289790498  0.793480358743
 0.701368773623  0.763959682901  0.722102425027  0.750417761764
 0.731404042423 ]
```

Solche Rekursionen sind in EMT schon vorprogrammiert. Hier zwei Beispiele. Die entsprechenden Funktion akzeptieren, wie üblich, Ausdrücke und Funktionen.

```
>iterate("cos",1,10)
 [ 0.540302305868  0.857553215846  0.654289790498  0.793480358743
 0.701368773623  0.763959682901  0.722102425027  0.750417761764
 0.731404042423  0.744237354901 ]
>iterate("cos",1)  // bis zur Konvergenz
 0.739085133216
>sequence("x[n-1]+x[n-2]",[1,1],10)  // allgemeine Rekursion
 [ 1  1  2  3  5  8  13  21  34  55 ]
```

Schleifen und Bedingungen können auch in der Kommandozeile angewendet werden, solange die Schleife und Bedingung in eine Zeile passt.

```
>x=1; n=6; v=zeros(1,n); v[1]=x;
>for i=2 to 6; x=(x+2/x)/2; v[i]=x; end;
>v
 [ 1  1.5  1.41666666667  1.41421568627  1.41421356237  1.41421356237 ]
```

Funktionen können rekursiv aufgerufen werden, das heißt sie können sich selbst aufrufen, oder in Funktionen aufgerufen werden, die sie selbst aufrufen. Dadurch sind sehr elegante Programme möglich, die aber nicht immer effizient sind.

```
>function ggtrek (x:nonnegative integer, y:nonnegative integer) ...
$  if x>y then return ggtrek(y,x);
$  elseif x==1 then return 1;
$  elseif x==0 then return y;
$  else return ggtrek(x,mod(y,x))
$  endif;
$endfunction
>ggtrek(25504574,6145974631)
 19
>ggtrek(17*19^2*31*41^3,19*41^3), factor(%)
 1309499
 [ 19  41  41  41 ]
>ggt(17*19^2*31*41^3,19*41^3)  // In Euler schon vorprogrammiert
 1309499
```

Selbstverständlich existiert die Funktion `ggt` schon in EMT. In diesem Beispiel verwenden wir zur Sicherheit typisierte Parameter. Die Funktion überprüft, ob die eingegeben Bedingungen für die Parameter erfüllt sind. Mehr darüber in der Referenz.

Programme in EMT können Default-Parameter haben. Solche Parameter werden zum Beispiel in `plot2d` oder `plot3d` sehr reichlich verwendet. Default-Parameter sind Parameter mit Default-Werten, die in der Form `param=value` deklariert werden. Wird kein Wert für den Parameter beim Aufruf eingegeben, also der Parameter weggelassen, so wird der Default-Wert verwendet.

Default-Parameter können auch durch Werte-Parameter mit `param=value` beim Aufruf der Funktion überschrieben werden, auch wenn diese Zuweisung nicht an der richtigen Parameterposition steht. Dafür haben wir schon viele Beispiele in dieser Einführung gesehen.

```
>function pnorm (v, p:nonnegative real=2) ...
$  if p==0 then return max(abs(v));
$  else return sum(abs(v)^p)^(1/p)
$endfunction
>pnorm(-2:2,0)  // p=0
 2
>pnorm(-2:2)  // Default p=2 aktiv
 3.16227766017
>pnorm(-2:2,p=1)  // Werte-Parameter p=1
 6
```

Ein wichtiges Thema sind Funktions-Parameter. Viele numerische Algorithmen und auch die Plot-Funktionen verwenden diese Technik, um Funktionen als Parameter entgegen zu nehmen. Wie schon im Abschnitt über numerische Algorithmen erläutert, werden Ausdrücke als Zeichenketten übergeben, die den Ausdruck enthalten. Die Übergabe von Funktionen ist mit dem

Funktionsnamen in einer Zeichenkette möglich. Symbolische Ausdrücke sind ebenfalls Zeichenketten. Symbolische Funktionen können einfach mit ihren Namen übergeben werden.

In der Funktion wird nun einfach die übergebene Zeichenkette mit Parametern aufgerufen. Als Beispiel programmieren wir die Iteration erneut, wobei wir als Abbruchkriterium den Vergleich ~= verwenden, der den relativen Fehler kleiner als das interne `epsilon` macht.

```
>function f(x) := cos(x)
>function fIter (f,xstart) ...
$  repeat
$    x=f(xstart);
$    until x ~= xstart;
$    xstart=x;
$  end;
$  return xstart;
$endfunction
>fIter("f",1)
 0.739085133214
>fIter("cos(x)",1)
 0.739085133214
```

Wie man sieht, funktioniert die Funktion sowohl mit Funktionsnamen, als auch mit Ausdrücken.

Wie kann man aber weitere Parameter für `f` festlegen? Funktionen können globale Variablen sehen, wenn `useglobal` aufgerufen wurde. Sie können sie allerdings nur im Wert, nicht im Typ, verändern. Ausdrücke können globale Variablen ebenfalls sehen. Globale Variablen sind daher eine eingeschränkte Möglichkeit, zusätzliche Parameter für `f` zu setzen?

Ein Problem entsteht lediglich noch, wenn zusätzliche Parameter benötigt werden, die nicht global sein sollen. Dazu gibt es in EMT Semikolon-Parameter und die Funktion `args`. Die Semikolon-Parameter werden mit Hilfe von `args()` an die Funktion *f* weitergereicht, die für diese Parameter definiert sein muss.

```
>function f(x,a) := cos(a*x)
>function fIter (f,xstart) ...
$  repeat
$    x=f(xstart,args());
$    until x ~= xstart;
$    xstart=x;
$  end;
$  return xstart;
$endfunction
>fIter("f",1;0.9)  // f mit a=0.9
 0.769576421736
```

Eine ähnliche Technik funktioniert auch für Ausdrücke. Ausdrücke können globale Variablen sehen. Wenn aber z.B. `fIter` in einem Unterprogramm verwendet wird, ist es nicht leicht möglich, globale Variablen zu verwenden, da die globale Variable schon vorher existieren müsste. Aber die Übergabe einer lokalen Variable funktioniert für Ausdrücke.

```
>function test ...
$  a=0.9;
$  return fIter("cos(a*x)",1;a);
$endfunction
>test
 0.769576421736
```

1.9 Euler-Dateien

Wenn viele Programme erstellt und verwendet werden sollen, ist es besser eine Euler-Datei zu schreiben, und diese Datei in EMT mit `load` zu laden. EMT-Dateien werden am einfachsten in dem Verzeichnis gespeichert, in dem das Notebook abgespeichert wird. Sie haben die Endung `*.e`, während Notebooks die Endung `*.en` haben. Beide Dateien können in Windows mit Doppelklick im Explorer geladen werden. EMT wird dann gestartet.

Es gibt auch einen Pfad, den EMT nach den Dateien absucht. In dem Pfad befinden sich normalerweise auch einige mitinstallierte EMT-Dateien.

```
>load perm
 Function to compute permutations.
 type "help perm.e" for more information.
>p := pcycle([1,2],3)
 [ 2  1  3 ]
>q := pcycle([2,3],3)
 [ 1  3  2 ]
>pmult(p,q), pprint(%)
 [ 3  1  2 ]
 (1 3 2)
```

Mit `help perm` kann man sich eine Übersicht über die in der Datei enthaltenen Definitionen verschaffen. Funktionsdefinitionen in EMT-Dateien sollten Hilfezeilen enthalten. Das sind Zeilen am Beginn der Definition, die mit `##` auskommentiert sind.

Die Datei `perm.e` sieht ungefähr folgendermaßen aus. Der Kommentar am Beginn wird beim Laden ausgegeben, wenn das `load`-Kommando nicht mit einem Semikolon abgeschlossen wurde.

```
comment
Function to compute permutations.
type "help perm.e" for more information.
endcomment

function pid (n)
## Identity permuation of n elements.
return 1:n
endfunction

function pmult (p,q)
## Multiply two permutations.
return q[p]
endfunction

...
```

EMT-Dateien können mit jedem Editor erstellt werden. Es wird aber ein Java-Editor mit EMT mitgeliefert, der die Syntax von EMT versteht. Um eine EMT-Datei zu erstellen, erzeugt man ein neues, leeres Notebook und speichert es unter einem aussagekräftigen Namen ab. Der ideale Ort ist das EMT-Verzeichnis in den eigenen Dateien. Danach gibt man die folgende Zeile ein.

```
>load testdatei
```

Wenn man in dieser Zeile die Taste F10 drückt, so öffnet sich ein externer Editor, mit dem man die Datei `testdatei.e` bearbeiten kann. Man muss dort zunächst bestätigen, dass die Datei neu angelegt werden soll. Alternativ wird mit F9 ein interner Editor geöffnet.

Man kann die Datei im externen Editor jederzeit speichern, und in EMT das `load`-Kommando starten. Die Datei wird dann geladen.

1.10 Maxima in Funktionen

Da es in EMT symbolische Funktionen gibt, ist es einfach, Maxima-Ergebnisse in EMT-Funktionen einzubauen. Im folgenden Beispiel erzeugen wir eine Funktion für den Gradienten einer Funktion und berechnen die Nullstelle mit dem Broyden-Verfahren (siehe Kapitel 4).

Wir verwenden zur Definition der Funktion und des Gradienten Vektor-Parameter. Dieser Trick erlaubt es, die Funktion als Funktion von zwei Variablen `f(x,y)` oder als Funktion eines Vektors `f(v)` aufzurufen. Das Broyden-Verfahren benötigt einen Vektor. Die symbolische Funktion kann dagegen nur mit zwei Parametern aufgerufen werden.

Die Nullstelle des Gradienten lässt sich hier übrigens nicht exakt berechnen. Es handelt sich also um ein Beispiel, in dem symbolisches Rechnen und numerisches Rechnen ineinandergreifen müssen.

```
>function f([x,y]) &= x^2+exp(x*y+1)-10*y^2+x-y
```

$$E^{x\,y+1} - 10\,y^2 - y + x^2 + x$$

```
>plot3d(&f(x,y),r=2,>contour,angle=90°);
>function Df([x,y]) &= gradient(f(x,y),[x,y])
```

$$\left[y\,E^{x\,y+1} + 2\,x + 1,\ x\,E^{x\,y+1} - 20\,y - 1\right]$$

```
>broyden("Df",[0,0])
 [ -0.358399529287  -0.100498041067 ]
```

Auch das schnellere Newton-Verfahren kann hier verwendet werden, falls es auf Geschwindigkeit ankommen sollte. Sogar eine garantierte Einschließung der Lösung mit Hilfe eines Interval-Newton-Verfahrens ist möglich (siehe Kapitel 4).

```
>function JDf([x,y]) &= jacobian(Df(x,y),[x,y])
```

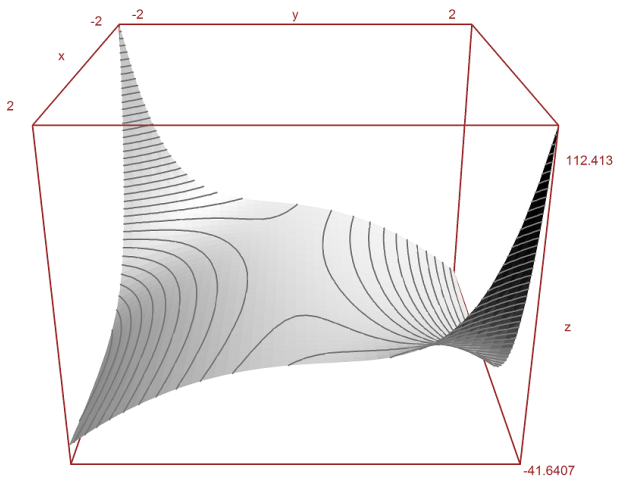

Abbildung 1.6: $x^2 + e^{xy+1} - 10y^2 - y$

```
[      2  x y + 1              x y + 1    x y + 1 ]
[     y  E        + 2      x y E        + E       ]
[                                                 ]
[      x y + 1    x y + 1      2  x y + 1         ]
[ x y E        + E           x  E        - 20     ]
```

```
>v0 := newton2("Df","JDf",[0,0])
 [ -0.358399529287  -0.100498041067 ]
>inewton2("Df","JDf",v0)
 [ ~-0.35839952928729046,-0.35839952928728969~
 ~-0.10049804106749669,-0.10049804106749641~ ]
```

Nicht nur in symbolischen Funktionen können von Maxima erzeugte symbolische Ausdrücke verwendet werden. In jeder Funktion kann man an beliebiger Stelle diese Ausdrücke einfügen. Die Syntax dafür ist `&:...` und wird zu der Zeit ausgewertet, wenn die Funktion eingegeben wurde. Eine Änderung der Funktion `f` im folgenden Beispiel wirkt sich also nicht automatisch auf `newtonIter` aus.

```
>function f(x) &= x^x;
>function newtonIter (x,y=0) ...
$  repeat;
$     f = &:f(x);
$     fd = &:diff(f(x),x);
$     xnew = x - (f-y)/fd;
$     until xnew~=x;
```

```
$     x = xnew;
$  end;
$  return xnew
$endfunction
>newtonIter(1,y=2), f(%)
 1.55961046946
 2
```

Die Funktion `newtonIter` sieht nun folgendermaßen aus. Sie enthält in der Tat die korrekten Ausdrücke.

```
>type newtonIter
 function newtonIter (x, y)
 ## Default for y : 0
 repeat;
      f = x^x;
      fd = x^x*(log(x)+1);
      xnew = x - (f-y)/fd;
      until xnew~=x;
      x = xnew;
   end;
   return xnew
 endfunction
```

Mehr zum Newton-Verfahren findet sich im Kapitel 4

Kapitel 2

Fehlerabschätzungen

2.1 Gleitkommaarithmetik

Die heute durchgängig verwendete Rechnerarithmetik ist eine Gleitkommaarithmetik. Zahlen werden als

$$0.d_1 \dots d_n \cdot b^m$$

dargestellt mit einer Basis b (normalerweise $b = 2$, selten $b = 10$), einer Mantisse

$$0.d_1 \dots d_n = d_1 b^{-1} + \dots d_n b^{-n}$$

mit Ziffern $d_i \in \{0, \dots, b-1\}$, die zur Basis b dargestellt wird, und einem Exponenten m.

Im Standard IEEE Double Precision ist

$$b = 2, \quad n = 53, \quad -1022 \leq m \leq 1023,$$

und eine Zahl verbraucht 8 Byte=64 Bit.

2.1. Beispiel: In EMT kann man die Darstellung direkt sehen.

```
>printdual(8)
 1.00000000000000000000000000000000000000000000000000000*2^3
>printdual(0.1)
 1.10011001100110011001100110011001100110011001100110011010*2^-4
```

2.2 Aufgabe: Die Dualdarstellung von $x \in]0, 1[$ erhält man, indem man diese Zahl immer wieder mit 2 multipliziert, und den ganzzahligen Anteil abschneidet. Ermitteln Sie die Dualdarstellung von 3/4, 127/128, 0.1 und 1/3.

2.3 Aufgabe: Welche Zahlen haben eine endliche Darstellung im Dualsystem? Welche im Dezimalsystem? Kann jede Zahl mit endlicher Darstellung im Dualsystem, endlich im Dezimalsystem ausgedrückt werden?

2.4. Definition: Die Menge der darstellbaren Gleitkommazahlen bezeichnen wir mit G. Ein Rundungsoperator ist eine Abbildung

$$\square : \mathbb{R} \to G$$

die monoton wachsend ist, und die Eigenschaft

$$\square x = x \qquad \text{für alle } x \in G$$

hat. Gewöhnlich wird wird zur nächsten Gleitkommazahl gerundet, im Zweifel von der 0 weg (kaufmännische Rundung).

In der Praxis kann man Programme so einstellen, dass $\square x$ einen Overflow liefert, für

$$x \notin [\min G, \max G],$$

oder dass in diesem Fall spezielle Zahlen MINF und INF geliefert werden. Ebenso kann man Software so einstellen, dass sie einen Underflow liefert, wenn $|x|$ zu klein wird, oder dass in diesem Fall 0 geliefert wird.

```
>underflows off; 10^(-1000)
 0
>underflows on; 10^(-1000)
 Underflow or overflow error.
 Error in ^
```

2.5. Definition: Die Maschinengenauigkeit ϵ einer Gleitkommadarstellung ist die kleinste positive Zahl, so dass gilt

$$\left|\frac{x - \square x}{\square x}\right| \leq \epsilon.$$

für alle $x \neq 0$ gilt, die keinen Overflow oder Underflow verursachen. Der Ausdruck auf der linken Seite ist der relative Rundungsfehler. Bei der kaufmännischen Rundung gilt dann

$$\square(1 + \epsilon) \neq 1,$$

und ϵ ist die kleinste Zahl mit dieser Eigenschaft. Für IEEE Double Precision ist

$$\epsilon = 2^{-53} \approx 10^{-16},$$

so dass man eine relative Genauigkeit von 15-16 Stellen erhält. Bei beliebigen anderen Rundungen gilt jedenfalls

$$\epsilon \leq \min\{g - 1 \in G : g > 1\}.$$

2.6 Aufgabe: Geben Sie in EMT das Kommando

```
>longestformat; n=60; 1+2^(-n)-1,
```

ein. Erniedrigen Sie n solange, bis das Ergebnis nicht mehr 0 ist. Was schließen Sie damit für die in EMT tatsächlich verfügbare duale Stellenzahl?

2.7. Definition: Funktionen und Operatoren sollen in der Gleitkommaarithmetik so ausgeführt werden, dass das exakte Ergebnis gerundet wird, also

$$\boxed{f}(x) := \square(f(x)).$$

bzw.

$$x \boxed{\circ} y = \square(x \circ y).$$

Dies lässt sich nicht immer durchhalten (Transzendente Funktionen oder nicht effektiv berechenbare Funktionen). Deswegen erlauben wir, dass der relative Rundungsfehler größer, z.B. doppelt so groß, werden kann.

2.8. Beispiel: Die interne Darstellung der Zahl π im IEEE Double Precision ist

$$\begin{aligned}\square\pi &= 3.14159265358979311599796346854418516159057617187 5\\ &= \frac{884279719003555}{2^{48}}\end{aligned}$$

Berechnet man den Sinus dieser Zahl auf 100 Stellen (etwa mit der Arithmetik von Maxima) so erhält man nicht exakt 0, sondern die Zahl

$$1.2246467991473531772\ldots\cdot 10^{-16}$$

In der Tat liefert EMT auch dieses Ergebnis

```
>longestformat; sin(pi)
  1.224646799147353e-016
```

2.2 Fehlerfortpflanzung

2.9. Definition: Wir bezeichnen den maximalen Faktor, um den sich die relative Genauigkeit bei einer Funktion oder Operation verbessert oder verschlechtert (unabhängig von der Rundung), als Kondition der Funktion oder der Operation.

Sei Δx etc. der absolute Fehler, und f bei x differenzierbar. Dann haben wir für $y = f(x)$ aufgrund des Mittelwertsatzes

$$\frac{\Delta y}{y} = \frac{f'(\xi)\Delta x}{y} = \frac{xf'(\xi)}{y}\cdot\frac{\Delta x}{x}$$

Also ist die Kondition einer einfachen Funktion in erster Näherung

$$\mathrm{cond}_f = \left|\frac{xf'(x)}{y}\right|$$

2.10. Beispiel: Die Kondition der Potenzierung $y = x^\alpha$ ist

$$\left|\frac{x\alpha x^{\alpha-1}}{x}\right| = |\alpha|.$$

Das Quadrieren ist also schlecht konditioniert mit $\alpha = 2$, und das Wurzelziehen ist gut konditioniert mit $\alpha = 0.5$.

Man kann dies mit EMT beobachten, indem man $x_0 = 2$ einige Male mit der Wurzelfunktion abbildet, und das dann durch Quadrieren rückgängig macht.

```
>x=2; for i=1 to 30; x=sqrt(x), end // 30 mal Wurzelziehen
       1.414213562373095
       1.189207115002721
       1.090507732665258
       ...
       1.000000001291087
       1.000000000645544
>2^((1/2)^30) // Das Ergebnis ist exakt!
       1.000000000645544
>for i=1 to 30; x=x^2, end // Sollte wieder auf 2 kommen!
       1.000000001291087
       1.000000002582175
       ...
       1.189207113780447
       1.414213559466022
       1.999999991777554
```

2.11. Beispiel: Es gilt

$$\lim_{n\to\infty} \left(1\boxed{+}1\boxed{/}n\right)\boxed{\hat{}}n = 1$$

wegen

$$1\boxed{+}1\boxed{/}n = 1 \qquad \text{für alle } n \geq N$$

mit $N = 2^{55}$ in IEEE Double Precision. Die relative Ungenauigkeit von $1 + 1/n$ wird dabei mit n multipliziert.

2.12 Aufgabe: Überlegen Sie sich, wie man $(1 + x)^n$ auch für kleine x exakt ausrechnen könnte.

2.13. Beispiel: Für die Funktion $f(x) = cx$ erhält man nach obiger Rechnung

$$\text{cond}_f = 1.$$

Dies gilt allerdings nur, wenn c exakt im Rechner darstellbar ist. Sonst muss man die Ungenauigkeit von c mit berücksichtigen.

2.14. Beispiel: Bei Funktionen mehrerer Variablen

$$y = f(x_1, \ldots, x_n)$$

erhält man für die Kondition analog

$$\text{cond}_f \leq \left|\frac{x_1 \frac{\partial f}{\partial x_1}(x)}{y}\right| + \ldots + \left|\frac{x_n \frac{\partial f}{\partial x_n}(x)}{y}\right|$$

Im Grenzfall ist dies die einzig mögliche Abschätzung für den maximalen Faktor, also gleich der Kondition.

2.15. Beispiel: Die Multiplikation ist relativ stabil. Wir haben für

$$f(x_1, x_2) = x_1 x_2$$

eine Abschätzung die Konditionszahl $\text{cond}_f = 2$. In der Tat gilt

$$(1 + \epsilon)(1 + \epsilon) = 1 + 2\epsilon + \epsilon^2$$

Der relative Fehler ist also doppelt so groß. Dies stimmt mit unseren Beobachtungen für $f(x) = x^2$ überein.

2.3 Auslöschung

Die Subtraktion ist schlecht konditioniert, wenn das Ergebnis gegenüber den Ausgangswerten klein ist. Wir haben für $f(x_1, x_2) = x_1 - x_2$

$$\text{cond}_f = \frac{|x_1| + |x_2|}{|x_1 - x_2|}.$$

Diese Phänomen nennt man Auslöschung.

2.16. Beispiel: Es gilt

$$1 \boxed{+} \epsilon \boxed{-} 1 = 0,$$

wenn nur ϵ klein genug ist. Obwohl 1 exakt darstellbar ist, ist der Fehler immens. Man kann das durch Umordnen vermeiden.

$$1 \boxed{-} 1 \boxed{+} \epsilon = \epsilon,$$

indem man zunächst die großen Werte verarbeitet. Manche Software besitzt auch ein exaktes Skalarprodukt.

```
>longestformat;
>accuload(0);
>accuadd([1,1e-20,-1])
                  1e-020
```

2.17. Beispiel: Eine gute Formel für numerisches Differenzieren ist

$$m \approx \frac{f(x+h) - f(x-h)}{2h}.$$

Allerdings leidet die Formel unter der Auslöschung. Man sieht in EMT sehr schön, woran das liegt. Als Funktion verwenden wir $f(x) = x^2$, und berechnen die Ableitung in 1.

```
>function f(x) &= x^2;
>h=1e-8; longestformat;
>f(1+h)
               1.00000002
>f(1-h)
               0.99999998
>f(1+h)-f(1-h)
  3.999999986792346e-008
>(f(1+h)-f(1-h))/(2*h)
       1.999999993396173
```

Das korrekte Ergebnis wäre

$$\frac{(1+h)^2 - (1-h)^2}{2h} = 2 + h^3.$$

Es müsste also auf 24 Stellen genau sein. Aber die Gleitkommaarithmetik berechnet die Differenz $f(1+h) - f(1-h)$ lediglich auf 8 Stellen. Eine Rechnung mit Maxima mit der langsamen Arithmetik auf 30 Stellen ergibt natürlich ein besseres Ergebnis.

```
>&(f(1b0+1b-8)-f(1b0-1b-8))/(2b-8)
                1.99999999999999999999999999502262b0
```

2.18 Aufgabe: Zeigen Sie, dass für eine dreimal stetig differenzierbare Abbildung f gilt

$$\frac{f(x+h)-f(1-h)}{2h} = f'(x) + R(h)$$

mit

$$|R(h)| \le \frac{1}{64} \max_{x-h\le\xi\le x+h} f'''(\xi)\, h^3.$$

2.19 Aufgabe: Lösen Sie die quadratische Gleichung

$$x^2 + 12331233.1233x + 0.00010001 = 0$$

mit quadratischer Ergänzung nach x auf. Verbessern Sie die eine Nullstelle mit dem Satz von Peano über das Produkt der beiden Nullstellen.

2.20 Aufgabe: Lösen Sie die Gleichung

$$\cos(x) + \frac{1}{\cos(x)} = 2 + c$$

für sehr kleine c, indem Sie $\cos(x) = 1 + d$ setzen, und anschließend eine Näherungsformel für $\cos(x)$ verwenden.

2.21 Aufgabe: Sei

$$y_n = \int_0^1 \frac{x^n}{x+5}\, dx$$

Beweisen Sie die Rekursionsformel

$$y_{n+1} = \frac{1}{n+1} - 5y_n.$$

Zeigen Sie, dass y_n gegen 0 konvergiert. Berechnen Sie y_{40} numerisch mit Hilfe dieser Rekursionsformel. Warum ist das Ergebnis so schlecht?

Berechnen Sie y_{40} dann mit Hilfe der rückwärts ausgewerteten Rekursionsformel, wobei Sie mit $y_{80} = 0$ starten. Setzen Sie diese umgekehrte Rekursion bis y_0 fort und vergleichen Sie mit dem exakten Ergebnis.

2.4 Intervallarithmetik

Um den Rundungsfehler in einem Programm zu kontrollieren, können wir bei jedem Schritt die Grenzen der Werte als Intervall

$$X = [a, b]$$

festhalten.

2.22. Definition: Funktionen und Operatoren werden auf Intervalle so angewendet, dass das Ergebnis ein Intervall ist, das alle möglichen Ergebnisse umfasst. Also zum Beispiel für Funktionen

$$\boxed{f}(X) \supseteq f(X) = \{f(x) : x \in X\}$$

und für Operatoren

$$X \boxed{\circ} Y \subseteq X \circ Y = \{x \circ y : x \in X \text{ und } y \in Y\}.$$

Man versucht immer, das kleinste Intervall zu wählen.

Man beachte, dass

$$[-1,1]^2 = [0,1]$$

gilt, aber

$$[-1,1] \cdot [-1,1] = [-1,1],$$

da der Operator auf alle Elemente von $[-1,1]$ unabhängig in den Operanden angewendet wird.

2.23. Beispiel: Für Messergebnisse hat man oft eine Fehlerschranke. Wenn etwa die Fallzeit im Vakuum mit $10.2s$ gemessen wird, und die Erdbeschleunigung mit $9.81m/s^2$ bekannt ist, so kann man die Eingabewerte mit der $\pm$-Syntax in EMT als Intervalle eingeben, und intervallmäßig rechnen.

```
>t=10.2±0.05, left(t), right(t)
 ~10.1,10.3~
 10.15
 10.25
>g=9.81±0.005
 ~9.8,9.82~
>1/2*g*t^2
 ~505,516~
```

Man erhält mit diesem Ungenauigkeiten ein recht beachtliches Intervall.

2.24 Aufgabe: Rechnen Sie das Beispiel ohne Intervallrechnung per Hand nach.

2.25. Beispiel: Die Fehler beim numerischen Differenzieren fallen bei einer Intervallrechnung automatisch auf. Bei intervallmäßger Rechnung ensteht ein Ergebnis mit sehr schlechter Genauigkeit.

```
>delta=~1e-12~; // intervallmäßiges delta
>(exp(delta)-exp(-delta))/(2*delta)
                      ~0.9998113,1.000256~
```

2.26. Beispiel: Wir berechnen die beiden Nullstellen von

$$x^2 + 12331233.1233x + 0.00010001 = 0$$

mit der Intervallarithmetik von EMT, wobei wir annehmen, dass die Koeffizienten völlig exakt gegeben sind. Dabei verwenden wir den Trick mit $z_1 z_2 = q$. Um die intervallmäßige Rechnung in EMT anzustoßen, machen wir aus den exakten Koeffizienten kleine Intervalle.

```
>p=~12331233.1233~;
>q=~0.00010001~;
>z1 = -p/2 + sqrt(p^2/4-q) // sehr ungenau
                        ~-5.6e-009,5.6e-009~
>z2 = -p/2 - sqrt(p^2/4-q)// genauer
 ~-12331233.123300007,-12331233.123299992~
>z1 = q/z2 // genauer
 ~-8.1103000000081202e-012,-8.1103000000081025e-012~
```

Nimmt man allerdings an, dass die Koeffizienten nur auf die angegebene Stellenzahl genau sind, so erhält man natürlich größere Ergebnisse.

```
>p=12331233.1233±5e-5
         ~12331233.123249,12331233.123351~
>q=0.00010001±5e-9
                ~0.000100005,0.00010001501~
>z1 = -p/2 + sqrt(p^2/4-q) // sehr ungenau
                           ~-5e-005,5e-005~
>z2 = -p/2 - sqrt(p^2/4-q)// genauer
       ~-12331233.123351,-12331233.123249~
>z1 = q/z2 // etwas genauer
         ~-8.1107055e-012,-8.1098945e-012~
```

2.27 Aufgabe: Geben Sie die Potenzreihenentwicklung von

$$F(x) \int_0^x e^{-t^2}\, dt$$

an, sowie eine Abschätzung von $|F(x) - T_9(x)|$ für $|x| \leq 1/2$. Berechnen Sie $F(1/2)$ intervallmäßig mit dieser Abschätzung.

Zur Verbesserung der Genauigkeit bei der intervallmäßigen Auswertung einer Funktion kann man den Mittelwertsatz verwenden. Es gilt nämlich offenbar

$$f[a,b] \subseteq f\left(\frac{a+b}{2}\right) + f'[a,b] \cdot \left[-\frac{b-a}{2}, \frac{b-a}{2}\right]$$

mit intervallmäßiger Rechnung. Eine weitere Verbesserung ist möglich, indem man die intervallmäßige Auswertung in Teilintervalle aufteilt. Dies gilt wegen

$$f(I_1 \cup \ldots \cup I_n) = f(I_1) \cup \ldots \cup f(I_n).$$

Kombiniert man beide Methoden, so erreicht man sehr gute Einschließungen.

2.28 Aufgabe: Beweisen Sie die erste Gleichung in dieser Bemerkung.

Kapitel 3

Interpolation und Approximation

3.1 Haarsche Unterräume

3.1. Definition: Sei X eine Menge und V eine Menge von Abbildungen von X nach $\mathbb{K}$ ($\mathbb{K} = \mathbb{R}$ oder $\mathbb{K} = \mathbb{C}$). Das Interpolationsproblem besteht darin, zu paarweise verschiedenen Punkten

$$x_1, \ldots, x_n \in X$$

und Werten

$$y_1, \ldots, y_n \in \mathbb{K}$$

eine Funktion $v \in V$ zu finden mit

$$v(x_k) = y_k \qquad \text{für } k = 1, \ldots, n.$$

Man sagt dann v interpoliert die Werte $y_1, \ldots, y_n$ in diesen Punkten. Falls

$$y_k = f(x_k)$$

für eine Funktion k, so sagt man v interpoliere f in diesen Punkten. Ein Hermite-Interpolationsproblem besteht darin zusätzlich noch Ableitungen zu interpolieren, d.h., es gilt in den Punkten

$$v(x_k) = f(x_k), v'(x_k) = f'(x_k), \ldots, v^{(m_k-1)}(x_k) = f^{(m_k-1)}(x_k)$$

mit gewissen Vielfachheiten m_k, $k = 1, \ldots, n$.

3.2. Beispiel: Sei $X = \mathbb{R}$, $f : X \to \mathbb{R}$, und $x_1 < x_2$. Dann interpoliert die Sekante

$$p(x) = f(x_1) + \frac{f(x_2) - f(x_1)}{x_2 - x_1}(x - x_1) = f(x_1)\frac{x - x_2}{x_1 - x_2} + f(x_2)\frac{x_1 - x}{x_1 - x_2}$$

die Gerade. Dabei ist V der Raum der Geradenfunktionen. Die Tangente

$$p(x) = f(x_1) + f'(x_1)(x - x_1)$$

interpoliert mit der Vielfachheit 2 in x_1.

3.3 Aufgabe: Rechnen Sie nach, dass diese Geraden p tatsächlich die gewünschte Interpolationseigenschaft haben. Überlegen Sie sich, dass die Geraden durch die Interpolationseigenschaft eindeutig bestimmt sind, indem Sie eine zweite Interpolierende $\tilde{p}$ annehmen, und die Differenz betrachten.

3.4. Beispiel: Das Interpolationsproblem ist ein lineares Gleichungssystem, wenn V ein linearer Teilraum endlicher Dimension von $\mathcal{A}(X, \mathbb{K})$ ist. Sei nämlich

$$V = \text{span}\,\{v_1, \ldots, v_n\},$$

und

$$x_1, \ldots, x_n \in X,$$

so ist das Interpolationsproblem äquivalent mit dem linearen Gleichungssystem

$$\begin{pmatrix} v_1(x_1) & \ldots v_n(x_1) \\ \vdots & \vdots \\ v_1(x_n) & \ldots v_n(x_n) \end{pmatrix} \cdot \begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}.$$

Die Funktion

$$v = a_1 v_1 + \ldots + a_n v_n$$

erfüllt dann die Interpolationsbedingungen

$$v(x_k) = y_k \qquad \text{für } k = 1, \ldots, n.$$

Die Möglichkeit oder Eindeutigkeit der Interpolation hängt von den Eigenschaften dieser Matrix ab.

3.5. Definition: Sei $\mathcal{A}(X, \mathbb{K})$ der Vektorraum der Abbildungen von X nach $\mathbb{K}$, und $V \subseteq \mathcal{A}(X, \mathbb{K})$ ein Unterraum. Dann heißt V Haarscher Unterraum der Dimension n, wenn gelten

(1) $\dim V = n$.

(2) Zu je n paarweise verschiedenen Punkten, und n Werten aus $\mathbb{K}$ existiert ein eindeutiges Interpolationspolynom aus V.

3.6 Aufgabe: Überlegen Sie sich, dass X dann mindestens n Punkte enthalten muss, wenn $\dim V = n$ ist.

3.7 Satz: (Äquivalente Definition) *V ist genau dann Haarscher Unterraum der Dimension n, wenn gelten*

(1) $\dim V = n$.

(2') *Jede Funktion $v \in V$ mit n Nullstellen ist identisch* 0.

Beweis: Offenbar folgt (2') aus (2). Umgekehrt betrachten wir für n paarweise verschiedene Punkte $x_1, \ldots, x_n \in X$ die lineare Abbildung $\phi : V \to \mathbb{K}^n$ mit

$$\phi(v) = (v(x_1), \ldots, v(x_n)).$$

Dann besagt (2'), dass der Kern von ϕ nur aus der Nullfunktion besteht. Deswegen ist ϕ bijektiv, und daher die Interpolation immer eindeutig möglich. Es folgt (2). **q.e.d.**

3.8 Satz: *X enthalte mindestens n Punkte, dann ist*

$$V = \operatorname{span}\{v_1, \ldots, v_n\}$$

genau dann ein Haarscher Unterraum der Dimension n, wenn für jedes

$$v = a_1 v_1 + \ldots + a_n v_n$$

mit mindestens n Nullstellen folgt, dass

$$a_1 = \ldots = a_n = 0$$

ist.

Beweis: Nehmen wir an, dass diese Bedingungen gelten. Wegen $|X| \geq n$ folgt leicht, dass $v_1, \ldots, v_n$ linear unabhängig sind. Also gilt $\dim V = n$, und diese Funktionen bilden eine Basis von V. Wenn v nun n Nullstellen hat, so folgt aufgrund der Bedingung, dass v identisch 0 ist. Also ist V ein Haarscher Unterraum der Dimension n.

Sei umgekehrt V ein Haarscher Unterraum der Dimension n, dann müssen $v_1, \ldots, v_n$ eine Basis von V sein. Wenn also das angegebene v mindestens n Nullstellen hat, so folgt, dass es identisch 0 ist. Da $v_1, \ldots, v_n$ linear Unabhängig sind, folgt $a_1 = \ldots = a_n = 0$. **q.e.d.**

3.9. Definition: Wir definieren den Raum

$$\mathcal{P}_n = \mathcal{P}_n^{\mathbb{K}}(X)$$

der reellen bzw. komplexen Polynome

$$p(x) = a_0 + a_1 x + \ldots + a_n x^n$$

vom Grade höchstens n auf X.

3.10 Satz: *Wenn $X \subseteq \mathbb{K}$ mindestens $n+1$ Punkte enthält, so ist $\mathcal{P}_n(X)$ ein Haarscher Unterraum der Dimension $n+1$ auf X. Insbesondere sind die Funktionen*

$$v_0(x) = 1, \quad v_1(x) = x, \quad \ldots, \quad v_n(x) = x^n$$

eine Basis von $\mathcal{P}_n$.

Beweis: Angenommen

$$p(x) = a_0 + a_1 x + \ldots + a_n x^n$$

hat die Nullstellen $x_0, \ldots, x_n$. Dann kann man die Nullstellen $x_1, \ldots, x_n$ ausdividieren, und p hat die Form

$$p(x) = a_n \cdot (x - x_1) \cdot \ldots \cdot (x - x_n)$$

Wegen $p(x_0) = 0$ folgt $a_n = 0$, also

$$p(x) = a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$$

Rekursiv folgt auf dieselbe Weise

$$a_{n-1} = \ldots = a_0 = 0.$$

q.e.d.

3.11 Aufgabe: Warum wird der Beweis sehr viel einfacher, wenn man $\dim \mathcal{P}_n = n+1$ auf X voraussetzt?

3.12. Beispiel: Seien $\lambda_1, \ldots, \lambda_n \in \mathbb{R}$ paarweise verschieden. Wir betrachten den von den Funktionen

$$e^{\lambda_1 x}, \ldots, e^{\lambda_n x}$$

aufgespannten Raum $V \subset C(\mathbb{R})$. Wir behaupten, dass V ein Haarscher Unterraum der Dimension n ist. Der Beweis geht per Induktion, wobei $n = 1$ klar ist. Eine Funktion

$$v(x) = a_1 e^{\lambda_1 x} + \ldots + a_n e^{\lambda_n x}$$

habe mindestens n Nullstellen

$$x_1 < \ldots < x_n.$$

Dann hat auch

$$\tilde{v}(x) = v(x) e^{-\lambda_1 x} = a_1 + a_2 e^{(\lambda_2 - \lambda_1)x} + \ldots + a_n e^{(\lambda_n - \lambda_1)x}$$

diese Nullstellen. Nach dem Satz von Rolle hat die Ableitung

$$\tilde{v}'(x) = a_2(\lambda_2 - \lambda_1) e^{(\lambda_2 - \lambda_1)x} + \ldots + a_n(\lambda_n - \lambda_1) e^{(\lambda_n - \lambda_1)x}$$

zwischen diesen Nullstellen immer noch $n-1$ Nullstellen. Aufgrund der Induktionsvoraussetzung folgt

$$a_2(\lambda_2 - \lambda_1) = \ldots = a_n(\lambda_n - \lambda_1) = 0.$$

Weil die λ_k paarweise verschieden sind, folgt

$$a_2 = \ldots = a_n = 0.$$

Es folgt dann auch $a_1 = 0$.

3.13 Satz: *X enthalte mindestens n Punkte, dann ist*

$$V = \operatorname{span}\{v_1, \ldots, v_n\}$$

genau dann ein Haarscher Unterraum der Dimension n, wenn die Interpolation in je n paarweise verschiedenen Punkten immer möglich ist.

Beweis: Wir betrachten für n paarweise verschiedenen Punkte $x_1, \ldots, x_n \in X$ die lineare Funktion $\phi : \mathbb{K}^n \to \mathbb{K}^n$ mit

$$\phi(a_1, \ldots, a_n) = (v(x_1), \ldots, v(x_n)),$$

mit

$$v = a_1 v_1 + \ldots a_n v_n.$$

Wenn die Interpolation immer möglich ist, muss diese Funktion surjektiv sein. Also ist sie auch injektiv, und ihr Kern ist der Nullraum. Die Aussage folgt daher aus dem vorigen Satz. **q.e.d.**

3.14. Beispiel: Damit hätten wir $\mathcal{P}_n$ auch klären können. Die Interpolation ist nämlich immer möglich, weil das Interpolationspolynom in der Form

$$p(x) = \sum_{k=1}^{n} y_k L_k(x)$$

mit

$$L_k(x) = \frac{(x - x_1) \ldots (x - x_{k-1})(x - x_{k+1}) \ldots (x - x_n)}{(x_k - x_1) \ldots (x_k - x_{k-1})(x_k - x_{k+1}) \ldots (x_k - x_n)}$$

angeschrieben werden kann. Man nennt diese Polynome die Lagrangesche Grundpolynome. Mit der Notation

$$\omega(x) = (x - x_1) \cdot \ldots \cdot (x - x_n)$$

kann sie in der Form

$$L_k(x) = \frac{\omega(x)}{(x - x_k)\omega'(x_k)}$$

schreiben.

3.15 Aufgabe: Rechnen Sie die Behauptungen dieses Beispiels nach.

3.16. Beispiel: Die Lagrangeschen Grundpolynome können wir in EMT auf die folgende Weise numerisch berechnen und plotten. Man beachte, dass wir die Funktion `L` nur auf ihre ersten beiden Argumente elementweise anwenden, falls diese Elemente Vektoren sind. Der Knotenvektor `xk` bleibt außen vor.

```
>function map L(x,k;xk)...
$  hn=x-xk; hn[k]=1;
$  hz=xk[k]-xk; hz[k]=1;
$  return prod(hn)/prod(hz);
$endfunction
>xk := -2:2;
>plot2d("L(x,4,xk)",-2,2);
>yk := xk==1;
>plot2d(xk,yk,>points,>add);
```

Selbstverständlich wäre es auch möglich, diese Polynome in Maxima zu berechnen. Wir verwenden hier rein symbolische Funktionen, die erst zur Laufzeit ausgewertet werden.

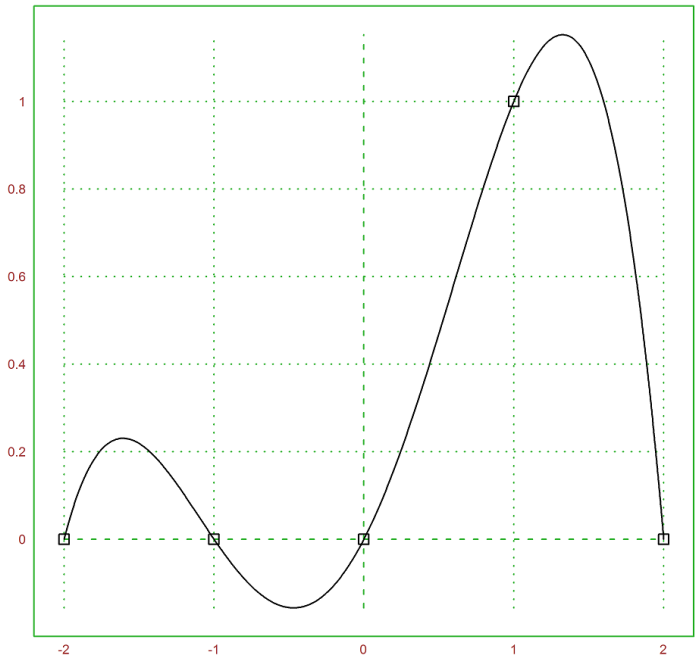

Abbildung 3.1: Lagrange-Polynom

```
>xk ::= -2:2
 [ -2  -1  0  1  2 ]
>function omega(x) &&= prod(x-xk[k],k,1,5)

                      product(x - xk , k, 1, 5)
                                k

>function L(x,k) &&= omega(x)/((x-xk[k])*diffat(omega(x),x=xk[k]))

                           omega(x)
                  ------------------------------------
                  (x - xk ) diffat(omega(x), x = xk )
                         k                          k

>&L(x,3)

                    (x - 2) (x - 1) (x + 1) (x + 2)
                    -------------------------------
                                  4
```

Für Polynome nimmt die Matrix aus dem Beispiel 4 die Form

$$\begin{pmatrix} 1 & x_0 & \dots & x_0^n \\ \vdots & \vdots & \dots & \vdots \\ 1 & x_n & \dots & x_n^n \end{pmatrix}$$

an. Die Matrix heißt Vardermondesche Matrix. Ihre Determinante ist

$$V(x_0,\dots,x_n = \prod_{0\le k<l\le n}(x_k - x_l).$$

Für paarweise verschiedene Punkte ist sie nicht 0. Dies beweist ebenfalls die Eindeutigkeit und Existenz der Interpolation.

3.17 Aufgabe: Beweisen Sie diese Formel per Induktion nach n, indem Sie das x_0-fache der j-ten von der $j+1$-ten Spalte abziehen, beginnend mit der vorletzten, und anschließend die erste Zeile von allen anderen.

3.18 Aufgabe: Zeigen Sie durch Entwickeln nach der ersten Zeile, dass

$$p(x) = V(x, x_1, \dots, x_n)$$

ein Polynom in $\mathcal{P}_n$ ist. Zeigen Sie, dass dieses Polynom die Form

$$p(x) = V(x_1,\dots,x_n)\cdot(x-x_1)\cdot\ldots\cdot(x-x_n)$$

haben muss, indem Sie den höchsten Koeffizienten und die Nullstellen von p bestimmen. Beweisen Sie nun die Gleichung mit Induktion nach n.

3.19. Beispiel: Wir rechnen die Behauptung symbolisch für 4 Punkte nach.

```
>xk &= [a,b,c,d]
```

$$[a,\ b,\ c,\ d]$$

```
>V &= genmatrix(lambda([i,j],xk[i]^(j-1)),4,4)
```

$$\begin{bmatrix} 1 & a & a^2 & a^3 \\ 1 & b & b^2 & b^3 \\ 1 & c & c^2 & c^3 \\ 1 & d & d^2 & d^3 \end{bmatrix}$$

```
>&factor(det(V))
```

$$(b-a)\,(c-a)\,(c-b)\,(d-a)\,(d-b)\,(d-c)$$

Numerisch könnte man mit Schleifen arbeiten. Man beachte die Ungenauigkeit bei der Berechnung der Determinanten mit dem Gauß-Verfahren. Diese Matrizen sind sehr schlecht zu berechnen.

```
>a=1:10; det(a^(0:9)')
 1.83493343011e+021
>p=1; n=length(a);
>for i=2 to n; for j=1 to i-1; p=p*(a[i]-a[j]); end; end;
>p
 1.83493347225e+021
```

Es geht aber auch wesentlich einfacher, indem man die Produkte der Matrix

$$(a[i]-a[j])_{i,j}$$

bildet, nachdem man die Diagonale auf 1 gesetzt hat.

```
>sqrt(abs(prod(prod(setdiag(a-a',0,1))')))
 1.83493347225e+021
```

3.20 Satz: *Sei $V \subset C(I)$ ein Haarscher Unterraum der Dimension n, $I \subseteq \mathbb{R}$ ein Intervall, $v_1, \ldots, v_n$ eine Basis von V. Dann hat*

$$V(x_1, \ldots, x_n) = \det \begin{pmatrix} v_1(x_1) & \ldots & v_n(x_1) \\ \vdots & & \vdots \\ v_1(x_n) & \ldots & v_n(x_n) \end{pmatrix}$$

für alle gemäß

$$x_1 < \ldots < x_n$$

angeordneten Punkte dasselbe Vorzeichen. Wenn $v \in V$ auf einer Punktmenge

$$x_0 < \ldots < x_n$$

schwach abwechselndes Vorzeichen hat, also

$$v_(x_0) \leq 0, \quad v(x_1) \geq 0, \quad \ldots,$$

dann gilt $v = 0$.

Beweis: Für zwei angeordnete Punkttupel $x_1, \ldots, x_n$, $\tilde{x}_1, \ldots, \tilde{x}_n$ sind auch

$$x_1 + t(\tilde{x}_1 - x_1) < \ldots x_n + t(\tilde{x}_n - x_n)$$

für $t \in [0, 1]$, angeordnete Punkttupel in I. Wegen

$$V(x_1 + t(\tilde{x}_1 - x_1), \ldots x_n + t(\tilde{x}_n - x_n)) \neq 0$$

kann V das Vorzeichen zwischen $t = 0$ und $t = 1$ nicht ändern. Zum Beweis der zweiten Behauptung schreiben wir

$$v = a_1 v_1 + \ldots a_n v_n.$$

Es folgt

$$\begin{pmatrix} v_1(x_0) & \ldots & v_n(x_0) & v(x_0) \\ \vdots & & \vdots & \vdots \\ v_1(x_n) & \ldots & v_n(x_n) & v(x_n) \end{pmatrix} \cdot \begin{pmatrix} a_1 \\ \vdots \\ a_n \\ -1 \end{pmatrix} = 0.$$

Es folgt, dass die Determinante der Matrix gleich 0 ist. Man erhält durch Entwickeln nach der letzten Spalte mit Hilfe der Vorzeichenwechsel von v

$$0 = |v(x_0)| \cdot V(x_1, \ldots, x_n) + \ldots |v(x_n)| \cdot V(x_0, \ldots, x_{n-1}).$$

Dies ist also nur möglich wenn

$$v(x_0) = \ldots = v(x_n) = 0$$

ist. Es folgt $v = 0$. **q.e.d.**

3.2 Dividierte Differenzen

Die Lagrangeschen Grundpolynome sind keine effiziente Art, das Interpolationspolynom auszurechnen. Die Vandermondesche Matrix andererseits ist oft schlecht konditioniert, so dass die numerische Berechnung der Koeffizienten des Interpolationspolynoms fehl schlägt. Als Ausweg bieten sich Newtonsche dividierte Differenzen an.

3.21. Definition: Seien $x_0, \ldots, x_n$ paarweise verschiedene Punkte in $\mathbb{K}$, und f eine auf diesen Punkten definierte Funktion. Dann bezeichnen wir mit

$$[x_0, \ldots, x_n]_f$$

die $n+1$-te Newtonsche dividierte Differenz. Sie ist definiert als der höchste Koeffizient des Interpolationspolynoms an f in diesen Punkten.

3.22 Aufgabe: Bezeichne $p_{k,\ldots,k+m}(x)$ das Interpolationspolynom zu f in den Punkten $x_k, \ldots, x_{k+m}$. Zeigen Sie

$$p_{k,\ldots,k+m}(x) = \frac{p_{k+1,\ldots,k+m}(x) \cdot (x - x_k) + p_{k,\ldots,k+m-1}(x) \cdot (x_{k+m} - x)}{x_{k+m} - x_k}.$$

3.23 Satz: *Seien $x_0, \ldots, x_n$ paarweise verschiedene Punkte in $\mathbb{K}$, und f eine auf diesen Punkten definierte Funktion. Dann gilt*

$$[x_k]_f = f(x_k)$$

für $k = 0, \ldots, n$, und

$$[x_k, \ldots, x_{k+m}]_f = \frac{[x_{k+1}, \ldots, x_{k+m}]_f - [x_k, \ldots, x_{k+m-1}]_f}{x_{k+m} - x_k}$$

für $0 \leq k < k + m \leq n$. Außerdem gilt für das Interpolationspolynom an f in diesen Punkten

$$\begin{aligned} p(x) = & [x_0]_f + \\ & [x_0, x_1]_f \cdot (x - x_0) + \\ & [x_0, x_1, x_2]_f \cdot (x - x_0) \cdot (x - x_1) + \\ & \ldots + \\ & [x_0, \ldots, x_n]_f \cdot (x - x_0) \cdot \ldots \cdot (x - x_{n-1}). \end{aligned}$$

Beweis: Per Induktion nach der Anzahl der Punkte in der dividierten Differenz. Für einen Punkt ist die Behauptung klar. Angenommen, wir haben $m+1$ Punkte $x_k, \ldots, x_{k+m}$. Dann folgt die Gleichung für die dividierten Differenzen sofort aus der obigen Aufgabe und der Induktionsvoraussetzung.

Jedes Polynom n-ten Grades hat eine Darstellung

$$p(x) = c_0 + c_1(x - x_0) + \ldots + c_n(x - x_0)\ldots(x - x_{n-1}),$$

da die Polynome auf der rechten Seite eine Basis von $\mathcal{P}_n$ bilden. Es folgt aus der Definition der dividierten Differenzen

$$c_n = [x_0, \ldots, x_n]_f,$$

wenn p das Interpolationspolynom ist. Andererseits ist dann

$$\tilde{p}(x) = c_0 + c_1(x - x_0) + \ldots + c_{n-1}(x - x_0)\ldots(x - x_{n-2}),$$

offenbar das Interpolationspolynom $n-1$-ten Grades in $x_0, \ldots, x_{n-1}$. Rekursiv folgt die Behauptung. **q.e.d.**

Die Berechnung der dividierten Differenzen kann mit Hilfe des Newton-Schemas

$$\begin{array}{llll}
x_0 & [x_0]_f & & \\
 & & [x_0, x_1]_f & \\
x_1 & [x_1]_f & & \\
 & & & \ddots \\
\vdots & \vdots & & & [x_0, \ldots, x_n]_f \\
 & & & \iddots \\
x_1 & [x_{n-1}]_f & & \\
 & & [x_{n-1}, x_n]_f & \\
x_n & [x_n]_f &
\end{array}$$

geschehen. Die Auswertung des Interpolationspolynom ist effektiv mit dem Horner-Schema

$$\begin{aligned}
y_n &= [x_0, \ldots, x_n]_f \\
y_{n-1} &= y_n \cdot (x - x_{n-1}) + [x_0, \ldots, x_{n-1}]_f \\
&\vdots \\
y_0 &= y_1 \cdot (x - x_0) + [x_0]_f
\end{aligned}$$

möglich. Dieses Schema beruht auf der Klammerung

$$\begin{aligned}
p(x) = [x_0]_f & \\
&+ (x - x_0) \cdot ([x_0, x_1]_f + (x - x_1) \cdot (\ldots + (x - x_{n-1}) \cdot [x_0, \ldots, x_n]_f) \ldots).
\end{aligned}$$

Die Klammern werden von innen nach außen ausgewertet. Es gilt dann $f(x) = y_0$.

3.24. Beispiel: Die Punkte seien

$$x_0 = 0, \quad x_1 = 1, \quad x_2 = 2,$$

und die Werte $y_k = f(x_k)$

$$y_0 = 1, \quad y_1 = 0, \quad y_2 = 2.$$

Dann lautet das Newtonsche Schema

$$\begin{array}{lll} 0 & 1 & \\ & & \frac{0-1}{1-0} = -1 \\ 1 & 0 & \qquad\qquad \frac{2-(-1)}{2-0} = \frac{3}{2} \\ & & \frac{2-0}{2-1} = 2 \\ 2 & 2 & \end{array}$$

Es folgt

$$p(x) = 1 + (-1)(x-0) + \frac{3}{2}(x-0)(x-1).$$

Wir können das Newtonschema in der Programmiersprache von EMT programmieren.

```
>function dd (x,y) ...
$  n=length(y);
$  d=y;
$  for i=1 to n-1;
$    for j=n to i+1 step -1;
$      d[j]=(d[j]-d[j-1])/(x[j]-x[j-i]);
$    end;
$  end;
$  return d
$endfunction
```

Dazu überschreiben wir die Werte in y mit den dividierten Differenzen wie folgt

$$\begin{pmatrix} y_0 = [x_0]_f \\ \vdots \\ y_n = [x_n]_f \end{pmatrix} \mapsto \begin{pmatrix} [x_0]_f \\ [x_0, x_1]_f \\ \vdots \\ [x_{n-1}, x_n]_f \end{pmatrix} \mapsto$$

$$\begin{pmatrix} [x_0]_f \\ [x_0, x_1]_f \\ [x_0, x_1, x_2]_f \\ \vdots \\ [x_{n-2}, x_{n-1}, x_n]_f \end{pmatrix} \mapsto \ldots \mapsto \begin{pmatrix} [x_0]_f \\ [x_0, x_1]_f \\ [x_0, x_1, x_2]_f \\ \vdots \\ [x_0, \ldots, x_n]_f \end{pmatrix}.$$

Zu beachten ist dabei, dass in EMT die Indizes mit 1 beginnen. Außerdem muss man von unten nach oben rechnen, und die korrekte Formel für die Indizes Zähler und Nenner der dividierten Differenz anwenden.

Im unserem Beispiel erhalten wir

```
>x=[0,1,2]; y=[1,0,2];
>d=dd(x,y)
 [ 1  -1  1.5 ]
```

Das Hornerschema programmiert sich dann wie folgt.

```
>function ddeval (x,dd,x0) ...
$  n=cols(dd);
$  y=dd[n];
$  for i=n-1 to 1 step -1;
$    y=y*(x0-x[i])+dd[i];
$  end
$  return y
$endfunction
```

Im Beispiel ergibt sich

```
>ddeval(x,d,4)
 15
>function f(t) := ddeval(x,d,t)
>plot2d("f",0,2); plot2d(x,y,>points,>add); insimg;
```

Das letzte Kommando plottet das Interpolationspolynom. Selbstverständlich sind diese Funktionen in EMT schon vorhanden.

```
>d=divdif(x,y)
 [ 1  -1  1.5 ]
>divdifeval(x,d,4)
 15
```

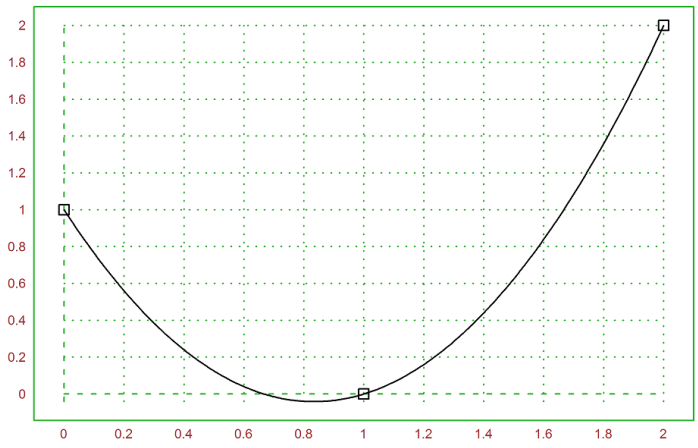

Abbildung 3.2: Interpolationsbeispiel

3.25 Aufgabe: Plotten Sie in EMT das Interpolationspolynom zu

$$f(x) = e^x$$

in den Punkten

$$x_k = \frac{1}{k} \qquad \text{für } k = 0, \ldots, n,$$

und $n = 1, 2, 3, 4, 5$, sowie die Differenz zwischen f und dem Interpolationspolynom auf $[0, 1]$.

3.26 Aufgabe: Zeigen Sie

$$[x_0, \ldots, x_n]_f = \sum_{k=0}^{n} \frac{f(x_k)}{\omega'(x_k)}$$

mit $\omega(x) = (x - x_0) \dots (x - x_n)$, indem Sie den höchsten Koeffizienten der Lagrange-Darstellung betrachten.

3.3 Fehlerabschätzung

Leider konvergiert eine Folge von Interpolationspolynomen nicht immer gegen die Funktion. Es gilt allerdings eine Fehlerabschätzung, die wir im folgenden herleiten. Diese Fehlerabschätzung liefert in vielen Fällen die gleichmäßige Konvergenz.

3.27 Satz: *Sei $f : I \to \mathbb{R}$ stetig und im Innern von I n-mal differenzierbar, $I \subseteq \mathbb{R}$ ein reelles Intervall, sowie für $n \geq 1$*

$$x_0, \dots, x_n \in I$$

paarweise verschieden. Dann gilt

$$[x_0, \dots, x_n]_f = \frac{f^{(n)}(\xi)}{n!}$$

mit einer Stelle ξ im Innern von I, die von den Punkten und von f abhängt.

Beweis: Sei p_{n-1} das Interpolationspolynom in $x_0, \dots, x_{n-1}$ und p_n das Interpolationspolynom in $x_0, \dots, x_n$. Dann gilt nach dem Satz über dividierten Differenzen

$$p_n(x) = p_{n-1}(x) + [x_0, \dots, x_n]_f \omega_{n-1}(x)$$

mit

$$\omega_{n-1}(x) = (x - x_0) \cdot \dots \cdot (x - x_{n-1}).$$

Wir betrachten die Funktion

$$h(x) = f(x) - p_n(x) = f(x) - p_{n-1}(x) - [x_0, \dots, x_n]_f \cdot \omega_{n-1}(x).$$

Diese Funktion hat $n + 1$ Nullstellen. Nach dem Satz von Rolle hat die Ableitung noch n Nullstellen, etc. Also hat die n-te Ableitung noch eine Nullstelle ξ. Wir erhalten

$$0 = h^{(n)}(\xi) = f^{(n)}(\xi) - [x_0, \dots, x_n]_f \cdot n!$$

Daraus folgt die Behauptung. **q.e.d.**

3.28 Satz: *Sei $f : I \to \mathbb{R}$ stetig und im Innern von I $n+1$-mal differenzierbar, $I \subseteq \mathbb{R}$ ein reelles Intervall, sowie für $n \geq 1$*

$$x_0, \dots, x_n \in I$$

Sei p_n das Interpolationspolynom in diesen Punkten. Dann gilt

$$f(x) - p_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \cdot \omega_n(x)$$

mit

$$\omega_n(x) = (x - x_0) \cdot \dots \cdot (x - x_n),$$

und einem ξ aus dem Innern von I, das von f, den Interpolationspunkten und $x \in I$ abhängt.

Beweis: Falls x mit einem der Interpolationspunkte übereinstimmt, so gilt der Satz, weil auf beiden Seiten 0 steht. Ansonsten bezeichnen wir mit p_{n+1} das Interpolationspolynom in $x_0, \ldots, x_n, x$. Dann gilt, analog zum obigen Beweis,

$$p_{n+1}(t) = p_n(t) + [x_0, \ldots, x_n, x]_f \omega_n(t).$$

Also

$$f(x) - p_n(x) = [x_0, \ldots, x_n, x]_f \cdot \omega_n(x).$$

Die Behauptung folgt unter Verwendung des obigen Satzes für den Polynomgrad $n+1$. **q.e.d.**

Die Sätze gelten auch für $n = 0$ mit einem $\xi \in I$. Der zweite Satz ist dann der Mittelwertsatz

$$f(x) - f(x_0) = f'(\xi) \cdot (x - x_0).$$

3.29 Aufgabe: Sei $f :]a, b[\to \mathbb{R}$ zweimal stetig differenzierbar. f heißt konvex, wenn f in jedem Teilintervall $]c, d[\subsetneq]a, b[$ größer oder gleich der Sekante $g_{c,d}$ durch $(c, f(c))$ und $(d, f(d))$ ist. Zeigen Sie, dass f genau dann konvex ist, wenn

$$f''(x) \geq 0$$

für alle $x \in]a, b[$ gilt.

3.30. Definition: Wir definieren die Supremums-Norm auf dem Vektorraum $C(X)$, X kompakt, durch

$$\|f\|_X = \max_{x \in X} |f(x)|.$$

Falls $p_n \to f$ in dieser Norm konvergiert, dann sagt man, p_n konvergiere gleichmäßig auf X gegen f.

3.31 Aufgabe: Zeigen Sie, dass diese Norm tatsächlich eine Norm auf dem Vektorraum $C(X)$ ist.

3.32. Beispiel: Für Funktionen wie e^{ax}, $\sin(ax)$ etc. kann man die Abschätzung

$$\sup_{x \in I} |f^{(n)}(x)| \leq a^n$$

erhalten. Eine grobe Abschätzung für ω_n ist

$$\|\omega_n\|_I \leq \operatorname{diam}(I)^{n+1}.$$

Man hat dann also

$$\|f(x) - p_n(x)\|_I \leq \frac{c^{n+1}}{(n+1)!} \to 0.$$

Die Folge der Interpolationspolynome geht also gleichmäßig gegen 0.

3.33. Beispiel: Wir interpolieren e^{2x} auf dem Intervall $[-1, 1]$ mit Polynomen vom Grad n in äquidistant verteilten Punkten

$$-1 = x_0 < x_0 + h < \ldots < x_0 + nh = x_n = 1.$$

und plotten die Fehlerfunktion in EMT. Dabei wird der Plotbereich ausgegeben, an dem man den maximalen Fehler ablesen kann. Bei $n = 10$ erhält man eine Fehler in der Größenordnung von $5 \cdot 10^{-7}$

```
>n=10; xp=linspace(-1,1,n)
 [ -1  -0.8  -0.6  -0.4  -0.2  0  0.2  0.4  0.6  0.8  1 ]
>yp=exp(2*xp)
 [ 0.135335283237  0.201896517995  0.301194211912  0.449328964117
 0.670320046036  1  1.49182469764  2.22554092849  3.32011692274
 4.9530324244  7.38905609893 ]
>dd=divdif(xp,yp);
>plot2d("divdifeval(xp,dd,x)-exp(2*x)",-1,1), h=max(-%[3],%[4])
 [ -1  1  -4.00067848849e-007  5.47276611584e-007 ]
 5.47276611584e-007
```

3.34 Aufgabe: (a) Gegeben sei die Funktion

$$f(x) = \frac{1}{a-x}$$

für $u \notin I$. Weisen Sie mit Hilfe der Fehlerabschätzung

$$\|f(x) - p_n(x)\|_I \le c \cdot \frac{d^n}{f^n}$$

nach mit einer von n unabhängigen Konstanten $c > 0$, wobei

$$d = \operatorname{diam}(I) = \sup\{|x-y| : x, y \in I\},$$

und

$$f = \operatorname{dist}(a, I) = \inf\{|a-x| : x \in I\}.$$

Geben Sie für $I = [-1, 1]$ Bedingungen für die Lage von $a \in \mathbb{R}$ an, so dass die Interpolation gleichmäßig konvergiert.

(b) Zeigen Sie, dass das Interpolationspolynom in $n+1$ Punkten $x_0, \ldots, x_{n+1} \in \mathbb{K}$ gleich

$$p_n(x) = \frac{\omega_n(x) - \omega_n(a)}{(x-a)\cdot\omega_n(a)},$$

ist, mit der Fehlerfunktion

$$f(x) - p_n(x) = \frac{\omega_n(x)}{(x-a)\cdot\omega_n(a)}.$$

Lösen Sie Aufgabenteil (a) mit dieser Formel.

Das Beispiel dieser Aufgabe zeigt, dass das Wachstum von ω_n entscheidend ist. Die Funktion muss klein auf I sein, und groß in a.

3.35 Satz: *Sei f in 0 in eine Potenzreihe mit Konvergenzradius $r > 3$ entwickelbar, und p_n eine Folge von Interpolationspolynomen mit Interpolationspunkten*

$$x_{0,n}, \ldots, x_{n,n} \in [-1, 1].$$

Dann konvergiert die Folge gleichmäßig gegen f auf $[-1, 1]$.

Beweis: Wir zeigen

$$\sup_{x\in[-1,1]} |f^{(n)}(x)| \le c \cdot \frac{n!}{(\rho-1)^n}$$

für jedes $3 < \rho < r$, wobei die Konstante $c > 0$ von ρ, aber nicht von n abhängt. Daraus folgt dann

$$\sup_{x\in[-1,1]} |f(x) - p_n(x)| \le c \cdot \frac{2^{n+1}}{(\rho-1)^{n+1}} \to 0.$$

wie in den obigen Beispielen und Aufgaben. Die Abschätzung der Ableitung kann mit der Cauchyschen Integraldarstellung für die Ableitung

$$f^{(n)}(x) = \frac{n!}{2\pi i} \oint_\gamma \frac{f(z)}{(x-z)^{n+1}}\, dz$$

bewiesen werden, wobei der Weg γ einmal um den Kreis mit Radius ρ läuft. Alternativ kann man die Potenzreihe wie folgt abschätzen. Die Koeffizienten der Potenzreihe seien mit a_k bezeichnet. Wegen

$$\limsup_n \sqrt[n]{|a_n|} = \frac{1}{r}$$

gibt es eine Konstante $c > 0$ mit

$$|a_k| \leq \frac{c}{\rho^k} \qquad \text{für alle } k \in \mathbb{N}$$

Für $|x| \leq 1$ folgt demnach

$$\begin{aligned}
|f^{(n)}(x)| &= \left| \sum_{k=0}^{\infty} a_k \cdot k \cdot \ldots \cdot (k-n+1) x^{n-k} \right| \\
&\leq c \sum_{k=0}^{\infty} \frac{k \cdot \ldots \cdot (k-n+1)}{\rho^k} \\
&= \frac{c}{\rho^n} \sum_{k=0}^{\infty} k \cdot \ldots \cdot (k-n+1) \cdot \left(\frac{1}{\rho}\right)^{n-k} \\
&= \frac{c}{\rho^n} \cdot g^{(k)}\left(\frac{1}{\rho}\right),
\end{aligned}$$

mit

$$g(x) = \sum_{k=0}^{n} x^k = \frac{1}{1-x}.$$

Es gilt

$$g^{(n)}(x) = \frac{n!}{(1-x)^{n+1}}.$$

Wir erhalten

$$|f^{(n)}(x)| \leq \frac{c}{\rho^n} \cdot \frac{n!}{(1-1/\rho)^{n+1}} = \frac{c\rho n!}{(\rho-1)^{n+1}}$$

wie behauptet. **q.e.d.**

Der obige Satz bleibt auch bei Interpolation auf dem Einheitskreis richtig. Insgesamt ergibt sich, dass die Interpolation auf

$$D_r = \{z \in \mathbb{C} : |z| \leq r\}$$

gleichmäßig konvergiert, wenn die Funktion f über D_{3r} hinaus analytisch ist.

3.4 Hermite-Interpolation

Es liegt nahe, zur Interpolation auch die Ableitungen von f heranzuziehen. Das Interpolationspolynom soll dann auch in gewissen Ableitungen mit f übereinstimmen.

3.36. Definition: Seien $x_0, \ldots, x_n \in \mathbb{K}$ Interpolationspunkte, die nicht notwendigerweise paarweise verschieden sind, und f in diesen Punkten genügend oft differenzierbar. Dann lautet das Hermitesche Interpolationsproblem

$$p(x_k) = f(x_k), \quad p'(x_k) = f'(x_k), \quad \ldots, \quad p^{(l_k-1)}(x_k) = f^{(l_k-1)}(x_k),$$

wobei l_k die Vielfachheit des Punktes x_k in dem Punkttupel $x_0, \ldots, x_k$ sei. Insgesamt sind dies wieder $n+1$ Bedingungen. Im komplexen Fall ist hier die komplexe Ableitung gemeint.

3.37. Beispiel: Wir suchen ein $p \in \mathcal{P}_3$ mit

$$p(-1) = 1, \quad p'(-1) = -1, \quad p(1) = 1, \quad p'(1) = 1.$$

Das Polynom

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

löst dieses Problem.

3.38 Satz: *Die Hermitesche Interpolationsaufgabe ist immer eindeutig lösbar.*

Beweis: Wir betrachten die Interpolation der Funktion $f(z) = 0$. Es entsteht ein Polynom mit $n+1$ Nullstellen, einschließlich Vielfachheit gezählt. Es folgt $p = 0$. Die Abbildung $\phi : \mathcal{P}_n \to \mathbb{R}^{n+1}$, die die Auswertung des Polynoms einschließlich der Ableitungen beschreibt, hat also den Nullraum als Kern, und ist deswegen bijektiv, also insbesondere surjektiv. Das ist die Behauptung. **q.e.d.**

3.39. Definition: Wir behalten die Definition der dividierten Differenzen als höchstem Koeffizienten der Interpolationspolynome bei.

3.40 Satz: *Die Darstellung des Interpolationspolynoms aus Satz 23 gilt auch für Hermite-Interpolation.*

3.41 Aufgabe: Beweisen Sie diesen Satz analog zum Beweis für paarweise verschiedene Punkte. Begründen Sie genau, warum $\tilde{p}$ das Hermitesche Interpolationspolynom für die Punkte $x_0, \ldots, x_{n-1}$ löst.

Man kann die Hermite-Interpolation als Grenzfall der Interpolation mit verschiedenen Punkten auffassen, wobei jeweils l_k Punkte zu einem konvergieren. Wegen

$$[x_0, \ldots, x_{k-1}]_f = \frac{f^{(k)}(\xi)}{k!}$$

konvergieren dabei die dividierten Differenzen. Es folgt, wenn $x_j \to a$ für $j = 0, \ldots, k-1$

$$[a, \ldots, a]_f = \frac{f^{(k)}(a)}{k!}.$$

Bei der Berechnung des Newton-Schemas muss man also diese Werte an den entsprechenden Stellen einsetzen.

3.42 Aufgabe: Zeigen Sie mit Hilfe dieser Grenzwertbildung, dass auch die Rekursionsformel für dividierte Differenzen aus Satz 23 richtig bleibt, sofern nur der Nenner ungleich 0 ist.

3.43. Beispiel: Um die dividierten Differenzen der Hermite-Interpolation zu berechnen, modifizieren wir die Funktion dd wie folgt.

```
>function dd (x,df) ...
$  n=length(x);
$  d=zeros(1,n);
$  for i=1 to n;
$    d[i]=df(x[i],0);
$  end;
$  for i=1 to n-1;
$    for j=n to i+1 step -1;
$      if x[j]==x[j-i] then d[j]=df(x[j],i)/i!;
$      else d[j]=(d[j]-d[j-1])/(x[j]-x[j-i]);
$      endif;
$    end;
$  end;
$  return d
$endfunction
```

Die Funktion benötigt nun eine weitere Funktion `df`, die in der Lage ist, eine Funktion und ihre Ableitungen zu berechnen. Mit Hilfe von symbolischen Ausdrücken, die schon bei der Eingabe der Funktion berechnet werden, ist das etwa folgendermaßen möglich.

```
>function f(x) &= exp(x^2);
>function map df(x,n) ...
$  if n==0 then return f(x);
$  elseif n==1 then return &:diff(f(x),x);
$  elseif n==2 then return &:diff(f(x),x,2);
$  end;
$endfunction
```

Schließlich können wir ein Beispiel berechnen, wobei wir die Funktion `divdifeval` von EMT wie von der gewöhnlichen Interpolation gewohnt verwenden können.

```
>xn=[-1,-0.5,-0.5,0,0,0.5,0.5,1]
 [ -1  -0.5  -0.5  0  0  0.5  0.5  1 ]
>d=dd(xn,"df")
 [ 2.71828182846  -2.86851282354  3.16897481371  -1.73702564709
 1.14533064734  -0.369090431727  0.246060287818  0 ]
>plot2d("exp(x^2)-divdifeval(xn,d,x)",-1,1);
```

3.44 Satz: *Die Abschätzung für den Interpolationsfehler aus Satz 28 bleibt für Hermite-Interpolation gültig.*

3.45 Aufgabe: Beweisen Sie den Satz auf die gleiche Weise wie den Fall für paarweise verschiedene

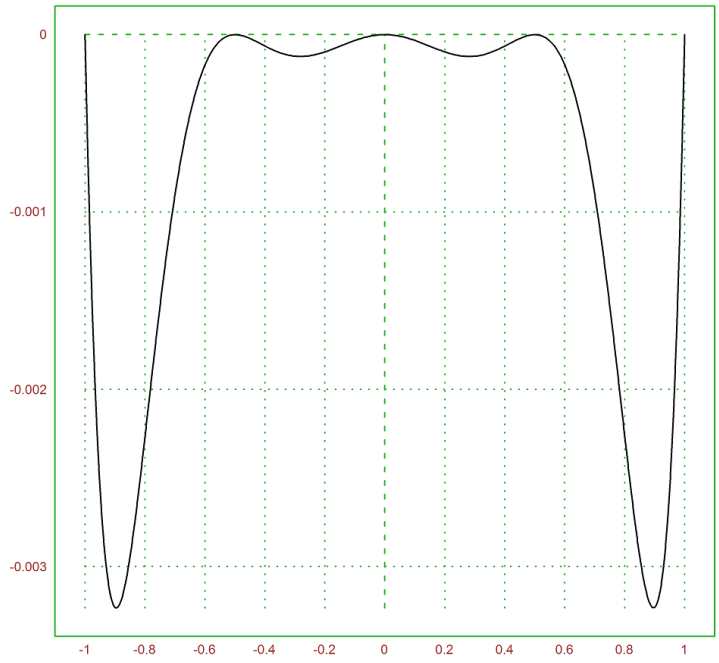

Abbildung 3.3: Fehler der Hermite-Interpolation

Punkte. Beachten Sie bei der Anwendung des Satzes von Rolle die Vielfachheiten.

3.46. Beispiel: Die Hermite-Interpolation in einem Punkt ist das Taylor-Polynom. Es gilt dann

$$\begin{aligned} p(x) &= [x_0]_f + [x_0, x_0]_f\,(x - x_0) + \ldots + [x_0, \ldots, x_0]_f\,(x - x_0)^n \\ &= f(x_0) + f'(x_0)\,(x - x_0) + \ldots + \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n. \end{aligned}$$

Die Fehlerformel wird zur bekannten Formel

$$R(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}(x - x_0)^{n+1}.$$

3.47 Aufgabe: Schreiben Sie das Interpolationspolynom auf, das $\sin(x)$ im Punkt 0 genau 6-fach und im Punkt π einmal interpoliert. Was ist also $[0, 0, 0, 0, 0, 0, \pi]_{\sin}$?

3.48 Aufgabe: Sei $f :]a, b[\to \mathbb{R}$ zweimal stetig differenzierbar, und

$$f''(x) \geq 0$$

für alle $x \in]a, b[$, sowie $T_c(x)$ die Tangente an f in einem Punkt $c \in]a, b[$. Zeigen Sie

$$f(x) \geq T_c(x)$$

für alle $x \in]a, b[$.

3.49 Aufgabe: Zeigen Sie, dass für eine genügend oft differenzierbare Funktion das Interpolationspolynom gleichmäßig auf kompakten Intervallen konvergiert, wenn alle $n + 1$ Interpolationspunkte gegen feste Interpolationspunkte konvergieren.

3.5 Trigonometrische Polynome

3.50. Definition: Wir bezeichnen mit $\mathcal{T}_n$ den Raum der auf $[0, 2\pi[$ definierten Funktionen

$$p(t) = a_0 + \sum_{k=1}^{n} (a_k \cos(kt) + b_k \sin(kt)) .$$

Diese Funktionen heißen trigonometrische Polynome vom Grad kleiner oder gleich n. Unendliche Reihen der Form

$$p(t) = a_0 + \sum_{k=1}^{\infty} (a_k \cos(kt) + b_k \sin(kt))$$

bezeichnet mal als Fourier-Reihen.

3.51 Satz: *Die Funktion $p : [0, 2\pi[\to \mathbb{R}$ ist genau dann ein trigonometrisches Polynom vom Grad kleiner oder gleich n, wenn sie eine Darstellung der Form*

$$p(t) = \sum_{k=-n}^{n} \alpha_k z^k$$

mit $z = e^{it}$ und

$$\alpha_k = \overline{\alpha_{-k}} \qquad \text{für all } k$$

hat.

Beweis: Man berechnet

$$\begin{aligned} a_k \cos(kt) + b_k \sin(kt) &= a_k \frac{z^{kt} + z^{-kt}}{2} + b_k \frac{z^{kt} - z^{-kt}}{2i} \\ &= \frac{1}{2}(a_k - ib_k) z^k + \frac{1}{2}(a_k + ib_k) z^{-k}. \end{aligned}$$

mit $z = e^{it}$. Folglich lässt sich jedes trigonometrische Polynom in der verlangten Form darstellen. Umgekehrt lässt sich diese Form mit Hilfe dieser Rechnung in ein trigonometrisches Polynom umwandeln. Man beachte dabei, dass aus $\alpha_0 = \overline{\alpha_0}$ auch $\alpha_0 \in \mathbb{R}$ folgt. **q.e.d.**

3.52 Aufgabe: Zeigen Sie: Genau dann ist

$$\sum_{k=-n}^{n} \alpha_k z^k$$

die komplexe Darstellung eines trigonometrischen Polynoms, wenn es auf dem Einheitskreis nur reelle Werte annimmt.

3.53 Aufgabe: Man stelle das trigonometrische Polynom

$$p(t) = 2 + 3\sin(t) + 4\cos(t) + 5\sin(2t)$$

in komplexer Form dar.

3.54 Satz: *$\mathcal{T}_n$ ist ein Haarscher Unterraum von $C[0, 2\pi[$ der Dimension*

$$\dim \mathcal{T}_n = 2n + 1.$$

Beweis: Wir nehmen an, dass $p \in \mathcal{T}_n$ mindestens $2n+1$ Nullstellen in $[0, 2\pi[$ hat, und stellen p in komplexer Form wie im Satz dar. Dann hat das Polynom $2n+1$-ten Grades

$$q(z) = z^n \sum_{k=-n}^{n} \alpha_k z^k$$

mindestens $2n+1$ Nullstellen auf dem Einheitskreis. Es folgt, dass alle $\alpha_k = 0$ sind. Rechnet man die Darstellung ins trigonometrische zurück, so folgt, dass alle Koeffizienten von p gleich 0 sind. **q.e.d.**

3.55 Aufgabe: Man zeige, dass für $p \in \mathcal{T}_n$ und $a \in \mathbb{R}$ auch die Funktion

$$\tilde{p}(t) = p(a+t)$$

in $\mathcal{T}_n$ ist. Dazu verwendet man entweder trigonometrische Identitäten, oder die komplexe Darstellung. Man folgere daraus, dass die trigonometrischen Funktionen vom Grade kleiner oder gleich n auch auf jedem Intervall $[a, a+2\pi[$ einen Haarschen Unterraum der Dimension $2n+1$ bilden.

3.56 Aufgabe: Zeigen Sie, dass die Funktionen

$$\frac{1}{\sqrt{2}}, \cos(x), \sin(x), \ldots, \cos(nx), \sin(nx)$$

orthonormal bezüglich des Skalarprodukts

$$\langle f, g \rangle = \frac{1}{\pi} \int_{-\pi}^{\pi} f(t) g(t)\, dt$$

sind.

3.57. Definition: Der Raum $\mathcal{C}_n$ der Funktionen

$$p(t) = a_0 + \sum_{k=1}^{n} a_k \cos(kt)$$

definiert auf $[0, \pi]$ heißt Raum der Kosinus-Polynome vom Grad kleiner oder gleich n.

3.58 Aufgabe: Zeigen Sie, dass $p \in \mathcal{T}_n$ genau dann ein Kosinus-Polynom ist, wenn seine komplexe Darstellung nur reelle Koeffizienten hat.

3.59 Satz: *Die Abbildung*

$$\hat{\cdot} : \mathcal{P}_n \to \mathcal{C}_n$$

definiert durch $q \mapsto \hat{q}$ *mit*

$$\hat{q}(t) = q(\cos(t))$$

ist eine linear und bijektive Isometrie. D.h.

$$\|q\|_{[-1,1]} = \|\hat{q}\|_{[0,\pi]}$$

für alle $q \in \mathcal{P}_n$.

Beweis: Die Funktion $\hat{q}$ liegt in $\mathcal{P}_n$. Denn es genügt, dies für $q(x) = x^n$ nachzuweisen, und es gilt

$$\hat{q}(t) = \cos(t)^n = \frac{1}{2^n}\left(z + \frac{1}{z}\right)^n.$$

mit $z = e^{it}$, also $\hat{q} \in \mathcal{C}_n$ nach obiger Aufgabe. Die Abbildung von $q \in \mathcal{P}_n$ nach $\hat{q} \in \mathcal{C}_n$ offenbar linear, und ihr Kern ist der Nullraum. Weil die Dimension von $\mathcal{C}_n$ höchstens $n+1$ ist, muss sie also gleich $n+1$ sein. Es ist leicht nachzuprüfen, dass es sich um eine Isometrie handelt. **q.e.d.**

3.60 Satz: *$\mathcal{C}_n$ ist ein Haarscher Unterraum von $C[0,\pi]$ der Dimension $n+1$.*

Beweis: Folgt sofort mit der Isometrie zwischen dem Haarschen Raum $\mathcal{P}_n$ und $\mathcal{C}_n$. **q.e.d.**

3.6 Chebyshev-Polynome

3.61. Definition: Die für $n \in \mathbb{N}_0$ definierten Funktionen $T_n : [-1,1] \to \mathbb{R}$ mit

$$T_n(x) = \cos(n \arccos x)$$

heißen Chebyshev-Polynome.

3.62 Satz: *T_n ist in der Tat ein Polynom aus $\mathcal{P}_n$, das wir uns auf ganz $\mathbb{R}$ fortgesetzt denken. Es gilt die Rekursionsformel*

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$$

für $n \in \mathbb{N}_0$, $x \in \mathbb{R}$. Es gilt

$$T_n(x) = 2^{n-1}x^n + \ldots,$$

für $n \in \mathbb{N}$, sowie

$$\|T_n(x)\|_{[-1,1]} = 1$$

für alle $n \in \mathbb{N}_0$.

Beweis: Mit $p_n(t) = \cos(nt)$ haben wir offenbar

$$\hat{T}_n(t) = p_n(t).$$

Also ist T_n ein Urbild unter der Abbildung $\hat{\cdot}$, und daher in $\mathcal{P}_n$. Es gilt

$$\begin{aligned}\cos((n+1)t) &= \cos(nt)\cos(t) - \sin(nt)\sin(t),\\ \cos((n-1)t) &= \cos(nt)\cos(t) + \sin(nt)\sin(t).\end{aligned}$$

Durch Addition folgt mit $t = \arccos(x)$

$$T_{n+1}(x) + T_{n-1}(x) = 2xT_n(x).$$

Wir haben außerdem

$$T_0(x) = 1, \quad T_1(x) = x.$$

Also folgt die Behauptung über den höchsten Koeffizienten induktiv. Die Behauptung über die Supremums-Norm folgt aus der Isometrie der Abbildung $\hat{}$. **q.e.d.**

Man hat aufgrund der Rekursionsformel weiter

$$T_2(x) = 2x^2 - 1,$$
$$T_3(x) = 4x^3 - 3x,$$
$$T_4(x) = 8x^4 - 8x^2 + 1.$$

In Maxima muss dazu ein Paket geladen werden. In EMT gibt es eine numerische Funktion, die die Koeffizienten berechnet, sowie eine numerische Funktion, die die Werte direkt berechnet.

```
>&load(orthopoly);
>&chebyshev_t(5,x)|expand

                              5       3
                          16 x  - 20 x  + 5 x

>chebpoly(5)
 [ 0  5  0  -20  0  16 ]
>n=1:5; plot2d("cheb(x,n')",-1,1);
```

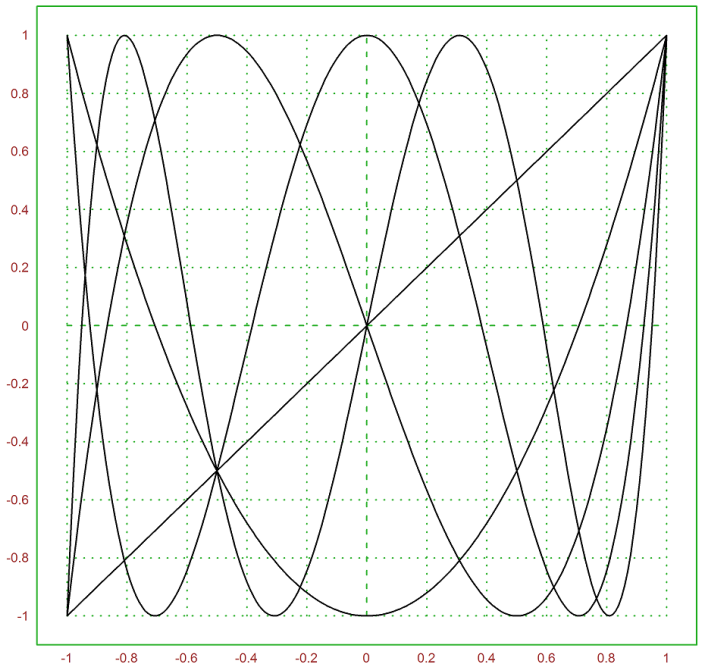

Abbildung 3.4: $T_1, \ldots, T_5$

3.63 Aufgabe: Zeigen Sie mit Hilfe der Rekursionsformel

$$T_n'(1) = n^2$$

für alle $n \in \mathbb{N}_0$.

3.64 Aufgabe: Zeigen Sie dass es zu T_n $n+1$ Punkte

$$-1 = x_{0,n} < x_1 < \ldots < x_{n,n} = 1$$

hat mit abwechselnden Werten ± 1, so dass gilt

$$T(x_{k,n}) = (-1)^{n-k},$$

und zeigen Sie, dass T_n sonst nirgends die Werte ± 1 annimmt.

3.65 Aufgabe: Zeigen Sie mit Hilfe von Satz 20, dass

$$\omega_n(x) = \frac{1}{2^{n-1}} T_n(x)$$

dasjenige Polynom $p_n \in \mathcal{P}_n$ mit höchstem Koeffizient 1 ist, dass

$$\max_{x \in [-1,1]} |p_n(x)|$$

minimiert.

3.66 Satz: *Sei $p_n \in \mathcal{P}_n$ beliebig mit*

$$\|p_n\|_{[-1,1]} \leq \|T_n(x)\|_{[-1,1]} = 1.$$

Dann gilt

$$|p_n(x)| \leq |T_n(x)| \qquad \text{für alle } |x| > 1$$

T_n wächst also maximal unter allen Polynomen, die auf $[-1,1]$ durch 1 beschränkt sind.

Beweis: Seien $x_0, \ldots, x_n$ die Extremalstellen von T_n in $[-1,1]$. Angenommen $|p_n(x)| > |T_n(x)|$ für $|x| > 1$. Wir können $x > 0$ und

$$p_n(x) > T_n(x)$$

annehmen. Dann hat $h_n = p_n - T_n$ die Vorzeichenwechsel

$$h(x) > 0, h(x_n) \leq 0, h(x_{n-1}) \geq 0, \ldots$$

von rechts nach links gezählt. Wegen $h_n \in \mathcal{P}_n$, folgt $h_n = 0$ aus Satz 20, was einen Widerspruch zu $h_n(x) > 0$ ergibt. **q.e.d.**

3.67 Aufgabe: Zeigen Sie, dass

$$\frac{1}{\sqrt{2}}, T_1, \ldots, T_n$$

eine Orthonormalbasis von $\mathcal{P}_n$ bezüglich des Skalarprodukts

$$\langle f, g \rangle = \frac{2}{\pi} \int_{-1}^{1} \frac{f(x)g(x)}{\sqrt{1-x^2}}\, dx$$

ist. In der Tat gilt

$$\langle f, g \rangle = \langle \hat{f}, \hat{g} \rangle$$

für alle $f, g \in C[-1,1]$, für die diese Integrale existieren, wobei auf der linken Seite das eben definierte Skalarprodukt auf $[-1,1]$ steht, und auf der rechten Seite das Skalarprodukt aus Aufgabe 56.

3.68 Satz: *Es gilt*

$$T_n\left(\frac{1}{2}\left(z + \frac{1}{z}\right)\right) = \frac{1}{2}\left(z^n + \frac{1}{z^n}\right).$$

für alle $z \in \mathbb{C}$, $z \neq 0$, $n \in \mathbb{N}_0$.

Beweis: Für $z = e^{it}$ gilt die Behauptung wegen

$$\cos(nt) = \frac{1}{2}\left(z^n + \frac{1}{z^n}\right)$$

für alle $n \in \mathbb{N}$. Die beiden Funktionen rechts und links stimmen daher auf dem Rand des Einheitskreises überein. Sie sind in $\mathbb{C} \setminus \{0\}$ analytisch. Aus dem Identitätssatz für analytische Funktionen folgt die Behauptung. **q.e.d.**

3.69 Aufgabe: Zeigen Sie für $|x| \geq 1$

$$T_n(x) = \frac{1}{2}\left(\left(x \pm \sqrt{x^2-1}\right)^n + \frac{1}{\left(x \pm \sqrt{x^2-1}\right)^n}\right).$$

Wie muss man das Vorzeichen wählen?

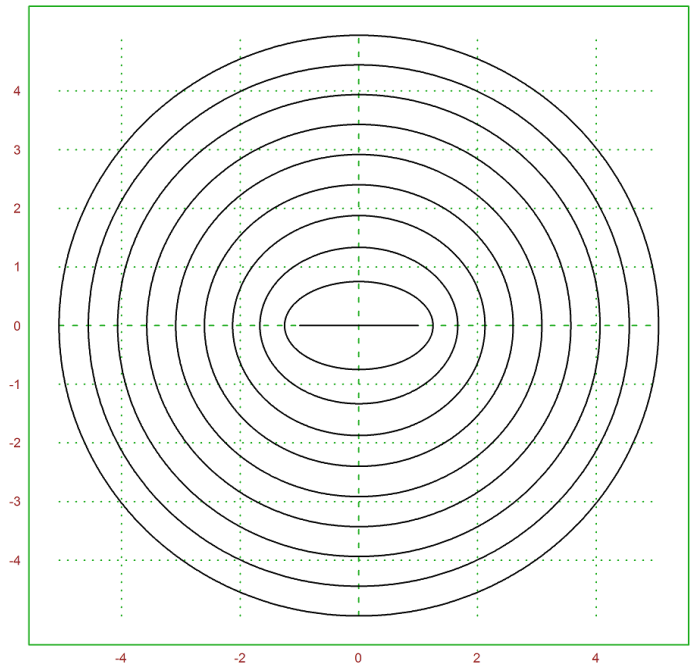

Abbildung 3.5: E_ρ für $\rho = 1, \ldots, 10$

3.70 Aufgabe: Sei $a \in \mathbb{C} \setminus [-1, 1]$. Dann gibt es eine Ellipse E_ρ mit Brennpunkten -1 und 1 auf der a liegt. In der Tat gilt

$$E_\rho = \{\frac{1}{2}\left(z + \frac{1}{z}\right) : |z| = \rho\}.$$

Berechnen Sie die Summe der Halbachsen ρ dieser Ellipse und weisen Sie nach, dass für

$$\omega_n = \frac{1}{2^n} T_{n+1}$$

gilt

$$\frac{\|\omega_n\|_{[-1,1]}}{|\omega_n(a)|} \leq \frac{C}{\rho^n} \to 0$$

wegen $\rho > 1$. Folgern Sie dass für alle $a \neq [-1,1]$ die Folge der Interpolationspolynome an die Funktion

$$f(x) = \frac{1}{x-a}$$

in den Nullstellen der Chebyshev-Polynome gleichmäßig auf $[-1,1]$ konvergiert.

Wir zeichnen einige dieser Ellipsen mit EMT.

```
>function phi(z) &= (z+1/z)/2;
>t=linspace(0,2pi,1000); z=exp(I*t);
>rho=1:10;
>w=phi(rho'*z);
>plot2d(re(w),im(w));
```

3.71 Satz: *Sei $f : [-1,1] \to \mathbb{R}$ analytisch. Das heißt, f ist in jedem Punkt $x \in [-1,1]$ in eine Potenzreihe entwickelbar. Dann konvergiert die Folge der Interpolationspolynome an f in den Nullstellen des Chebyshev-Polynoms gleichmäßig gegen f. In der Tat existiert ein $\rho > 1$, so dass*

$$\|f - p_n\|_{[-1,1]} \leq \frac{C}{\rho^n}$$

für eine von n unabhängige Konstante $C > 0$.

3.72. Beispiel: Wir berechnen die Interpolationsfehler für die Funktion

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

mit äquidistanten Interpolationspunkten und mit den Nullstellen des Chebyshev-Polynoms numerisch für $n = 10$.

```
>function f(x) := 1/(1+5*x^2);
>xp=-1:2/10:1; length(xp), yp=f(xp); dd=divdif(xp,yp);
 11
>x=linspace(-1,1,1000); y=divdifeval(xp,dd,x);
>plot2d(x,y-f(x)); max(abs(y-f(x)))
 0.151534164593
>xp=cos((1:2:21)/22*pi); length(xp), yp=f(xp); dd=divdif(xp,yp);
 11
>x=linspace(-1,1,1000); y=divdifeval(xp,dd,x);
>plot2d(x,y-f(x)); max(abs(y-f(x)))
 0.00847064214693
```

Man erhält mit dem Polynomgrad 10 mit äquidistanten Punkten die Norm 0.15 und mit den Chebyshev-Punkten die Norm 0.00085.

3.7 Der Satz von Weierstraß

Unser Ziel ist, den folgenden Satz von Weierstraß (1885) zu beweisen.

3.73 Satz: *Für jedes $f \in C[a,b]$ gibt es eine Folge von Polynomen, die gleichmäßig gegen f konvergieren.*

Beweis: Wir verwenden den Originalbeweis von Weierstraß. Es genügt, den Satz für $[a,b] = [0,1]$ zu beweisen. Man definiert

$$p_n(x) = \sum_{k=0}^{n} f\left(\frac{k}{n}\right) B_{k,n}(x),$$

wobei die Bernstein-Polynome durch

$$B_{k,n}(x) = \binom{n}{k} x^k (1-x)^{n-k}.$$

definiert sind. Man zeigt nun, dass p_n gleichmäßig gegen f konvergiert. Wegen der binomischen Formel gilt

$$\sum_{k=0}^{n} B_{k,n}(x) = (x + (1-x))^n = 1.$$

Wir erhalten daher für festes $x \in [0,1]$

$$f(x) - p_n(x) = \sum_{k=0}^{n} \left(f(x) - f\left(\frac{k}{n}\right)\right) B_{k,n}(x).$$

Wir spalten diese Summe für $\delta > 0$ in zwei Teilsummen auf.

$$f(x) - p_n(x) = \sum_{|x-k/n|<\delta} \ldots + \sum_{|x-k/n|\geq\delta} \ldots$$

Da f gleichmäßig stetig ist, erhalten wir zu $\epsilon > 0$ ein $\delta > 0$, so dass die erste Summe sich wegen $B_{k,n} \geq 0$ mit

$$\left| \sum_{|x-k/n|<\delta} \ldots \right| \leq \epsilon \cdot \sum_{|x-k/n|<\delta} B_{k,n}(x) \leq \epsilon \cdot \sum_{k=0}^{n} B_{k,n}(x) = \epsilon$$

abschätzen lässt. Zur Abschätzung der zweiten Summe verwenden wir

$$\left| \sum_{|x-k/n|\geq\delta} \ldots \right| \leq 2\|f\|_{[0,1]} \cdot \sum_{|x-k/n|\geq\delta} B_{k,n}(x).$$

Zur Abschätzung der Summe auf der rechten Seite greifen wir auf bekannte Ergebnisse der Wahrscheinlichkeitstheorie zurück. Es gilt

$$\sum_{k=0}^{n} \left(x - \frac{k}{n}\right)^2 B_{k,n}(x) = \frac{x(1-x)}{n}.$$

Daraus folgt

$$\delta^2 \cdot \sum_{|x-k/n|\geq\delta} B_{k,n}(x) \leq \frac{x(1-x)}{n}.$$

Bei festem $\delta > 0$ kann man daher durch Wahl eines genügend großen n die gleichmäßige Konvergenz erhalten. **q.e.d.**

3.74 Aufgabe: Folgern Sie die im Beweis verwendete Gleichung aus Ergebnissen der Wahrscheinlichkeitstheorie. Dazu verwenden Sie n unabhängige Kopien einer binomial-verteilten Zufallsvariablen $X_1, \ldots, X_n$ mit Erwartungswert x und deren Varianz

$$\frac{x(1-x)}{n} = \mathrm{Var}\left(\frac{1}{n}\sum_{k=1}^{n} X_k\right) = \sum_{k=0}^{n}\left(x-\frac{k}{n}\right)^2 \binom{n}{k} x^k (1-x)^{n-k}.$$

Beachten Sie, dass der Mittelwert der X_k den Erwartungswert x hat, und Werte k/n, $k = 0, \ldots, n$ annimmt.

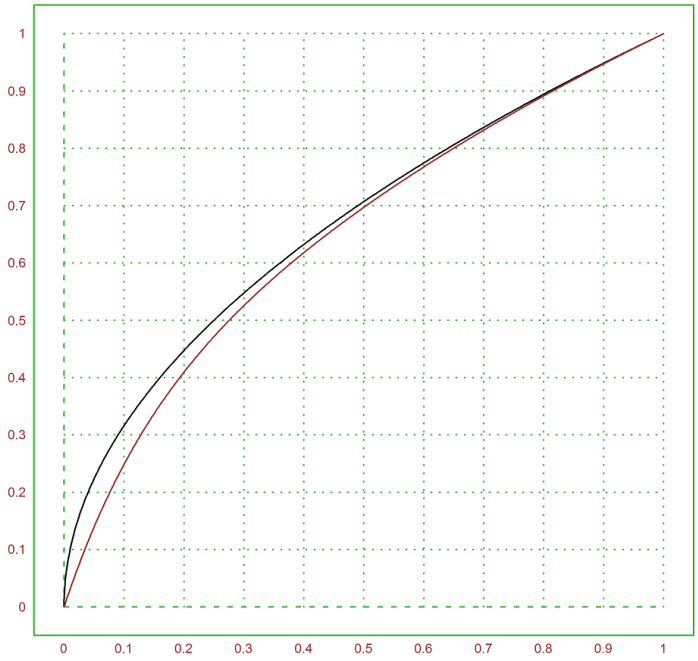

Abbildung 3.6: $\sqrt{x}$ approximiert mit Bernstein-Polynomen

3.75. Beispiel: Wir approximieren $\sqrt{x}$ auf $[0, 1]$ mit diesem Verfahren. Zur Berechnung der Bernstein-Polynome verwenden wir die Matrixsprache von EMT, und stellen eine Matrix

$$M = (B_{i,n}(t_j))_{i,j}$$

her, wobei $0 \leq t_1 < \ldots < t_m = 1$ die Punkte sind, in denen wir die Approximation auswerten wollen. Dann gilt für die Werte $s_j = (t_j)$ der Approximation

$$(f(0), f(1/n), f(2/n) \ldots, f(1)) \cdot M = (s_1, \ldots, s_m).$$

Die Approximation ist keineswegs optimal.

```
>function f(x) := sqrt(x)
>function B(x,n,k) := bin(n,k)*x^k*(1-x)^(n-k)
>t := linspace(0,1,500);
>n := 10; k := 0:n;
>MB := B(t,n,k');  // Matrix der Werte B(i,n) in t[j]
>s := f(k/n).MB;  // Zeilen mit f(i/n) gewichtet addieren
>plot2d("f(x)",0,1);
>plot2d(t,s,>add,color=red);
```

Es ist keineswegs so, dass jede Folge von Interpolationspolynomen gegen f konvergiert. In der Tat kann man mit funkionalanalytischen Methoden zeigen, dass es zu jeder Folge von Interpolationspunkten eine Funktion f gibt, so dass die Folge der Interpolationspolynome nicht konvergiert.

Umgekehrt kann man aber zeigen, dass es zu jedem stetigen f eine Folge von Interpolationspunkten gibt, so dass die zugehörigen Interpolationspolynome gleichmäßig gegen f konvergieren. Dies wird aus dem Ergebnis des folgenden Abschnitts folgen.

3.8 Gleichmäßige Approximation

3.76. Definition: Wir bezeichnen für $f \in C[a,b]$ mit

$$e_n(f) = \inf\{\|f - p_n\|_{[a,b]} : p_n \in \mathcal{P}_n\}$$

den Approximationsfehler von f bezüglich $\mathcal{P}_n$ auf $[a,b]$. Ein Polynom $p_n \in \mathcal{P}_n$ mit

$$e_n(f) = \|f - p_n\|_{[a,b]}$$

heißt gleichmäßig beste Approximation an f bezüglich $\mathcal{P}_n$ auf $[a,b]$.

3.77 Satz: *Es gibt zu $f \in \mathbb{N}$, $n \in \mathbb{N}$, genau eine gleichmäßig beste Approximation p_n^* bezüglich $\mathcal{P}_n$ auf $[a,b]$. Diese beste Approximation ist dadurch gekennzeichnet, dass es $n+2$ Punkte*

$$a \le x_1 < \ldots < x_{n+2} \le b$$

gibt und ein Vorzeichen $\sigma_n = \pm 1$, so dass

$$f(x_k) - p_n^*(x_k) = \sigma \cdot (-1)^k \cdot \|f - p_n^*\|_{[a,b]}$$

für $k = 1, \ldots, n+2$ gilt. Man bezeichnet solche Punkte als Alternantenpunkte.

Dieser Satz heißt Alternantensatz von Chebyshev.

3.78. Beispiel: EMT enthält ein Verfahren zur Berechnung dieser besten Approximation. Wir wenden es für $f(x) = \sqrt{x}$ auf $[0,1]$ mit $n = 10$ an.

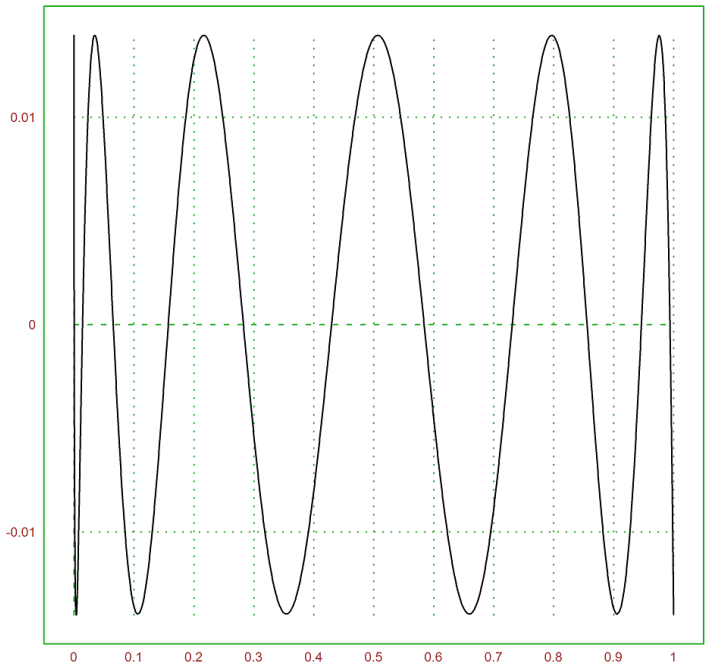

Abbildung 3.7: $f - p_{10}^*$ für $f(x) = \sqrt{x}$

```
>x=0:0.001:1; y=sqrt(x);
>xd,dd=remez(x,y,10);
>plot2d("divdifeval(xd,dd,x)-sqrt(x)",0,1);
```

Beweis: Zur Abkürzung setzen wir $I = [a, b]$. Offenbar brauchen wir nur unter den Polynomen $p \in \mathcal{P}_n$ mit

$$\|p\|_I \leq 2\|f\|$$

zu suchen. Diese Menge ist aber als beschränkte, abgeschlossene Teilmenge von $\mathcal{P}_n$ kompakt, und die stetige Funktion

$$p \mapsto \|f - p\|_I$$

nimmt daher ein Minimum an. Dies beweist die Existenz der besten Approximation.

Wir zeigen nun, dass notwendigerweise eine Alternante für jede beste Approximation p^* existiert. Dazu nehmen wir das Gegenteil an. Dann können wir die Extremalpunktmenge

$$E = \{x \in [a, b] : |f(x) - p^*(x)| = \|f - p^*\|_I\}$$

in $k < n + 2$ Teilmengen

$$E_1 < \ldots < E_k$$

zerlegen, so dass mit einem Vorzeichen $\sigma = \pm 1$ gilt

$$f(x) - p^*(x) = \sigma \cdot (-1)^k \cdot \|f - p^*\|_I \qquad \text{für alle } x \in E_k.$$

Die Mengen E_k sind daher kompakt. Wir wählen durch geschicktes Setzen der Nullstellen zwischen den E_k ein Polynom $q \in \mathcal{P}_n$, das auf den E_k dasselbe Vorzeichen hat wie $f - p^*$. Das ist wegen $k \leq n+1$ möglich. Es gilt dann

$$(f(x) - p^*(x)) \cdot q(x) > 0 \qquad \text{für alle } x \in E.$$

Nun behaupten wir, dass für $0 < \lambda$ klein genug

$$\|f - (p^* - \lambda q)\|_I < \|f - p_n^*\|_I.$$

Dies wäre ein Widerspruch dazu, dass p_n^* beste Approximation ist. Da ER kompakt ist, gibt es ein $\delta > 0$ mit

$$(f(x) - p^*(x)) \cdot q(x) > \delta \qquad \text{für alle } x \in E.$$

Die Menge aller Punkte $x \in [a,b]$ bezeichnen wir mit U. Ihr $K = [a,b] \setminus U$ ist kompakt. Es gilt für $x \in U$

$$\begin{aligned} |f(x) - (p^*(x) - \lambda q(x))|^2 &= |f(x) - p^*(x)|^2 - 2\lambda(f(x) - p^*(x))q(x) + \lambda^2 |q(x)|^2 \\ &\leq |f(x) - p^*(x)| - \lambda(2\delta - \lambda |q(x)|^2) \\ &\leq |f(x) - p^*(x)| - \lambda\delta \\ &< \|f - p^*\|_I - \lambda\delta \end{aligned}$$

für

$$\lambda < \frac{\delta}{\|q\|_I^2}.$$

Für $x \in K$ gilt

$$|f(x) - (p^*(x) - \lambda q(x))| \leq \|f - p^*\|_K + \lambda \|q\|_K < \|f - p^*\|_I$$

für

$$\lambda < \frac{\|f - p^*\|_I - \|f - p^*\|_K}{\|q\|_K}.$$

Also, weil K kompakt ist

$$\|f - (p^* - \lambda q)\|_K < \|f - p^*\|_K.$$

Dies zeigt unsere Behauptung.

Zum Beweis der Eindeutigkeit nehmen wir an, dass neben p^* auch q eine beste Approximation wäre. Wenn $x_1, \ldots, x_{n+2}$ eine Alternante gemäß dem Satz ist, dann gilt wegen

$$\begin{aligned} q(x_k) - p^*(x_k) &= (f(x_k) - p^*(x_k)) - (f(x_k) - q(x_k)) \\ &= \sigma(-1)^k \|f - q^*\|_I - (f(x_k) - q(x_k)), \end{aligned}$$

dass $q - p^*$ mindestens $n+2$ schwache Vorzeichenwechsel im Sinne von Satz 20 hat. Also folgt $q = p^*$. **q.e.d.**

3.79 Aufgabe: Seien $a \leq x_1 < \ldots < x_{n+2} \leq b$ und $p \in \mathcal{P}_n$ mit

$$f(x) - p(x) = \sigma \cdot (-1)^k \cdot h_k$$

mit positiven h_k, $k = 1, \ldots, n+2$, und $\sigma = \pm 1$. Zeigen Sie

$$e_n(f) \geq \min_k h_k.$$

Man nennt dies die Ungleichung von de la Vallée Poussin. Folgern Sie daraus nochmals, dass jede Fehlerfunktion mit einer Alternanten eine beste Approximation liefert.

3.80 Aufgabe: Berechnen Sie die beste Approximation p_1^* für $f(x) = e^x$ auf $[-1, 1]$ bezüglich $\mathcal{P}_1$, indem Sie explizit eine Gerade p_1^* konstruieren, so dass $f - p_1^*$ eine Alternante der Länge 3 hat.

3.81 Aufgabe: Folgern Sie aus dem Alternantensatz, dass man die beste Approximation mit Interpolation erhalten kann. Folgern Sie weiter, dass es eine Folge von Interpolationspunkten gibt, so dass die Interpolationspolynome gegen f konvergieren.

Leider weiß man nicht, wo die günstigen Interpolationspunkte liegen. Kadec hat aber bewiesen, dass sie zumindest für eine Teilfolge ähnlich wie die Nullstellen der Chebyshev-Polynome verteilt sein müssen.

3.82 Aufgabe: Zeigen Sie, dass die beste Approximation an eine gerade (ungerade) Funktion auf $[-1, 1]$ gerade (ungerade) ist.

3.83 Aufgabe: Zeigen Sie, dass man die beste Approximation an $\sqrt{x}$ auf $[0, 1]$ durch die beste Approximation an $|x|$ auf $[-1, 1]$ erhalten kann, und umgekehrt.

3.9 Kleinste Quadrate

Wir messen in diesem Abschnitt als Approximationsfehler nicht mit der Supremums-Norm, sondern mit einer Norm, die von einem Skalarprodukt stammt.

Aus der linearen Algebra ist bekannt, dass die beste Approximation in einem Skalarproduktraum durch die orthogonale Projektion berechnet werden kann. Es gilt

$$\|f - p^*\| = \inf\{\|f - p\| : p \in U\}$$

genau dann, wenn

$$f - p^* \perp U$$

ist. Dabei ist

$$\|h\| = \sqrt{\langle h, h\rangle}$$

die Norm auf einem Skalarproduktraum V mit Unterraum U. Falls $u_1, \ldots, u_n$ eine Orthonormalbasis von U ist, so hat man

$$p^* = \sum_{k=1}^{n} \langle f, u_k\rangle \cdot u_k.$$

Falls $u_1, \ldots, u_n$ irgendeine Basis von U ist, so hat man

$$p^* = \lambda_1 u_1 + \ldots + \lambda_n u_n,$$

wobei $a_1, \ldots, a_n$ das Gleichungssystem

$$\begin{pmatrix} \langle u_1, u_1\rangle & \ldots & \langle u_n, u_1\rangle \\ \vdots & & \vdots \\ \langle u_1, u_n\rangle & \ldots & \langle u_n, u_n\rangle \end{pmatrix} \cdot \begin{pmatrix} \lambda_1 \\ \vdots \\ \lambda_n \end{pmatrix} = \begin{pmatrix} \langle f, u_1\rangle \\ \vdots \\ \langle f, u_n\rangle \end{pmatrix}$$

lösen.

3.84 Aufgabe: Weisen Sie nach, dass die Lösung dieses Gleichungssystem in der Tat die beste Approximation ergibt, indem Sie nachweisen, dass p^* orthogonal auf $u_1, \ldots, u_n$ steht. Was passiert, wenn $u_1, \ldots, u_n$ nur ein Erzeugendensystem von U ist?

3.85. Beispiel: Wenn $u_1, \ldots, u_n$ eine orthogonale Basis ist, dann reduziert sich das Gleichungssystem zu

$$\begin{pmatrix} \langle u_1, u_1\rangle & & 0 \\ & \ddots & \\ 0 & & \langle u_n, u_n\rangle \end{pmatrix} \cdot \begin{pmatrix} \lambda_1 \\ \vdots \\ \lambda_n \end{pmatrix} = \begin{pmatrix} \langle f, u_1\rangle \\ \vdots \\ \langle f, u_n\rangle \end{pmatrix}$$

Die Lösung ist dann die Funktion

$$p^* = \sum_{k=1}^{n} \frac{\langle f, u_k\rangle}{\langle u_k, u_k\rangle} u_k.$$

Die Funktion $f - p_n^*$ steht dann senkrecht auf U. Dies wird im Schmidtschen Orthonormalisierungsverfahren verwendet.

3.86. Beispiel: Wir haben bereits nachgewiesen, dass

$$\frac{1}{\sqrt{2}}, \cos(x), \sin(x), \ldots, \cos(nx), \sin(nx)$$

orthonormal bezüglich des Skalarprodukts

$$\langle f, g\rangle = \frac{1}{\pi}\int_{-\pi}^{\pi} f(t)g(t)\,dt$$

sind. Die beste Approximation bezüglich $\mathcal{T}_n$ in der Norm, die zu diesem Skalarprodukt gehört, ist daher gemäß der vorigen Bemerkung

$$p^*(t) = \frac{a_0}{2} + \sum_{k=1}^{n} a_k \cos(kt) + b_k \sin(kt)$$

mit

$$a_k = \frac{1}{\pi}\int_{-\pi}^{\pi} f(t)\cos(kt)\,dt,$$
$$b_k = \frac{1}{\pi}\int_{-\pi}^{\pi} f(t)\sin(kt)\,dt.$$

Man beachte, dass für $k = 0$ aus dem ersten Term

$$\langle f, \frac{1}{\sqrt{2}}\rangle \cdot \frac{1}{\sqrt{2}} = \frac{a_0}{2}$$

folgt. Man bezeichnet die Bestimmung der a_k als Fourier-Analyse. Diese Bezeichnung verwendet man bisweilen für alle orthogonalen Projektionen in Skalarproduktträumen von Funktionen.

3.87 Satz: *Sei $A \in \mathbb{K}^{m\times n}$ eine Matrix, $b \in \mathbb{K}^m$. Dann wird*

$$\|Ax - b\|$$

für $x \in \mathbb{K}^n$ minimal genau dann, wenn x die Normalgleichung

$$A^*Ax = A^*b$$

löst. Die Norm bezeichnet hier die Euklidsche Norm auf dem $\mathbb{K}^n$, und wir setzen wie üblich $A^* = \overline{A^T}$.

Beweis: Setzt man

$$U = \text{Bild}\, A = \text{span}\,\{a_1, \ldots, a_n\}$$

mit den Spalten $a_1, \ldots, a_n \in \mathbb{K}^m$ von A, so haben wir $\|u - b\|$ für $u \in U$ zu minimieren. Das Minimum ist gemäß der obigen Bemerkung

$$u = x_1 a_1 + \ldots + x_n a_n = Ax$$

gekennzeichnet durch die Lösung des Gleichungssystem

$$\begin{pmatrix} \langle a_1, a_1\rangle & \ldots & \langle a_n, a_1\rangle \\ \vdots & & \vdots \\ \langle a_1, a_n\rangle & \ldots & \langle a_n, a_n\rangle \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} \langle b, a_1\rangle \\ \vdots \\ \langle b, a_n\rangle \end{pmatrix}.$$

Das Gleichungssystem ist äquivalent zu

$$(Ax)^T \cdot \overline{A} = b^T \overline{A}$$

Transponieren ergibt die Normalgleichung. **q.e.d.**

3.88. Beispiel: Seien m Punkte $x_1, \ldots, x_m \in \mathbb{R}$ gegeben und Werte

$$y_1, \ldots, y_m \in \mathbb{R}.$$

Dann können wir mit Polynomen vom Grad kleiner als $m - 1$ nicht mehr interpolieren. Wir können aber polynomiale Regression durchführen, indem wir

$$\sum_{k=1}^{m} (p(x_k) - y_k)^2$$

unter allen $p \in \mathcal{P}_n$ minimieren. Für $m = 1$ nennt man das Ergebnis die lineare Ausgleichsgerade

$$p(x) = a + bx.$$

Zu minimieren ist also

$$\left\| \begin{pmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_m \end{pmatrix} \cdot \begin{pmatrix} a \\ b \end{pmatrix} - \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix} \right\|$$

was zur Normalgleichung

$$\begin{pmatrix} m & \sum_k x_k \\ \sum_k x_k & \sum_k x_k^2 \end{pmatrix} \cdot \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} \sum_k y_k \\ \sum_k x_k y_k \end{pmatrix}$$

führt.

Wir berechnen in EMT eine solche Ausgleichsgerade für normalverteilt gestörte Werte einer gegebenen Gerade.

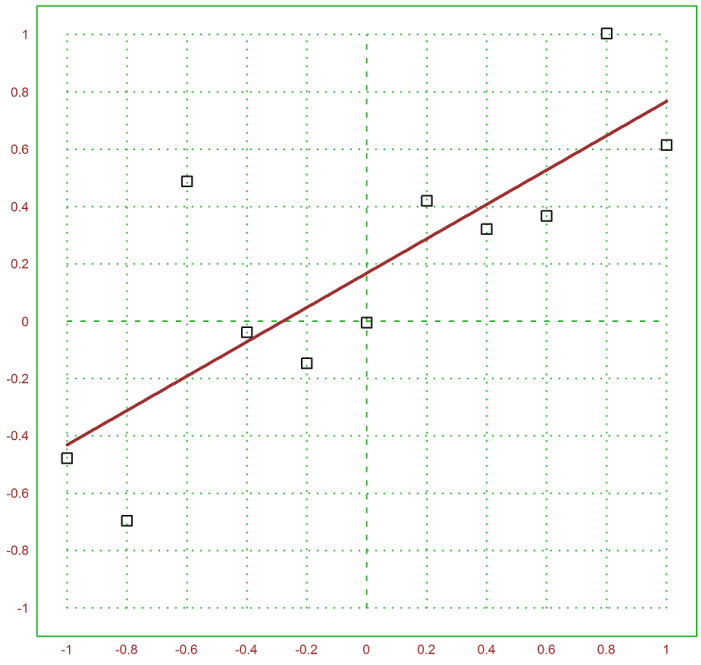

Abbildung 3.8: Ausgleichsgerade

```
>xp=linspace(-1,1,10); yp=0.1+0.5*xp+normal(size(xp))*0.2;
>plot2d(xp,yp,r=1,>points);
>p=polyfit(xp,yp,1)
 [ 0.168281157636  0.599406114325 ]
>plot2d("evalpoly(x,p)",>add,color=red,thickness=2);
```

3.89 Aufgabe: Geben Sie mit Hilfe der Cramerschen Regel explizite Formeln für a und b an.

3.10 Fourier-Transformation

Wir bezeichnen bei gegebenem $n \in \mathbb{N}$, $n \geq 2$, mit

$$\xi_k = e^{2\pi i k/n} \qquad \text{für } k = 0, \ldots, n-1$$

die n-ten Einheitswurzeln. Dies sind die komplexen Lösungen der Gleichung $z^n = 1$. Offenbar gilt für alle k

$$\xi_k = \xi_1^k, \quad \xi_1 = e^{2\pi i/n}.$$

Dann bilden die Vektoren

$$v_0 = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}, v_1 = \begin{pmatrix} \xi_0 \\ \vdots \\ \xi_{n-1} \end{pmatrix}, \ldots, v_{n-1} = \begin{pmatrix} \xi_0^{n-1} \\ \vdots \\ \xi_{n-1}^{n-1} \end{pmatrix}$$

eine orthogonale Basis des $\mathbb{K}^n$. In der Tat gilt

$$\sum_{k=0}^{n-1} \xi_k^p \overline{\xi_k^q} = \sum_{k=0}^{n-1} e^{2\pi i(p-q)k/n} = \begin{cases} 0 & p \neq q, \\ n & p = q. \end{cases}$$

Denn für $L = q - p$, so dass $\xi_L \neq 1$ ist, gilt

$$\sum_{k=0}^{n-1} e^{2\pi i L k/n} = \sum_{k=0}^{n-1} \xi_L^k = \frac{1 - \xi_L^n}{1 - \xi_L}$$

Für Werte

$$y_0, \ldots y_{n-1} \in \mathbb{C}$$

erhalten wir also aus Bemerkung 85 die diskrete beste Approximation im Sinne der kleinsten Quadrate bezüglich $\mathcal{P}_n$ auf den Einheitswurzeln durch

$$p^*(z) = a_0 + a_1 z + \ldots + a_{m-1} z^{m-1}$$

mit

$$a_p = \frac{\langle y, v_p \rangle}{\langle v_p, v_p \rangle} = \frac{1}{n} \sum_{k=0}^{n-1} y_k \xi_k^{-p}.$$

Die diskrete Approximation fällt im Fall $m = n$ mit der Interpolation zusammen, da man dann einfach in den n Einheitswurzeln interpolieren kann. Die obige Formel ist daher für $m = n$ eine Interpolationsformel.

3.90 Aufgabe: Rechnen Sie für $m = n$ direkt nach, dass mit den oben angegebenen a_p

$$p^*(\xi_k) = y_k \qquad \text{für alle } k = 0, \ldots, n-1$$

gilt.

Insgesamt folgt der folgende Satz.

3.91 Satz: *Das Interpolationspolynom*

$$p(z) = a_0 + a_1 z + \ldots + a_{n-1} z^{n-1}$$

in den Einheitswurzeln zu Werten $y_0, \ldots, y_{n-1}$ ist durch

$$a_p = \frac{1}{n} \sum_{k=0}^{n-1} y_k \xi_k^{-p}.$$

gegeben. Das abgeschnittene Polynom

$$p_m(z) = a_0 + a_1 z + \ldots + a_m z^m$$

minimiert

$$\sum_{k=0}^{n} |p_m(\xi_k) - y_k|^2$$

unter allen komplexen Polynomen $p_m \in \mathcal{P}_m$.

3.92. Definition: Wir definieren die diskrete Fourier-Transformation von

$$a = (a_0, \ldots, a_{n-1})$$

durch

$$\hat{a} = (p(\xi_0), \ldots, p(\xi_{n-1}))$$

mit

$$p(z) = a_0 + a_1 z + \ldots + a_{n-1} z^{n-1}.$$

Die Fourier-Transformation

$$\phi : \mathbb{C}^n \to \mathbb{C}^n$$

definiert durch $\phi(a) = \hat{a}$ ist also die simultane Auswertung des Polynoms mit den Koeffizienten a_p in allen n-ten Einheitswurzeln. Ihre Inverse ϕ^{-1} ist die Interpolation in den n-ten Einheitswurzeln.

3.93 Satz: *Sei*

$$\hat{a} = (y_0, \ldots, y_{n-1})$$

die diskrete Fourier-Transformation von $a \in \mathbb{C}^n$. Dann gilt

$$\phi^{-1}(y) = \frac{1}{n}\overline{\phi\left(\overline{y}\right)}.$$

Beweis: Sei

$$\phi(a) = y$$

Dann gilt also die angegebene Formel für die a_p. Setzt man

$$q(z) = \overline{y_0} + \overline{y_1} z + \ldots \overline{y_{n-1}} z^{n-1},$$

so folgt

$$\frac{1}{n}\overline{q(\xi_p)} = \frac{1}{n}\overline{\sum_{k=0}^{n} \overline{y_k}\xi_p^k} = \frac{1}{n}\overline{\sum_{k=0}^{n} \overline{y_k}\xi_1^{pk}} = \frac{1}{n}\overline{\sum_{k=0}^{n} \overline{y_k}\xi_k^{p}} = \frac{1}{n}\sum_{k=0}^{n} y_k \xi_k^{-p} = a_p.$$

für $p = 0, \ldots, n-1$. Es folgt

$$\frac{1}{n}\overline{\phi\left(\overline{y}\right)} = a = \phi^{-1}(y).$$

Das ist die Behauptung. **q.e.d.**

Die diskrete Fourier-Transformation und ihre Inverse lässt sich sehr schnell mit Hilfe der Fast-Fourier-Transformation (FFT) berechnen. Seien dazu $\xi_{k,2n}$ die $2n$-ten Einheitswurzeln und $\xi_{k,n} = \xi_{k,2n}^2$ die n-ten Einheitswurzeln, sowie

$$p(z) = a_0 + \ldots + a_{2n-1} z^{2n-1}.$$

Dann gilt

$$\begin{aligned} y_p &= p(\xi_{p,2n}) \\ &= \sum_{k=0}^{2n} a_k \xi_{p,2n}^k \\ &= \sum_{k=0}^{n} a_{2k} \xi_{p,2n}^{2k} + \xi_{p,2n} \sum_{k=0}^{n} a_{2k+1} \xi_{p,2n}^{2k} \\ &= \sum_{k=0}^{n} a_{2k} \xi_{p,n}^{k} + \xi_{p,2n} \sum_{k=0}^{n} a_{2k+1} \xi_{p,n}^{k} \end{aligned}$$

für $p = 0, \ldots, 2n-1$. Dies eine offensichtliche Möglichkeit die Werte

$$\phi(d_0, \ldots, d_{2n-1}) = (y_0, \ldots, y_{2n-1})$$

aus den Werten

$$\begin{aligned} \phi(d_0, d_2, \ldots, d_{2n-2}) &= (\tilde{y}_0, \tilde{y}_2, \ldots, \tilde{y}_{2n-2}), \\ \phi(d_1, d_3, \ldots, d_{2n-1}) &= (\tilde{y}_1, \tilde{y}_3, \ldots, \tilde{y}_{2n-1}) \end{aligned}$$

zu berechnen, nämlich

$$y_p = \tilde{y}_{2p} + \xi_{p,2n} \tilde{y}_{2p+1}.$$

Man erhält man nun wegen $\xi_{2n}^n = -1$

$$y_{p+n} = \tilde{y}_{2p} - \xi_{p,2n} \tilde{y}_{2p+1}.$$

Damit lassen sich alle y_p, $p = 0, \ldots, 2n-1$ berechnen.

3.94. Beispiel: Die FFT wird gewöhnlich zur Frequenz-Analyse verwendet. Die Funktionen

$$v_k(t) = e^{ikt} \qquad \text{für } k \in \mathbb{N}_0$$

erzeugen dabei die Frequenz k. Sie bilden ein Orthonormalsystem auf $[-\pi, \pi[$ bezüglich des komplexen Skalarprodukts

$$\langle v, w \rangle = \frac{1}{2\pi} \int_{-\pi}^{\pi} v(t) \overline{w(t)}\, dt.$$

Die orthogonale Projektion

$$p(t) = \sum_{k=0}^{n} \langle f, v_k \rangle v_k$$

hat die Koeffizienten

$$c_k = \langle f, v_k \rangle$$

deren Beträge den Frequenzanteil k von f widerspiegeln. Es liegt nahe, die Fourier-Entwicklung nach diesem System durch die Summe

$$c_p = \langle f, v_p \rangle \approx \frac{1}{n} \sum_{k=0}^{n-1} f(t_k) \overline{v_p(t_k)} = \frac{1}{n} \sum_{k=0}^{n-1} f(t_k) \overline{\xi_p^k}. = \frac{1}{n} \sum_{k=0}^{n-1} f(t_k) \xi_k^{-p}.$$

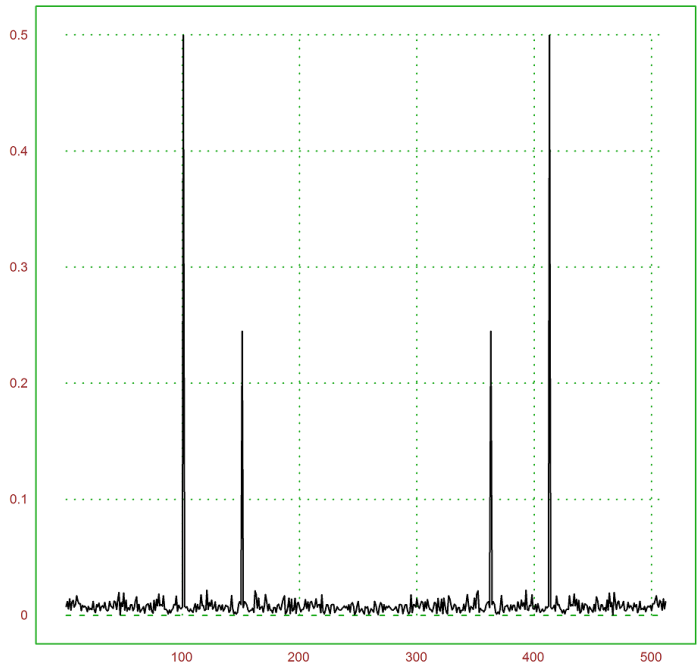

Abbildung 3.9: Frequenz-Analyse

zu ersetzen mit

$$\xi_k = e^{it_k} \qquad \text{für } k = 0, \ldots, n-1.$$

Dies ist die inversen FFT von

$$y_0 = f(t_0), \quad \ldots, \quad y_{n-1} = f(t_{n-1}).$$

Das folgende Beispiel analysiert ein verrauschtes Signal mit Frequenzen 100 und 150.

```
>n=512; t=(0:n-1)*2pi/n;
>s=cos(100*t)+sin(150*t)/2+normal(1,n)*0.2;
>plot2d(abs(ifft(s)));
```

Es tritt dabei der Effekt auf, dass die Frequenzen spiegelbildlich erscheinen. Das überlegt man sich am besten, indem man ϕ selbst betrachtet. Für ein Polynom p mit reellen Koeffizienten gilt nämlich

$$p(\xi_k) = p(\overline{\xi_{n-k}}) = \overline{p(\xi_{n-k})}.$$

Also haben die Auswertungen in ξ_p und ξ_{n-p} denselben Betrag. Die Fourier-Transformation ist spiegelbildlich zur Mitte. Für die inverse Transformation ϕ^{-1} gilt nach dem obigen Satz dasselbe.

Nach der FFT kann das Signal um das Rauschen bereinigt werden, indem Frequenzen mit niedrigen Werten auf 0 gesetzt werden. Oder es kann nur mit relevanten Frequenzen gespeichert

werden (MP3-Signalkompression).

3.95 Aufgabe: Sei $g : [\pi, \pi[\to \mathbb{R}$ eine Funktion, und

$$f(e^{it}) = g(t).$$

Setzen Sie die Koeffizienten der Fourier-Entwicklung von f

$$c_k =< f, v_k > \qquad \text{für } k = 0, 1, 2, \ldots$$

und die Fourier-Koeffizienten a_k, b_k von g zueinander in Beziehung.

3.96 Aufgabe: Berechnen Sie die Fourier-Koeffizienten a_k, b_k von

$$f(t) = \text{sign}\,(t).$$

für $t \in [-\pi, \pi[$.

3.97 Aufgabe: Berechnen Sie mit EMT und `ifft` die Interpolation in den 8-ten Einheitswurzeln an die Exponentialfunktion. Berechnen Sie dann die beste Approximation aus $\mathcal{P}_5$ mit Hilfe der Normalgleichung und vergleichen Sie die Ergebnisse.

Kapitel 4

Iterationsverfahren

4.1 Fixpunktiteration

Wir wollen Gleichungen der Form

$$f(x) = 0$$

auf, wobei

$$f : D \to \mathbb{R}^l, \quad D \subseteq \mathbb{R}^m$$

ist. Wir haben also m Unbekannte und l Gleichungen. Damit unter allgemeinen Bedingungen lokal eindeutige Lösungen existieren können, muss $m = l$ sein.

4.1. Beispiel: Wir behandeln hier Gleichungen mit einer Variablen, wie

$$f(x) = x^2 - 2 = 0,$$

oder auch nicht-lineare Gleichungssysteme mit mehreren Variablen, wie

$$x_1^2 + x_2^2 = 10,$$
$$x_1 + x_2 = 1.$$

Dieses Beispiel schreiben wir als

$$f\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} x_1^2 + x_2^2 - 10 \\ x_1 + x_2 - 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

Nach dem Satz über die lokale Umkehrbarkeit, gibt es eine Umkehrfunktion der stetig differenzierbaren Funktion f lokal um $x \in D$, wenn die Ableitungsmatrix $Df(x)$ invertierbar ist. Falls dann

$$f(x) = 0$$

ist, so ist x die einzige Lösung dieser Gleichung in dieser Umgebung. Dieser Satz setzt voraus, dass $D \subset \mathbb{R}^m$ offen ist, und f nach $\mathbb{R}^m$ abbildet, dass also genauso viele Unbekannte wie Gleichungen vorhanden sind.

Man kann ein Gleichungssystem als Fixpunktproblem umschreiben. Es gilt

$$f(x) = 0 \Leftrightarrow g(x) = x$$

mit

$$g(x) = x + \phi(f(x)),$$

wenn ϕ genau die eine Nullstelle 0 hat. Eine Lösung x von $g(x) = x$ heißt Fixpunkt von g. Beispielsweise kann man

$$g(x) = x + h(x)f(x)$$

setzen mit einer Funktion $h : D \to \mathbb{R}$, die nicht 0 wird. Das Iterationsverfahren

$$x_0 \in D, \quad x_{n+1} = g(x_n)$$

bezeichnet man als Fixpunkt-Iteration oder auch als Picard-Iteration. Die Iteration ist durchführbar, wenn

$$g : D \to D$$

eine Abbildung ist, auch für sehr allgemeine Mengen D, etwa für Mengen von Funktionen. Die oben verwendete Funktion ϕ trägt dazu bei, dass die Iteration möglichst gut konvergiert.

4.2 Satz: *Sei D ein metrischer Raum, $g : D \to D$ stetig. Wenn dann die Fixpunkt-Iteration mit einem Startpunkt $x_0 \in D$ konvergiert, so konvergiert sie gegen einen Fixpunkt.*

Beweis: Sei x_F der Grenzwert der Folge. Es gilt dann

$$g(x_F) = \lim_{n\to\infty} g(x_n) = \lim_{n\to\infty} x_{n+1} = x_F$$

wegen der Stetigkeit von g. **q.e.d.**

4.3. Beispiel: Wir betrachten die Fixpunktiteration

$$x_{n+1} = \sqrt{1 + x_n},$$

die wir mit einem $x_0 > 0$ starten. Die Funktion $g(x) = \sqrt{1+x}$ hat in $I = [0, \infty[$ den Fixpunkt

$$x_F = \frac{1+\sqrt{5}}{2} \approx 1.618$$

und die Ableitung

$$\frac{1}{2} > g'(x) = \frac{1}{2\sqrt{1+x}} > 0.$$

Mit Hilfe des Mittelwertsatzes gilt

$$x_F - x_{n-1} = g(x_F) - g(x_n) = g'(\xi)(x_F - x_n)$$

für ein ξ zwischen x_n und x_F. Wenn wir mit $0 < x_0 < x_F$ starten, so steigt die Folge also streng monoton, und sie ist beschränkt durch x_F. Sie konvergiert also nach dem obigen Satz gegen x_F.

Wenn wir mit $x_0 > x_F$ starten, so folgt mit dem gleichen Argument, dass die Folge monoton fallend gegen x_F konvergiert.

Wir können solche Iterationen in EMT mit einer Schleife oder mit der Funktion `iterate` durchrechnen. Es gibt auch eine schöne Art, diese Iterationen zu visualisieren.

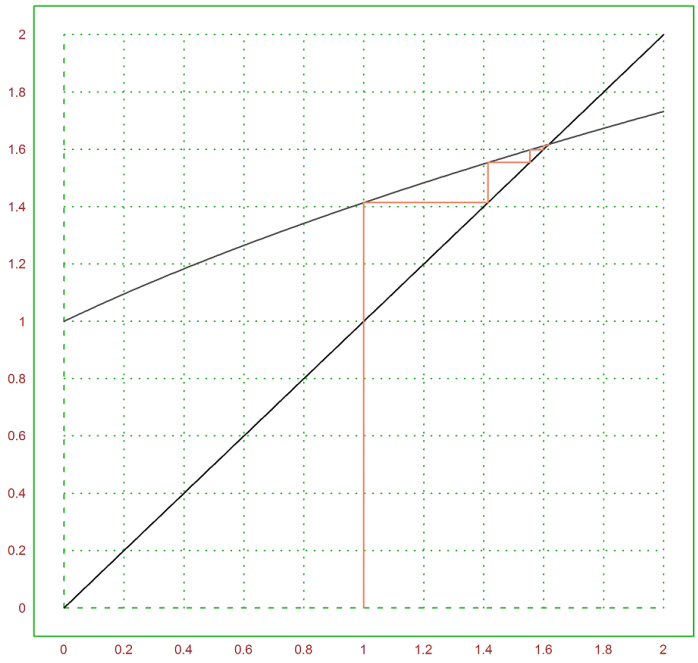

Abbildung 4.1: Visualierung der Fixpunkt-Iteration mit $x_0 = 1$

```
>function g(x) &= sqrt(1+x);
>iterate("g(x)",1,n=20)'
       1.41421356237
       1.55377397403
       1.59805318248
       1.61184775413
       ...
       1.6180339883
       1.61803398861
       1.61803398871
>iterate("g(x)",2,n=20)'
       1.73205080757
       1.65289165028
       1.62876998078
       ...
       1.61803398899
       1.61803398882
       1.61803398877
>fwebplot("g(x)",0,2,1,5);
```

4.4 Aufgabe: Sei $g : I \to I$ eine stetig differenzierbare Funktion mit Fixpunkt $x_F \in I$, $I \subseteq \mathbb{R}$ ein offenes Intervall. Zeigen Sie, dass es ein $r > 0$ gibt, so dass die Fixpunktiteration mit beliebigem Startpunkt $x_0 \in [x_F - r, x_F + r]$ gegen x_F konvergiert, wenn

$$|g'(x_F)| < 1$$

ist. Man nennt dann x_F eine anziehenden Fixpunkt. Zeigen Sie, dass die Fixpunktiteration für keinen

Startpunkt $x_0 \in I$ gegen x_F konvergieren kann, wenn

$$|g'(x_F)| > 1$$

ist, außer sie trifft zufällig x_F genau. Man nennt dann x_F eine abstoßenden Fixpunkt.

4.5 Aufgabe: Sei $g : I \to I$ differenzierbar, und $g' > 0$ auf I, $I \subseteq \mathbb{R}$ ein offenes Intervall. Zeigen Sie, dass die Fixpunktiteration monoton fällt, wenn nur $x_1 < x_0$ ist.

4.6 Aufgabe: Sei $g : I \to \mathbb{R}$, $I =]0, \infty[$ definiert durch

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

Zeigen Sie, dass g auf I nur den Fixpunkt $x_F = \sqrt{2}$ hat. Berechnen Sie die Nullstellen und die Vorzeichen der Ableitung g' auf I. Folgern Sie

$$x > x_F \Rightarrow g(x) > x_F$$

für alle $x \in I$. Sei $x_{n+1} = g(x_n)$ die Fixpunktiteration mit Startwert $x_0 = 2$. Zeigen Sie, dass die Folge monoton fallend gegen x_F konvergiert. Führen Sie die Fixpunktiteration auf einem Rechner durch. Zeichen Sie den Webplot mit EMT. Zeigen Sie mit Hilfe der Taylorformel

$$|x_{n+1} - x_F| = \frac{|g''(\xi)|}{2} \cdot |x_n - x_F|^2 \leq c|x_n - x_F|^2$$

für ein $\xi \in]x_F, x_n[$. Eine solche Konvergenz nennt man quadratische Konvergenz.

4.7 Satz: *Sei $g : I \to \mathbb{R}$ p-mal stetig differenzierbar auf dem offenen Intervall I, $p \in \mathbb{N}$, und $x_F \in I$ ein Fixpunkt von g. Sei*

$$g(x_F) = x_F, \quad g'(x_F) = \ldots = g^{(p-1)}(x_F) = 0, \quad g^{(p)}(x_F) \neq 0,$$

Zeigen Sie, dass dann das Iterationsverfahren mit Konvergenzordnung p konvergiert, das also

$$|x_{n+1} - x_F| \leq c|x_n - x_F|^p$$

mit einer Konstanten $c > 0$ gilt, sofern man x_0 nahe genug bei x wählt. Falls $p = 1$ ist und $|g'(x_F)| < 1$, so kann man $0 \leq c < 1$ wählen.

Beweis: Es gilt mit der Taylorformel

$$g(x) = x_F + \frac{g^{(p)}(\xi_x)}{p!}(x - x_F)^p.$$

Da die Ableitung in einer Umgebung beschränkt ist, folgt die Existenz eines $c > 0$ mit

$$|x_{n+1} - x_F| = |g(x_n) - x_F| \leq c|x - x_F|^p.$$

Im Fall $p = 1$ kann man $c < 1$ wählen, wenn $|g'(x_F)| < 1$ gilt. **q.e.d.**

4.8. Beispiel: Wenn $f : \mathbb{R}^k \to \mathbb{R}^k$ eine Abbildung ist, so gilt

$$f(x) = 0 \Leftrightarrow x = g(x) := (I + M(x)f(x)) \cdot x$$

mit einer invertierbaren Matrix $M(x)$, die sogar in Abhängigkeit von x gewählt werden kann. Durch geschickte Wahl von $M(x)$ kann man die Konvergenz erheblich beschleunigen.

4.9 Satz: *Sei X ein metrischer Raum mit Metrik d, der vollständig sei (jede Cauchy-Folge konvergiert), und $f : X \to X$ eine Abbildung, so dass es ein*

$$0 \leq \rho < 1$$

gibt mit

$$d(f(a), f(b)) \leq \rho \cdot d(a, b) \qquad \text{für alle } a, b \in X.$$

Eine solche Abbildung nennt man kontrahierende Abbildung. Dann hat f genau eine Fixpunkt

$$f(x_F) = x_F$$

in X, und die Fixpunktiteration

$$x_{n+1} = f(x_n)$$

konvergiert für jeden Startpunkt $x_0 \in X$ gegen x_F.

Dieser Satz heißt Banachscher Fixpunktsatz.

Beweis: Man zeigt per Induktion

$$d(x_{n+1}, x_n) \leq \rho^n \cdot d(x_1, x_0)$$

für alle $n \in \mathbb{N}_0$. Es folgt mit der Dreiecksungleichung

$$\begin{aligned} d(x_{n+k}, x_n) &\leq d(x_{n+k}, x_{n+k-1}) + \ldots + d(x_{n+1}, x_n) \\ &\leq (\rho^{n+k-1} + \ldots + \rho^n) d(x_0, x_1) \\ &\leq \rho^n (1 + \rho + \rho^2 + \ldots) d(x_0, x_1) \\ &\leq \frac{d(x_0, x_1)}{1 - \rho} \rho^n \end{aligned}$$

für $n, k \in \mathbb{N}_0$. Daraus folgt, dass $(x_n)_{n \in \mathbb{N}}$ eine Cauchy-Folge ist, die daher konvergiert. Aus der Kontraktionsbedingung folgt, dass f stetig ist. Also konvergiert die Folge gegen einen Fixpunkt x_F.

Sei $\tilde{x}_F$ ein weiterer Fixpunkt. Dann muss gelten

$$d(x_F, \tilde{x}_F) = d(g(x_F), g(\tilde{x}_F)) \leq \rho \cdot d(x_F, \tilde{x}_F).$$

Wegen $\rho < 1$ folgt $x_F = \tilde{x}_F$. **q.e.d.**

4.10 Aufgabe: Wenden Sie diesen Satz auf die Kosinus-Funktion an. Zeigen Sie, dass die Fixpunktiteration sogar bei beliebigem $x_0 \in \mathbb{R}$ konvergiert. Testen Sie die Iteration.

Wir haben unter der Kontraktionsbedingung

$$d(x_n, x_0) < \frac{d(x_1, x_0)}{1 - \rho} = r$$

Die Kontraktionsbedingung braucht daher nur in

$$U_r(x_0) = \{x \in \mathbb{R} : d(x_0, x) < r\}.$$

zu gelten.

4.11 Aufgabe: Zeigen Sie

$$d(x_n, x_F) \le \frac{d(x_n, x_{n+1})}{1-\rho}$$

unter den Bedingungen des Satzes.

Unter geeigneten Bedingungen konvergiert

$$\frac{d(x_{n+1}, x_n)}{d(x_n, x_{n-1})} \to \rho.$$

Es ist daher sinnvoll

$$\frac{d(x_{n+1}, x_n)^2}{d(x_{n+1}, x_n) - d(x_{n+2}, x_{n+1})} < \epsilon$$

als Abbruchkriterium für die Iteration zu verwenden.

Wir testen dies mit EMT. Dazu definieren wir eine Funktion `iter`, die die Iteration nach diesem Kriterium abbricht. Die Funktion erwartet eine Funktion `f` als Parameter, sowie den Startpunkt x_0 der Iteration und den Abbruchwert ϵ.

```
>function f(x) := sqrt(1+x)
>function iter (f,x0,eps) ...
$  x=f(x0);
$  d=abs(x-x0);
$  loop 1 to 1000
$    xn=f(x);
$    dn=abs(x-xn);
$    if d^2/(d-dn)<eps then return xn; endif;
$    x=xn; d=dn;
$  end
$  error("Two many iterations!");
$endfunction
>iter("f",1,1e-5), abs(%-solve("f(x)-x",1)),
 1.61803347393
 5.14821744124e-007
```

4.12 Aufgabe: Sei $g : I \to \mathbb{R}$ differenzierbar, I ein offenes Intervall, und g habe in I einen Fixpunkt x_F. Sei

$$0 \le g'(x) \le 1$$

für alle $x \in I$. Zeigen Sie $g(I) \subseteq I$.

4.13 Aufgabe: Sei $g : [x_F - r, x_F + r] \to \mathbb{R}$ differenzierbar, x_F ein Fixpunkt von g. Sei

$$|g'(x)| \le 1$$

für alle $x \in I$. Zeigen Sie $g(I) \subseteq I$.

4.14 Aufgabe: Sei

$$g(x) = 1 + \frac{1}{x},$$

sowie x_n die Folge der Fixpunktiterationen mit $x_0 = 2$. Zeigen Sie, dass $I = [3/2, 2]$ in sich selbst abgebildet wird, und dass g dort mit $\rho = 4/9$ kontrahiert. Zeigen Sie, dass $(x_{2n})_{n\in\mathbb{N}}$ monoton fällt, und dass $(x_{2n+1})_{n\in\mathbb{N}}$ monoton wächst.

4.15. Beispiel: Sei für Funktionen $y \in C(I)$ die Funktion $\phi(y)$ definiert durch

$$\phi(y)(x) = y_0 + \int_{x_0}^{x} f(t, y(t))\, dt$$

mit $x_0, y_0 \in \mathbb{R}$ und einem offenen Intervall I mit $x_0 \in I$. Dabei sei $f : \mathbb{R}^2 \to \mathbb{R}$ eine Funktion, die einer Lipschitz-Bedingung

$$|f(a, b) - f(a, \tilde{b})| \leq L \cdot |b - \tilde{b}| \qquad \text{für alle } a, b, \tilde{b} \in \mathbb{R}$$

genügt. Dann kontrahiert der Operator $\phi : C(I) \to C(I)$ in der Supremumsnorm, wenn man nur I klein genug um x_0 wählt. Denn für zwei Funktionen $y, \tilde{y} \in C(I)$ gilt

$$\begin{aligned} \|\phi(y) - \phi(\tilde{y})\|_{[x_0-\delta, x_0+\delta]} &= \max_{x\in[x_0-\delta, x_0+\delta]} \left| \int_{x_0}^{x} (f(t, y(t)) - f(t, \tilde{y}(t)))\ dt \right| \\ &\leq \delta \cdot L \cdot \|y - \tilde{y}\|_{[x_0-\delta, x_0+\delta]}. \end{aligned}$$

Daher konvergiert die Fixpunktiteration mit dem Startwert $y(x) = y_0$ gegen eine Funktion $y_F(x)$, die Fixpunkt von ϕ ist. Es folgt

$$y_F(x) = y_0 + \int_{x_0}^{x} f(t, y_F(t))\, dt,$$

und zwar gleichmäßig auf I. y_F löst daher das Anfangswertproblem

$$y'(x) = f(x, y(x)), \quad y(x_0) = y_0.$$

4.16 Aufgabe: Zeigen Sie mit Hilfe einer vorigen Aufgabe, dass es hier genügt, wenn f auf einem offenen $D \subseteq \mathbb{R}^2$ definiert ist, und lokal um (x_0, y_0) eine Lipschitzbedingung erfüllt. Zeigen Sie insbesondere, dass dann ϕ während der ganzen Iteration wohldefiniert ist.

4.2 Das Newton-Verfahren

4.17. Beispiel: Für $f : I \to \mathbb{R}$ suchen wir die Nullstelle

$$f(x) = 0.$$

Das Newton-Verfahren nimmt als nächste Näherung die Nullstelle von

$$T_a(x) = f(a) + f'(a)(x - a)$$

wobei a eine schon berechnete Näherung sei. Löst man $T_a(x) = 0$ nach x auf, so erhält man Lösung

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

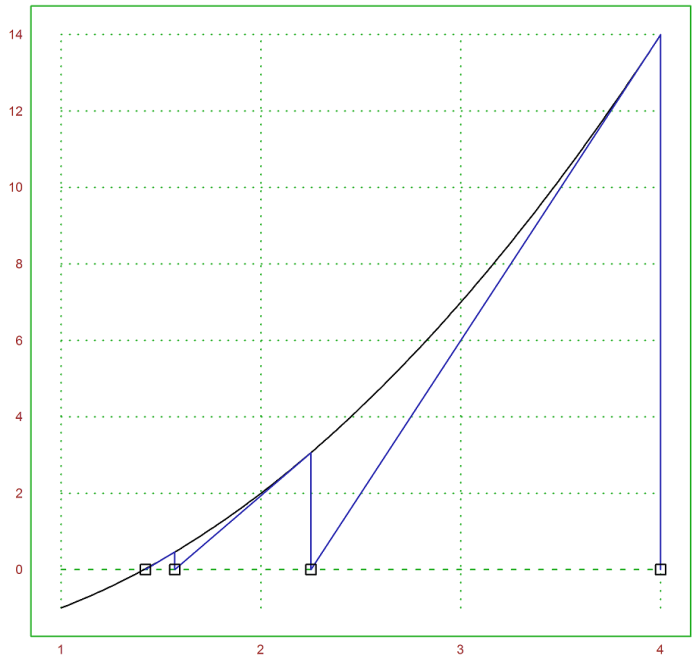

Abbildung 4.2: Newton-Verfahren für $f(x) = x^2 - 2$

Das ergibt das Iterationsverfahren

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

4.18. Beispiel: Für die Nullstelle von

$$f(x) = x^2 - 2$$

erhalten wir die Iteration

$$x_{n+1} = x_n - \frac{x_n^2 - 2}{2x_n} = \frac{1}{2}\left(x_n + \frac{2}{x_n}\right),$$

die wir schon in einer Übung untersucht haben.

4.19 Satz: *Sei $f : I \to \mathbb{R}$ zweimal stetig differenzierbar, I ein offenes Intervall, und $a \in I$ mit*

$$f(a) = 0, \quad f'(a) \neq 0.$$

Dann gibt es eine Umgebung U von a, so dass das Newton-Verfahren

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

für alle Startwerte $x_0 \in U$ gegen a konvergiert, und es gilt

$$|x_{n+1} - a| \leq c|x_n - a|^2$$

mit einer Konstanten $c > 0$. Eine solche Konvergenz nennt man quadratische Konvergenz.

Beweis: Der Iterations-Operator

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

hat die Eigenschaft

$$g'(a) = \frac{f'(a)f''(a)}{f'(a)^2} = 0.$$

Es gibt daher eine Umgebung $[a - \delta, a + \delta]$ von a, in der

$$|g'(x)| < \frac{1}{2}$$

ist, so dass dort

$$|x_{n+1} - a| = |g'(\xi)||x_n - a| \leq \frac{1}{2}|x_n - a|$$

gilt, und daher das Newton-Verfahren für alle Startwerte konvergiert. Weiter gilt nach dem Mittelwertsatz

$$g'(\xi) = \frac{f''(\xi)}{f'(\xi)^2} f(\xi) = \frac{f''(\xi)}{f'(\xi)^2} (f(\xi) - f(a)) = \frac{f''(\xi)}{f'(\xi)^2} f'(\tilde{\xi})(\xi - a).$$

mit

$$|\tilde{\xi} - a| < |\xi - a| < |x_n - a|.$$

Wegen $f'(a) \neq 0$, und weil f zweimal stetig differenzierbar ist, kann man $\delta > 0$ so klein wählen, dass

$$|g'(\xi)| \leq c|x_n - a|$$

Es folgt die Behauptung. **q.e.d.**

Wir haben schon Aufgabe 6 gesehen, dass

$$|x_{n+1} - a| = \frac{g''(\xi)}{2}|x_n - a|^2$$

aus der Taylorformel folgt, wenn g zweimal differenzierbar ist. Im obigen Beweis haben wir aber nur verwendet, dass f zweimal stetig differenzierbar, also g einmal stetig differenzierbar, ist.

4.20 Aufgabe: Sei

$$f''(a) = 0.$$

Folgern Sie aus dem obigen Beweis

$$|x_{n+1} - a| \leq c|x_n - a|^3,$$

also die kubische Konvergenz. Verifizierten Sie das am Beispiel $f(x) = x - x^3$ für die Nullstelle $a = 0$.

4.21. Beispiel: Das Newton-Verfahren ist in EMT implementiert. Es verwendet entweder eine vorgegebene Ableitung, oder es berechnet die Ableitung mit Hilfe von Maxima. Der Zielwert für $f(x) = y$ kann zusätzlich angegeben werden.

```
>newton("x^2-2","2x",1)
 1.41421356237
>newton("x^2","2x",1,n=5,y=2)
 1.41421356237
>mxmnewton("x^2-2",1)
 1.41421356237
```

4.22. Definition: Ersetzt man die Ableitung $f'(x_n)$ durch die Sekantensteigung

$$f'(x_n) \approx \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}},$$

so entsteht das Sekanten-Verfahren

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

Man benötigt zwei Startwerte x_0 und x_1. Die Berechnung der Ableitung ist nicht notwendig.

4.23 Satz: *Sei $f : I \to \mathbb{R}$ zweimal stetig differenzierbar, I ein offenes Intervall, und $a \in I$ mit*

$$f(a) = 0, \quad f'(a) \neq 0.$$

Dann gibt es eine Umgebung U von a, so dass das Sekanten-Verfahren für alle Startwerte $x_0 \in U$ gegen a konvergiert, und es gilt

$$|x_{n+1} - a| \leq \frac{e^{\alpha q^n}}{c}$$

mit einer Konstanten $c > 0$ und

$$q = \frac{1 + \sqrt{5}}{2} = 1.618\ldots$$

Beweis: Mit Hilfe von dividierten Differenzen haben wir

$$\begin{aligned} x_{n+1} - a &= (x_n - a) - \frac{f(x_n)}{[x_{n-1}, x_n]_f} \\ &= (x_n - a) \cdot \left(1 - \frac{[x_n, a]_f}{[x_{n-1}, x_n]_f}\right) \\ &= (x_n - a) \cdot (x_{n-1} - a) \cdot \frac{[x_{n-1}, x_n, a]_f}{[x_{n-1}, x_n]_f}. \end{aligned}$$

Es existieren außerdem ξ_1, ξ_2 im kleinsten Intervall, dass x_n, x_{n-1} und a enthält, so dass

$$[x_{n-1}, x_n, a]_f = \frac{1}{2} f''(\xi_1), \quad [x_{n-1}, x_n]_f = f'(\xi_2).$$

Wegen der Voraussetzung $f'(a) \neq 0$ erhalten wir

$$|x_{n+1} - a| \leq c\,|x_n - a|\,|x_{n-1} - a|$$

für eine Konstante $c > 0$, wenn x_n, x_{n-1} in einer geeigneten Umgebung von a liegen. Wählt man diese Umgebung klein genug, so folgt die Konvergenz des Verfahrens. Das im Satz angegebene q erfüllt

$$q^2 = q + 1.$$

Wir wählen ein $\alpha < 0$ mit

$$|x_n - a| \leq \frac{e^{\alpha q^n}}{c}$$

für $n = 0$ und $n = 1$. Dann gilt diese Gleichung per Induktion für alle $n \in \mathbb{N}$. Es gilt nämlich

$$|x_{n+1} - a| \leq c\,|x_n - a|\,|x_{n-1} - a| \leq \frac{e^{\alpha(q^n + q^{n-1})}}{c} = \frac{e^{\alpha q^{n+1}}}{c}$$

im Induktionsschritt. **q.e.d.**

Die quadratische Konvergenz

$$|x_{n+1} - a| \leq c\,|x_n - a|^2$$

bedeutet eine Abschätzung

$$|x_n - a| \leq \frac{e^{\alpha 2^n}}{c}$$

also mit $q = 2$. Zwei Schritte des Sekanten-Algorithmus ergeben aber eine Konvergenzordnung von $q^2 > 2$.

4.24. Beispiel: Wir führen das Sekanten-Verfahren zur Bestimmung von $\sqrt{2}$ durch.

```
>function f(x) &= x^2-2
```

$$x^2 - 2$$

```
>function fsecant(a,b) &= ratsimp(a - (b-a)/(f(b)-f(a))*f(a))
```

$$\frac{a\,b + 2}{b + a}$$

```
>n=7;
>x=zeros(1,n); x[1]=2; x[2]=1.8;
>for i=3 to n; x[i]=fsecant(x[i-1],x[i-2]); end;
>x'
                  2
                1.8
       1.47368421053
       1.42122186495
       1.41435753537
       1.41421391821
       1.41421356239
```

Nun schätzen wir die Konvergenzordnung ab. Dazu nehmen wir

$$\beta_n = \log|x_n - \sqrt{2}| = \alpha q^n - \log c$$

an, und berechnen q. Also

$$q = \frac{\beta_{n-1} - \beta_n}{\beta_n - \beta n - 1}.$$

```
>d=x-sqrt(2)
 [ 0.585786437627  0.385786437627  0.0594706481532  0.00700830257867
 0.000143972994612  3.55837791011e-007  1.81117343345e-011 ]
>beta=log(d)
 [ -0.53479999674  -0.952471333024  -2.8222723965  -4.9606597501
 -8.84588481339  -14.8487908532  -24.7344610819 ]
>betad=differences(beta)
 [ -0.417671336285  -1.86980106347  -2.1383873536  -3.88522506329
 -6.00290603977  -9.88567022878 ]
>betad[2:6]/betad[1:5]
 [ 4.4767282335  1.14364431349  1.8168948936  1.54506005237
 1.64681408693 ]
```

Das Sekanten-Verfahren ist das Verfahren, dass die Funktion `solve` von EMT verwendet. Es wird dabei neben dem Startwert x_0 ein zweiter Wert $x_1 = x_0 + \epsilon$ vom Programm ermittelt, falls kein Wert angegeben ist.

```
>solve("x^2-2",1,1.8)
 1.41421356237
>solve("x^2-2",1)
 1.41421356237
```

4.25. Definition: Verfahren die statt $f'(x_n)$ eine Näherung für die Ableitung verwenden, nennt man Quasi-Newton-Verfahren.

4.26. Beispiel: Das Sekantenverfahren kann man noch weiter vereinfachen, indem man

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

setzt, wobei

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

hin und wieder neu berechnet wird, aber für über mehrere Schritte konstant bleibt.

4.3 Konvergenzbeschleunigung

4.27. Beispiel: Sei $(x_n)_{n\in\mathbb{N}}$ eine gegen $x \in \mathbb{R}$ konvergente Folge, für die näherungsweise

$$x_{n+1} - x = k\,(x_n - x)$$

gilt (lineare Konvergenz der Ordnung 1). Dann kann man aus

$$x_{n+2} - x = k(x_{n+1} - x)$$
$$x_{n+1} - x = k(x_n - x)$$

x ausrechnen. Es folgt

$$x = \frac{x_n x_{n+2} - x_{n+1}^2}{x_{n+2} - 2x_{n+1} + x_n} = x_n - \frac{(x_{n+1} - x_n)^2}{x_{n+2} - 2x_{n+1} + x_n}.$$

Wir könnten also versuchen, die Konvergenz einer annähernd linear konvergenten Folge durch

$$\tilde{x}_n = x_n - \frac{(x_{n+1} - x_n)^2}{x_{n+2} - 2x_{n+1} + x_n}$$

zu verbessern. Definiert man den Folgenoperator

$$\Delta x_n = x_{n+1} - x_n$$

so gilt

$$x = x_n - \frac{(\Delta x_n)^2}{\Delta^2 x_n}.$$

Daher heißt diese Methode Δ^2-Methode von Aitken.

4.28 Satz: *Gilt für die Folge*

$$x_{n+1} - x = (k + \delta_n)(x_n - x) \neq 0$$

mit $\delta_n \to 0$, so existieren die $\tilde{x}_n$ für $n > N$ und

$$\lim_{n\to\infty} \frac{\tilde{x}_n - x}{x_n - x} = 0.$$

Die modifiziert Folge konvergiert also schneller.

Beweis: Sei $e_n = x_n - x$, also $e_{n+1} = (k + \delta_n)e_n$. Dann haben wir

$$\begin{aligned} x_{n+2} - 2x_{n+1} - x_n &= e_{n+2} - 2e_{n+1} + e_n \\ &= e_n((k + \delta_{n+1})(k + \delta_n) - 2(k + \delta_n) + 1) \\ &= e_n((k-1)^2 + \mu_n) \end{aligned}$$

mit $\mu_n \to 0$. Für genügend großes n ist dieser Ausdruck also verschieden von 0. Außerdem

$$\tilde{x}_n - x = e_n - e_n \frac{((k-1) + \delta_n)^2}{(k-1)^2 + \mu_n}.$$

Also

$$\lim_{n\to\infty} \frac{\tilde{x}_n - x}{x_n - x} = \lim_{n\to\infty} \left(1 - \frac{((k-1) + \delta_n)^2}{(k-1)^2 + \mu_n}\right) = 0.$$

Es folgt die Behauptung. **q.e.d.**

4.29. Beispiel: Wir iterieren $x_n = \cos(x_n)$. Die Beschleunigung der Konvergenz ist deutlich zu sehen.

```
>n=10;
>x=iterate("cos(x)",0.7,n+2)
 [ 0.764842187284  0.721491639598  0.750821328839  0.731128772573
 0.744421183627  0.735480200406  0.74150865166  0.73745045315
 0.740185285397  0.738343610351  0.739584428695  0.738748709662 ]
>dx=differences(x); ddx=differences(dx);
>x[1:n]-dx[1:n]^2/ddx // Aitkens Methode
 [ 0.738985502384  0.73903940107  0.739064555867  0.739075745916
 0.739080889637  0.739083202926  0.739084258817  0.739084736008
 0.739084953119  0.739085051454 ]
>sol=solve("cos(x)-x",2)
 0.739085133215
```

4.30. Definition: Zur Fixpunktsuche

$$g(x) = x$$

liegt daher es nahe, den neuen Iterations-Operator

$$g_S(x) = x - \frac{(g(x) - x)^2}{g(g(x)) - 2g(x) + x}$$

zu verwenden. Dieses Verfahren nennt man das Steffenson-Verfahren.

Wenn $g : I \to \mathbb{R}$ auf dem offenen Intervall I definiert ist, so ist die Auswertung von g_S natürlich nicht immer möglich. Es entspricht aber ein Schritt des Steffenson-Verfahrens, dem Aitken-Verfahren, angewendet auf

$$x_n, \quad x_{n+1} = g(x_n), \quad x_{n+2} = g(x_{n+1}).$$

Falls also g kontrahierend ist, und einen Fixpunkt $x \in I$ hat, so legt der Satz über das Aitken-Verfahren nahe, dass auch das Steffenson-Verfahren funktioniert.

Setzt man

$$f(x) = g(x) - x$$

so haben wir als Iteration

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

mit dem Fixpunkt x, der dann Nullstelle von f ist. Wegen

$$\frac{f(g(x_n)) - f(x_n)}{g(x_n) - g(x_n)} \approx f'(x_n)$$

kann das Steffenson-Verfahren als ein Quasi-Newton-Verfahren angesehen werden.

Falls x Fixpunkt von g_S ist, so ist es offenbar auch Fixpunkt von g.

4.31 Aufgabe: Sei g stetig differenzierbar, und x Fixpunkt von g mit $g'(x) \neq 1$. Zeigen Sie, dass g_S in x stetig ist, und x Fixpunkt von g_S ist.

4.32 Satz: *Unter den Voraussetzungen von Satz 7 mit $p \geq 2$ hat der Steffenson-Operator die Ordnung $2p - 1$.*

Beweis: Ersetzt man

$$\tilde{g}(x) = g(x_F + x) - x_F$$

so ist 0 Fixpunkt von $\tilde{g}$ mit denselben Voraussetzungen. Sieht man auf die Darstellung der Aitken-Methode mit Hilfe des Δ-Operators, so wird klar, dass für die Iterierten des Steffenson-Operators $\tilde{g}_S$ gilt

$$\tilde{x}_n = x_n - x_F.$$

Wir können daher $x_F = 0$ annehmen. Es gilt dann mit der Taylorformel

$$g(x) = \frac{g^{(p)}(\xi)}{p!}x^p.$$

Also

$$|g(x)| \leq c_1|x|^p.$$

Man berechnet

$$g_S(x) = \frac{xg(g(x) - g(x)^2}{g(g(x)) - 2g(x) + x}.$$

Daraus folgt

$$|g_S(x)| \leq c_2|x|^{2p-1}$$

Es folgt die Behauptung. **q.e.d.**

Der Aufwand für das Steffenson-Verfahren lohnt im Allgemeinen nicht. Offensichtlich konvergiert ja schon $g_2(x) = g(g(x))$ mit der Konvergenzordnung p^2, wegen

$$|g(g(x)) - x_F| \leq c|g(x) - x_F|^p \leq c^2\left(|x - x_F|^p\right)^p = c^2|x - x_F|^{p^2}.$$

Daher wird das Verfahren höchstens im Fall $p = 1$ eingesetzt, wo es als Quasi-Newton-Verfahren mehr als lineare Konvergenz verspricht.

4.4 Matrixnormen

4.33. Definition: Wir definieren für eine lineare Abbildung $\phi : V \to W$ eines normierten Vektorraums V in einen normierten Vektorraum W

$$\|\phi\| = \sup_{\|v\|=1} \|\phi(v)\|.$$

Speziell für Matrizen $A \in \mathbb{K}^{m\times nl}$ also

$$\|A\| = \sup_{\|v\|=1} \|A \cdot v\|.$$

Dies entspricht der Norm der zugehörigen linearen Abbildung $\phi(v) = Av$. Diese Matrixnorm hängt von der Wahl der Normen auf V und W ab.

4.34 Aufgabe: Man zeige

$$\|\phi(v)\| \leq \|\phi\| \cdot \|v\|$$

für alle $v \in V$.

4.35 Aufgabe: Man zeige, dass aus der Ungleichung

$$\|\phi(v)\| \leq c \cdot \|v\| \qquad \text{für alle } v \in V$$

$\|\phi\| \leq c$ folgt. Falls zusätzlich ein $v \in V$, $v \neq 0$, existiert mit

$$\|\phi(v)\| = c \cdot \|v\|,$$

so folgt $\|\phi\| = c$.

4.36 Aufgabe: Zeigen Sie

$$\|\phi \circ \psi\| \leq \|\phi\| \cdot \|\psi\|,$$

wobei $\psi : V \to W$, $\phi : W \to U$ lineare Abbildungen zwischen normierten Vektorräumen seien.

4.37 Satz: *Die lineare Abbildung $\phi : V \to W$ von einem normierten Vektorraum V in einen normierten Vektorraum W ist genau dann stetig, wenn*

$$\|\phi\| < \infty$$

gilt.

Beweis: Wir haben

$$\|\phi(v - w)\| \leq \|\phi\| \cdot \|v - w\|$$

Daraus folgt die Stetigkeit im Falle von $\|\phi\| < \infty$. Wenn andererseits ϕ stetig ist, so existiert zu $\epsilon = 1$ ein $\delta > 0$ mit

$$\|w\| < \delta \Rightarrow \|\phi(w)\| \leq 1.$$

Es folgt für $\|v\| = 1$

$$\|\phi(v)\| = \frac{1}{\delta}\|\phi(\delta v)\| \leq \frac{1}{\delta}.$$

Also

$$\|\phi\| \leq \frac{1}{\delta}.$$

q.e.d.

4.38 Aufgabe: Zeigen Sie, dass die Matrixnorm tatsächlich eine Norm auf dem Vektorraum aller Matrizen ist. Zeigen Sie, dass

$$\|A\|_F = \sqrt{\sum_{i,j} |a_{i,j}|^2}$$

eine Norm auf dem Raum $\mathbb{R}^{m \times m}$ ist, die zu keiner Vektorraumnorm gehört.

Da jede lineare Abbildung, die auf einem endlich dimensionalen Vektorraum definiert ist, stetig ist, ist die Matrixnorm nie gleich ∞.

4.39 Aufgabe: Sei für $v \in \mathbb{K}^m$

$$\|v\|_\infty = \max_{1 \leq k \leq m} |v_k|$$

die ∞-Norm. Wir wählen diese Norm für den $\mathbb{K}^m$ und den $\mathbb{K}^m$ und bezeichnen die zugehörige Matrixnorm mit $\|A\|_\infty$. Dann gilt

$$\|A\|_\infty = \max_{1 \leq j \leq m} \sum_{i=1}^{n} |a_{i,j}|.$$

Diese Norm bezeichnet man als Zeilensummennorm. Beweisen Sie das mit Hilfe der vorigen Aufgabe.

4.40 Aufgabe: Sei für $v \in \mathbb{K}^m$

$$\|v\|_1 = \sum_{k=1}^{n} |v_k|$$

die 1-Norm. Zeigen Sie, genau wie in der vorigen Aufgabe,

$$\|A\|_1 = \max_{1\le i\le m} \sum_{j=1}^{m} |a_{i,j}|$$

Dies ist die Spaltensummennorm.

4.41 Satz: *Die zur Euklidschen Norm gehörige Matrixnorm ist*

$$\|A\| = \max\{\sqrt{\lambda} : \lambda \text{ ist Eigenwert von } A^*A\}.$$

Beweis: Die $m \times m$-Matrix A^*A ist Hermitesch und positiv semi-definit. Sei

$$v_1, \ldots, v_m \in \mathbb{K}^m$$

eine Orthonormalbasis aus Eigenvektoren zu den nicht-negativen Eigenwerten

$$\lambda_1, \ldots, \lambda_n \ge 0$$

Dann gilt für $v = \alpha_1 v_1 + \ldots + \alpha_n v_n$

$$\begin{aligned}
\|Av\| &= \sqrt{\langle Av, Av\rangle} \\
&= \sqrt{\langle v, A^*Av\rangle} \\
&= \sqrt{\langle \sum_{k=1}^{m} \alpha_k v_k, \sum_{k=1}^{m} \alpha_k \lambda_k v_k\rangle} \\
&= \sqrt{\sum_{k=1}^{m} \lambda_k |\alpha_k|^2} \\
&\le \|v\| \max_{1\le k\le m} \sqrt{\lambda_k}
\end{aligned}$$

Es folgt $\le$ in der Behauptung des Satzes. Setzt man andererseits $v = v_k$ für den Eigenvektor zum größten Eigenwert, so folgt $=$. **q.e.d.**

4.42 Aufgabe: Zeigen Sie für $A \in \mathbb{K}^{m\times m}$

$$\|A\| \ge \sigma(A) = \max\{|\lambda| : \lambda \in \mathbb{C} \text{ ist Eigenwert von } A\}.$$

für jede zu einer Norm auf dem $\mathbb{K}^n$ gehörige Matrixnorm. Man nennt $\sigma(A)$ den Spektralradius von A. Die Abschätzung ist trivial für $\mathbb{K} = \mathbb{C}$. Für den reellen Fall weisen Sie nach, dass

$$\|v\|_{\mathbb{C}} = \sup\{\|\mathrm{Re}\left(e^{it}v\right)\| : t \in \mathbb{R},\ \|v\| = 1\}$$

eine Norm auf dem $\mathbb{C}^m$ ist, und dass für die zugehörigen Matrixnormen

$$\|A\|_{\mathbb{C}} = \|A\|$$

gilt.

4.43 Aufgabe: Zeigen Sie für jede Norm auf dem $\mathbb{K}^m$

$$\|x\| \leq \sum_{k=1}^{m} |x_k| \cdot \|e_k\|,$$

wobei e_k die Einheitsvektoren im $\mathbb{K}^m$ seien. Folgern Sie für die zugehörige Matrixnorm

$$\|A\| \leq C \cdot \sum_{i,j} |a_{i,j}|$$

mit einer Konstanten $C > 0$. Folgern Sie mit der Dreiecksungleichung für die Matrixnorm, dass die Matrixnorm stetig von den Einträgen der Matrix abhängt.

4.44 Aufgabe: Sei $M \in \mathbb{K}^{m\times m}$ invertierbar. Dann ist für jede Norm $\|\cdot\|$ auf dem $\mathbb{K}^m$ die Abbildung

$$x \mapsto \|Mx\|$$

wieder eine Norm. Die zugehörige Matrixnorm ist

$$\|A\|_M = \|MAM^{-1}\|.$$

Zeigen Sie dazu

$$\|Ax\|_M \leq \|MAM^{-1}\| \cdot \|x\|_M,$$

sowie die Existenz eines x, so dass hier die Gleichheit steht.

4.45 Satz: *Zu jedem Matrix $A \in \mathbb{K}^{m\times m}$ und jedem $\epsilon > 0$ existiert eine Norm auf dem $\mathbb{K}^m$, so dass*

$$\rho(A) \leq \|A\| \leq \rho(A) + \epsilon$$

ist.

Beweis: Es genügt, den Satz für $\mathbb{K} = \mathbb{C}$ zu beweisen, da eine Norm auf dem $\mathbb{C}^m$ eine Norm auf dem $\mathbb{R}^m \subset \mathbb{C}^m$ ist. Aufgrund der Definition ist die zugehörige reelle Matrixnorm nicht größer als die zugehörige komplexe Matrixnorm.

In $\mathbb{C}$ zerfällt das charakteristische Polynom in Linearfaktoren. Aufgrund von Sätzen der linearen Algebra ist A ähnlich zu einer rechten oberen Dreiecksmatrix R, in deren Diagonalen die Eigenwerte von A stehen.

$$R = M^{-1}AM.$$

Wir setzen D_δ gleich der Diagonalmatrix mit den Einträgen $1, \ldots, \delta^{n-1}$. Dann konvergiert

$$D_\delta^{-1} R D_\delta$$

für $\delta \to 0$ gegen eine Diagonalmatrix D mit den Eigenwerten von A. Das die Norm stetig von den Einträgen von einer Matrix abhängt, folgt

$$\|A\|_{(MD_\delta)^{-1}} \to \rho(A)$$

mit den Bezeichnungen der vorigen Aufgabe. **q.e.d.**

4.46. Beispiel: Das lineare Gleichungssystem

$$Ax = b$$

ist äquivalent zum Fixpunktproblem

$$x = (I - RA)x + Rb = g(x)$$

für eine invertierbare Matrix R. Der Operator g kontrahiert, wenn

$$\|I - RA\| < 1$$

für irgendeine Norm gilt. In manchen Problemen hat man es mit Matrizen $A \in \mathbb{K}^{n\times n}$ zu tun, die diagonaldominant sind, also

$$|a_{i,i}| \geq \sum_{j\neq i} |a_{i\,j}|$$

für alle $i = 1, \ldots, n$. In diesem Fall kann man

$$R = D^{-1}$$

wählen, wobei D die Matrix ist, die nur aus der Diagonale von A besteht.

4.47 Aufgabe: Zeigen Sie, dass dann in der Zeilensummennorm $\|I - RA\| < 1$ gilt. Schreiben Sie eine Formel für die Elemente von $x_{n+1} = (I - RA)x_n$ auf.

4.48 Satz: *Sei $D \subseteq \mathbb{R}^m$ offen, $f : D \to \mathbb{R}^m$ stetig differenzierbar, sowie $a, b \in D$, so dass die Strecke zwischen a und b auch zu D gehört. Dann gilt*

$$\|f(b) - f(a)\| \leq \max_{0\leq t\leq 1} \|Df(a + t(b - a))\| \cdot \|b - a\|.$$

Beweis: Sei $\gamma : [0, 1] \to \mathbb{R}^m$ der Weg

$$\gamma(t) = f(a + t(b - a))$$

der Funktionswerte entlang der Strecke von a nach b. Aufgrund von Sätzen der Analysis über die Kurvenlänge stetig differenzierbarer Kurven gilt

$$\|f(b) - f(a)\| = \|\gamma(1) - \gamma(0)\| \leq l(\gamma) = \int_0^1 \|\gamma'(t)\|\, dt,$$

wobei $l(\gamma)$ die Länge des Weges γ bezeichne. Wegen

$$\gamma'(t) = Df(a + t(b - a)) \cdot (b - a)$$

folgt daraus die Behauptung. **q.e.d.**

Der Satz gilt in jeder Norm mit zugehöriger Matrixnorm. Für die Euklidsche Norm kann man den Satz folgendermaßen beweisen: Wir definieren

$$h(t) = \langle \gamma(t) - \gamma(0), \gamma(1) - \gamma(0)\rangle.$$

Dann gibt es ein $\xi \in]0, 1[$ mit

$$h'(\xi) = h(1) - h(0) = \|\gamma(1) - \gamma(0)\|^2.$$

Mit Hilfe der Schwarzschen Ungleichung erhält man

$$h'(\xi) = \langle \gamma'(\xi), \gamma(1) - \gamma(0) \rangle \leq \|\gamma'(\xi)\| \cdot \|\gamma(1) - \gamma(0)\|.$$

Es folgt

$$\|f(b) - f(a)\| = \|\gamma(1) - \gamma(0)\| \leq \|\gamma'(\xi)\| = \|Df(a + \xi(b-a)) \cdot (b-a)\|,$$

woraus die Behauptung folgt. Man benötigt dieses spezielle Resultat, um den Satz über die Kurvenlänge einer stetig differenzierbaren Funktion allgemein zu beweisen.

4.49 Aufgabe: (a) Sei $I \subset \mathbb{R}$ ein offenes Intervall, $0 \in I$, und $\gamma : I \to \mathbb{R}^m$ ein zweimal stetig differenzierbarer Weg mit

$$\gamma'(0) = 0.$$

Zeigen Sie, dass es eine Konstante $c > 0$

$$\|\gamma(t) - \gamma(0)\| \leq c t^2$$

für alle $t \in I$. Verwenden Sie dazu den Weg

$$h(t) = \gamma(t) - (\gamma(0) + \gamma'(0) \cdot t)$$

und Beweis aus dem obigen Satz.

(b) Folgern Sie daraus, dass für eine zweimal stetig differenzierbare Funktion $f : D \to \mathbb{R}^m$ mit $D \subseteq \mathbb{R}^m$ offen, eine Konstante $c > 0$ existiert mit

$$f(\tilde{x}) = f(x) + Df(x) \cdot (x - \tilde{x}) + R(x),$$

wobei

$$R(x) \leq c\|x - \tilde{x}\|^2$$

für alle $x \in D$ gilt.

4.5 Nicht-Lineare Gleichungssysteme

4.50. Definition: Analog zum Newton-Verfahren für eine Variable können wir zur Bestimmung der Nullstelle von

$$f : D \to \mathbb{R}^m$$

mit einer offenen Menge $D \subseteq \mathbb{R}^m$ die Iteration

$$x_{n+1} = x_n - Df(x_n)^{-1} \cdot f(x_n)$$

verwenden. Sie verwendet die Nullstelle der tangentialen Abbildung

$$T(\tilde{x}) = x + Df(x) \cdot (\tilde{x} - x)$$

als nächste Näherung. Dies ist das Newton-Verfahren für mehrere Gleichungen und Variablen.

4.51. Beispiel: Sei

$$M = \{(x, y) \in \mathbb{R}^2 : x^2 y + y^2 x + x^4 = 1.\}$$

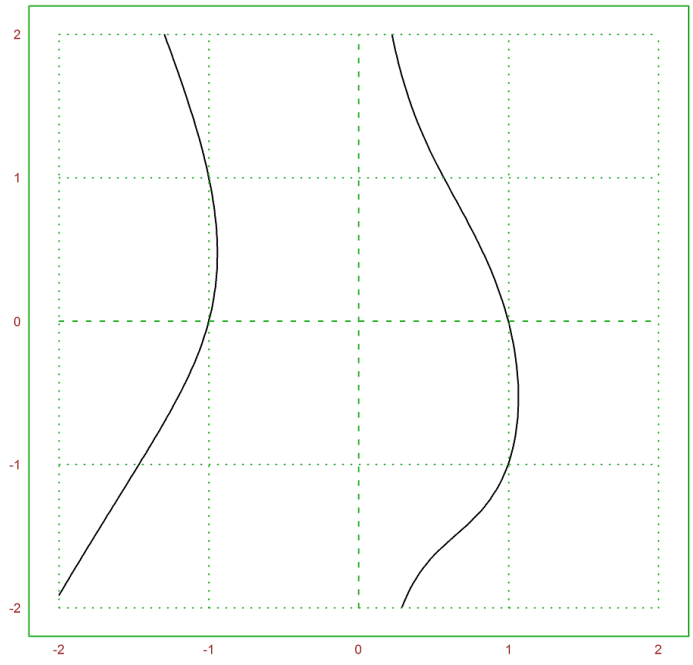

Abbildung 4.3: $x^2y + y^2x + x^4 = 1$

Wir suchen den Punkt auf M, der von $(1,1)$ den minimalen quadrierten Abstand hat. Wir wollen also

$$h(x,y) = (x-1)^2 + (y-1)^2$$

unter der Nebenbedingung

$$g(x,y) = x^2y + y^2x + x^4 - 1 = 0$$

minimieren. Die Lagrange-Bedingungen dazu lauten

$$\begin{aligned} y^2 + 2xy + 4x^3 &= 2\lambda(x-1), \\ 2xy + x^2 &= 2\lambda(y-1), \\ x^2y + y^2x + x^4 &= 1. \end{aligned}$$

Dies ist ein Gleichungssystem mit 3 Variablen und 3 Gleichungen. Man erhält die Gleichungen auch durch Nullsetzen des Gradienten der Lagrange-Funktion

$$L(x,y,\lambda) = h(x,y) - \lambda g(x,y).$$

zu lösen ist also

$$f(x,y,\lambda) = \text{grad } L(x,y,\lambda) = 0.$$

In EMT können wir Df symbolisch mit Maxima berechnen, und das Newton-Verfahren starten.

```
>function h(x,y) &= x^2 * y + y^2 * x + x^4 // Bedingung h(x,y)=1

                              2    2      4
                           x y  + x  y + x

>plot2d(h,niveau=1,r=2); // siehe Abbildung
>function L([x,y,lambda]) &= (x-a)^2+(y-b)^2 - lambda*(h(x,y)-1)

               2    2      4                      2          2
         (- x y  - x  y - x  + 1) lambda + (y - b)  + (x - a)

>function f([x,y,lambda]) &= gradient(L(x,y,lambda),[x,y,lambda])

             2              3
        [(- y  - 2 x y - 4 x ) lambda + 2 (x - a),
                       2                            2    2      4
             (- 2 x y - x ) lambda + 2 (y - b), - x y  - x  y - x  + 1]

>a=1; b=1; // Punkt (a,b)
>function Df([x,y,lambda]) &= jacobian(f(x,y,lambda),[x,y,lambda]);
>function newtoniter(v) := v - f(v).inv(Df(v))' // Zeilenvektoren!
>xsol=iterate("newtoniter",[1,1,0]) // findet minimalen Punkt
 [ 0.661468492772  0.823282064697  -0.23150454352 ]
>f(xsol) // Teste Lösung
 [ 0  0  0 ]
>plot2d(xsol[1],xsol[2],>points,>add); insimg; // siehe Abbildung
>iterate("newtoniter",[1,1,0],n=7) // nur wenige Schritte nötig!
      0.758620689655      0.896551724138    -0.0689655172414
       0.67242290684      0.834619627196      -0.18437947536
      0.661612280577      0.823563892249     -0.229661112122
      0.661468501264      0.823282208311     -0.231503373038
      0.661468492772      0.823282064697      -0.23150454352
      0.661468492772      0.823282064697      -0.23150454352
      0.661468492772      0.823282064697      -0.23150454352
```

Die eingebaute Funktion `newton2` führt diese Iteration automatisch durch.

```
>newton2("f","Df",[1,1,0])
 [ 0.661468492772  0.823282064697  -0.23150454352 ]
```

Anstatt die Inverse der Jacobi-Matrix zu berechnen, können wir das Gleichungssystem

$$Df(x_n) \cdot d_n = f(x_n)$$

lösen und dann

$$x_{n+1} = x_n - d_n$$

setzen.

4.52 Satz: *Sei*

$$f : D \to \mathbb{R}^m$$

zweimal stetig differenzierbar, $D \subseteq \mathbb{R}^m$ offen. Falls $Df(x)$ in einer Nullstelle $x \in D$ von f invertierbar ist, so existiert eine Umgebung U von x, so dass das Newton-Verfahren für jeden Startpunkt $x_0 \in U$ gegen $x$konvergiert. Es gibt eine Konstante $c > 0$ mit

$$\|x - x_{n+1}\| \leq c\|x - x_n\|^2.$$

Die Konvergenz ist also quadratisch.

Beweis: Weil f zweimal stetig differenzierbar ist, existiert nach Aufgabe 49 eine Restfunktion $R : D \to \mathbb{R}^m$ mit

$$0 = f(x) = f(x_n) + Df(x_n) \cdot (x - x_n) + R(x)$$

mit

$$R(x) \leq c_1 \cdot \|x - x_n\|^2.$$

Es folgt

$$x_{n+1} = x_n - Df(x_n)^{-1} \cdot f(x_n) = x + Df(x_n)^{-1} \cdot R(x - x_n).$$

Die Inverse von $Df(x_n)$ existiert in einer Umgebung von x, da f stetig differenzierbar ist. Außerdem existiert ein $c_2 > 0$ mit

$$\|Df(\tilde{x})^{-1}\| \leq c_2$$

für $\tilde{x}$ in einer Umgebung $U_r(x)$ von x. Wählt man r so klein, dass $c_1 c_2 r < 1$ gilt, so hat man $x_n \in U_r$ für alle $n \in \mathbb{N}$, und

$$\|x - x_{n+1}\| \leq c_1 c_2 \|x - x_n\|^2.$$

Dies ist die Behauptung. **q.e.d.**

4.53. Definition: Quasi-Newton-Verfahren verwenden Näherungen

$$J_n \approx Df(x_n),$$

so dass also

$$x_{n+1} = x_n - J_n^{-1} f(x_n)$$

ist. Äquivalent dazu ist das Gleichungssystem

$$J_n \cdot \Delta x_n = -f(x_n)$$

mit

$$\Delta x_n = x_{n+1} - x_n.$$

Das Broyden-Verfahren setzt im ersten Schritt

$$J_0 = \frac{1}{\epsilon} \left(f(x_0 + \epsilon e_1) - f(x_0), \ldots, f(x_0 + \epsilon e_n) - f(x_0) \right).$$

Dies ist eine Approximation von $Df(x_0)$ durch Sekantensteigungen. Dann korrigiert es diese Approximation in jedem Schritt mittels

$$J_{n+1} = J_n + \frac{\Delta f_n - J_n \Delta x_n}{\|\Delta x_n\|^2} \cdot \Delta x_n^T = J_n + \frac{f(x_{n+1}) \cdot \Delta x_n^T}{\|\Delta x_n\|^2}.$$

wobei

$$\Delta f_n = f(x_{n+1}) - f(x_n)$$

sei. Das entspricht dem Minimum der Frobenius-Norm $\|J_{n+1} - J_n\|_F$ unter der Nebenbedingung

$$J_{n+1} \Delta x_n = \Delta f_n,$$

wie in der folgenden Aufgabe gezeigt werden soll.

4.54. Beispiel: Die folgende sehr einfache Implementation in EMT zeigt, dass das Verfahren zwar konvergiert, aber nicht so schnell wie das Newton-Verfahren. Das Beispiel entspricht dem obigen Beispiel.

```
>function F(v) := f(v')' // F operiert mit Spaltenvektoren
>function testbroyden ... // Broyden-Verfahren
$  x=[1,1,0]'; y=F(x);
$  d=0.1; M=d*id(3); m1=M[:,1]; m2=M[:,2]; m3=M[:,3];
$  J=(h(x+m1)|h(x+m2)|h(x+m3))-y;
$  J=J/d;
$  n=1;
$  repeat
$    xn=x-inv(J).y; yn=F(xn);
$    dx=xn-x; dy=yn-y;
$    J=J+(yn.dx')/(dx'.dx);
$    c=norm(dy), // druckt c aus!
$    until c<1e-12; // Abbruchkriterium
$    n=n+1;
$    x=xn; y=yn;
$  end;
$  return xn;
$endfunction
>testbroyden, // Gibt Fehlernorm in jedem Schritt aus
 1.45419153275
 0.401317483678
 0.187827378853
 0.0979099536272
 0.0461659211838
 0.0141094298565
 0.00449519986667
 4.54666283138e-005
 6.02778792788e-006
 2.16053484066e-007
 1.13027685106e-010
 0
      0.661468492772
      0.823282064697
      -0.23150454352
```

Die Folge der Approximationen J_n konvergiert gegen $Df(x)$ in der Lösung $f(x) = 0$.

Es gibt jedoch auch andere Quasi-Newton-Verfahren, die Variationen dieser Idee sind.

4.55 Aufgabe: Zeigen Sie, dass für $a, b \in \mathbb{R}^m$ das Minimum von $\|H\|_F$ unter allen $H \in \mathbb{R}^{m\times m}$ mit $Ha = b$ die Matrix

$$H = \frac{b \cdot a^T}{\|a\|^2}$$

ist. Setzen Sie dazu die Lagrange-Bedingung zur Minimierung von

$$F(H) = \|H\|_F$$

unter der Nebenbedingung

$$G(H) = \|Ha - b\|^2 = 0$$

an durch Differenzieren nach $h_{i,j}$ auf, und folgern Sie, dass die i-te Spalte von einem minimalen H ein Vielfaches von a^T ist. Folgern Sie aus der Nebenbedingung, dass H die verlangte Form hat. Begründen Sie, warum ein Minimum existieren muss.

Wenden Sie nun dieses Ergebnis an, um zu zeigen, dass im Broyden-Verfahren $J_{n+1} = J_n + H$ die angegebene Darstellung hat. Beachten Sie

$$J_{n+1}\Delta x_n = \Delta f_n \quad \Leftrightarrow \quad H\Delta x_n = \Delta f_n - J_n \Delta x_n.$$

4.6 Garantierte Einschließungen

4.56 Satz: *Sei $X \subseteq \mathbb{R}^m$ kompakt und konvex, sowie*

$$g : X \to X$$

stetig. Dann hat g einen Fixpunkt in X. Diesen Satz heißt Browerscher Fixpunktsatz

4.57 Aufgabe: Beweisen Sie diesen Satz für den Fall $m = 1$. Für den Fall $m > 1$ muss man erheblich mehr Aufwand betreiben.

4.58 Satz: *Sei $f : X \to \mathbb{R}$ stetig differenzierbar, X ein abgeschlossenes, reelles Intervall, $x \in X$, und*

$$f'(x) \neq 0 \qquad \text{für alle } x \in X.$$

Sei

$$\tilde{X} = x - \frac{f(x)}{f'(X)} = \{x - \frac{f(x)}{f'(\xi)} : \xi \in X\}$$

Für jede Nullstelle $a \in X$ von f gilt dann $a \in \tilde{X}$. Außerdem folgt aus

$$\tilde{X} \subseteq X$$

die Existenz einer Nullstelle in X.

Beweis: Wenn $a \in X$ eine Nullstelle von f ist, so gilt

$$f(x) = f(x) - f(a) = f'(\xi)(x - a),$$

für eine $\xi \in X$, also

$$a = x - \frac{f(x)}{f'(\xi)} \in \tilde{X}.$$

Wir betrachten den Operator

$$g(x) = x - \frac{f(x)}{\frac{f(x) - f(m)}{x - m}}.$$

Dann ist g stetig in X, für ein $\xi_x \in X$ gilt

$$f'(\xi_x) = \frac{f(x) - f(m)}{x - m}.$$

Wir erhalten

$$g(x) = x - \frac{f(x)}{f'(\xi_x)} = m - \frac{f(m)}{f'(\xi_x)}.$$

Falls also $\tilde{X} \subseteq X$ ist, so hat g einen Fixpunkt in X, der dann Nullstelle von f sein muss. **q.e.d.**

Man kann

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

intervallmäßig berechnen. Dabei sei x_n zum Beispiel der Mittelpunkt des Intervalls X_n. Sobald

$$X_{n+1} \subseteq X_n$$

ist, ist die Existenz einer Nullstelle rechnerisch bewiesen. Von diesem Zeitpunkt an kann man

$$X_{n+1} = X_n \cap \left(x_n - \frac{f(x_n)}{f'(X_n)} \right)$$

rechnen. Als Abbruchkriterium eignet sich

$$X_{n+1} = X_n.$$

4.59. Beispiel: Die Implementation der Iteration ist in EMT nicht schwer. Unsere Funktion ergibt einen Fehler, wenn keine Nullstelle gefunden wurde.

```
>function intervalnewton (f,df,X) ... // Intervall-Newton-Verfahren
$  inclusion=false;
$  repeat
$    m=~middle(X)~; Xn=m-f(m)/df(X),
$    until X==Xn; // Abbruchkriterium
$    if not inclusion and Xn<<X then inclusion=true; endif;
$    if inclusion then X=X&&Xn; else X=Xn; endif;
$    X=Xn;
$  end;
$  if not inclusion then error("No inclusion found!"); endif;
$  return X
$endfunction
>intervalnewton("cos(x)-x","-sin(x)-1",~0,1~) // Test
 ~0.7,0.88~
 ~0.7376,0.7414~
 ~0.73908482,0.73908552~
 ~0.739085133215154,0.739085133215168~
 ~0.73908513321516023,0.73908513321516101~
 ~0.73908513321516023,0.73908513321516101~
 ~0.73908513321516023,0.73908513321516101~
```

Eine entsprechende Funktion ist in EMT vordefiniert. Die Ableitung wird von Maxima automatisch berechnet.

```
>mxminewton("cos(x)-x",1) // Ruft Maxima wegen der Ableitung auf
 ~0.73908513321516045,0.73908513321516089~
```

Alternativ können wir die Ableitung auch in einem symbolischen Ausdruck berechnen lassen. Im folgenden Beispiel wird eine Nullstelle der Ableitung von x^x, also ein Minimum der Funktion gefunden.

```
>intervalnewton(&diff(x^x,x),&diff(x^x,x,2),~0.1,0.9~) // diff(x^x,x)=0
 ~0.28,0.48~
 ~0.3644,0.3715~
 ~0.36787903,0.36787985~
 ~0.3678794411714408,0.3678794411714438~
 ~0.36787944117144206,0.36787944117144261~
 ~0.36787944117144206,0.36787944117144261~
 ~0.36787944117144206,0.36787944117144261~
```

Wir können diesen Algorithmus für Gleichungssysteme verallgemeinern. Grundlage dafür ist der folgende Satz.

4.60 Satz: *Sei $\mathcal{X}$ ein Intervallvektor in $\mathbb{R}^m$,*

$$f : \mathcal{X} \to \mathbb{R}^m$$

stetig und in $\mathcal{X}^\circ$ stetig partiell differenzierbar, sowie $\mathcal{L}$ eine Intervallmatrix mit

$$\{Df(x) : x \in \mathcal{X}\} \subseteq \mathcal{L},$$

dir nur invertierbare Matrizen enthalte. Für ein fest gewähltes $x \in \mathcal{X}^\circ$ setzen wir

$$\tilde{\mathcal{X}} = x - \mathcal{L}^{-1} f(x)$$

b mit

$$\mathcal{L}^{-1} = \{L^{-1} : L \in \mathcal{L}\}.$$

Falls f dann eine Nullstelle $a \in \mathcal{X}$ hat, so gilt $a \in \tilde{\mathcal{X}}$. Falls

$$\tilde{\mathcal{X}} \subseteq \mathcal{X}$$

so hat f genau eine Nullstelle $a \in \mathcal{X}$.

Beweis: Sei

$$f(x) = \begin{pmatrix} f_1(x) \\ \vdots \\ f_m(x) \end{pmatrix}.$$

Dann gibt es für alle $x, y \in X$ Punkte $\xi_{i,j} \in X^\circ$ mit

$$\begin{aligned} f_i(y) - f_i(x) &= f_i(y_1, \ldots, y_n) - f_i(x_1, y_2, \ldots, y_n) + \\ &\quad \ldots + f_i(x_1, \ldots, x_{n-1}, y_n) - f_i(x_1, \ldots, x_n) \\ &= \sum_j \frac{\partial f_i}{\partial x_j}(\xi_{i,j})(y_j - x_j). \end{aligned}$$

Es folgt

$$f(x) - f(y) = L_{x,y} \cdot (x - y)$$

für ein $L_{x,y} \in L$. Wenn $a \in \mathcal{X}$ eine Nullstelle von f ist, so haben wir

$$f(x) = L_{x,a} \cdot (x - a),$$

also

$$a = x - L_{x,a}^{-1} \cdot f(x) \in \tilde{\mathcal{X}}.$$

Wir definieren den Operator

$$g(z) = x - L_{x,z}^{-1} f(x).$$

Aufgrund der Konstruktion der Matrizen $L_{x,y}$ kann man $L_{x,y}$ stetig von y abhängen lassen. Denn es gilt für $y_j \neq x_j$ für die i,j-te Komponente von L

$$l_{i,j} = \frac{f_i(y_1, \ldots, y_j, x_{j+1}, \ldots, x_n) - f_i(y_1, \ldots, y_{j-1}, x_j, \ldots, x_n)}{y_j - x_j}.$$

Für $x_j = y_j$ müssen wir dann

$$l_{i,j} = \frac{\partial f_i}{\partial x_j}(y_1, \ldots, y_{j-1}, x_j, \ldots, x_n).$$

setzen. Also ist der Operator $h : \mathcal{X} \to \tilde{\mathcal{X}}$ stetig. Falls $\tilde{\mathcal{X}} \subseteq \mathcal{X}$ ist, so existiert nach dem Browerschen Fixpunktsatz ein Fixpunkt a von h. Also

$$a = x - L_{x,a}^{-1} \cdot f(x),$$

oder

$$f(x) = L_{x,a} \cdot (x - a) = f(x) - f(a).$$

Es folgt f(a)=0. Falls $\tilde{a} \in \mathcal{X}$ eine zweite Nullstelle wäre, so erhalten wir

$$f(b) = 0 = L_{b,a} \cdot (b - a),$$

was der Invertierbarkeit von $L_{b,a} \in \mathcal{L}$ widerspricht. **q.e.d.**

Zur Berechnung benötigt man eine Intervall-Matrix $\mathcal{M}$ mit

$$Df(y)^{-1} \in \mathcal{M} \qquad \text{für alle } y \in X,$$

also eine intervallmäßige berechnete Inverse von $Df(\mathcal{X})$. Diese Matrix ist nur in Ausnahmefällen effektiv zu berechnen. Wir verwenden daher die im folgenden Satz eingeführten Krawzyk-Operatoren.

4.61 Satz: *Zusätzlich zu den Bezeichnungen des vorigen Satzes sei eine Matrix $R \in \mathbb{R}^{m \times m}$ gegeben. Wir setzen dann*

$$\tilde{\mathcal{X}} = x - R \cdot f(x) + (I - R\mathcal{L}) \cdot (\mathcal{X} - x).$$

Falls dann f eine Nullstelle $a \in \mathcal{X}$ hat, so gilt $a \in \tilde{\mathcal{X}}$. Falls

$$\tilde{\mathcal{X}} \in \mathcal{X}^\circ,$$

so ist R invertierbar, sowie alle $A \in L$, und f hat genau eine Nullstelle $a \in \mathcal{X}$.

Beweis: Es gilt für eine Nullstelle $a \in \mathcal{X}$

$$\begin{aligned} a \in \mathcal{X} &= \mathcal{X} - R \cdot (f(x) + f(a) - f(x)) \\ &= \mathcal{X} - R \cdot (f(x) + L_{a,x} \cdot (a - x)) \\ &\in \mathcal{X} - R \cdot (f(x) + \mathcal{L} \cdot (\mathcal{X} - x)) \\ &= \tilde{\mathcal{X}}. \end{aligned}$$

Angenommen $\tilde{\mathcal{X}} \subset \mathcal{X}^\circ$. Sei $L \in \mathcal{L}$. Dann existiert ein $z \in \mathcal{X}$ mit

$$x - R \cdot f(x) + (I - RL) \cdot (z - x) = z.$$

Falls nun RL nicht invertierbar wäre, so existiert ein $w \in \mathbb{R}^m$, $w \neq 0$, mit $RLw = 0$. Also

$$x - R \cdot f(x) + (I - LA) \cdot (z - \lambda w - x) = z - \lambda w.$$

für alle $\lambda \in \mathbb{R}$. Wählt man λ so, dass $z - \lambda x \in \partial\mathcal{X}$, dann muss gelten

$$z - \lambda w \in \tilde{\mathcal{X}} \subseteq \mathcal{X}^\circ.$$

Dies ist ein Widerspruch. Also ist R invertierbar, ebenso wie alle $L \in \mathcal{L}$. Sei nun der Operator

$$h(z) = x - R \cdot f(x) + (I - RL_{x,z}) \cdot (z - x) \in \tilde{X} \subseteq X^\circ.$$

für $z \in \mathcal{X}$ definiert. Dann hat dieser Operator eine Fixpunkt $a \in \tilde{\mathcal{X}}$. Es folgt

$$\begin{aligned} a &= x - R \cdot f(x) + (I - RL_{x,a})(a - x) \\ &= -R \cdot f(x) + a - R \cdot L_{x,a} \cdot a + R \cdot L_{x,a} \cdot x \\ &= -R \cdot f(x) + a + R \cdot L_{x,a} \cdot (x - a) \\ &= a - R \cdot f(x) + R \cdot (f(x) - f(a)) \\ &= a - R \cdot f(a). \end{aligned}$$

Da R invertierbar ist, folgt $f(a) = 0$. Da alle Matrizen $L_{a,b}$ in $\mathcal{L}$ invertierbar sind, kann f analog zum vorigen Beweis keine zwei Nullstellen haben. **q.e.d.**

Die Konvergenz hängt von der Norm $\|I - RL\|$ für Matrizen $L \in \mathcal{L}$ ab. Damit diese Norm klein ist, sollte R eine Näherungsinverse zu allem Matrizen in $\mathcal{L}$ sein.

Die Iterationsvorschrift lautet also

$$\mathcal{X}_{n+1} = x_n - R \cdot f(x_n) + (I - R \cdot Df(\mathcal{X}_n)) \cdot (\mathcal{X}_n - x_n)$$

zum Beispiel mit dem Zentrum $x_n \in \mathcal{X}_n$, und $R = Df(x_n)^{-1}$. Wird dabei R lediglich im ersten Schritt bestimmt und nachher festgehalten, so ist nur lineare Konvergenz zu erwarten. $Df(\mathcal{X}_n)$ muss in der Iteration natürlich intervallmäßig berechnet werden. Die intervallmäßige Einschließung von $Df(\mathcal{X}_n)^{-1}$ ist aber nicht nötig.

Um eine möglichst schnelle Lösung zu erhalten, ist es daher sinnvoll, zunächst das gewöhnliche Newton-Verfahren durchzuführen, bis man zu einer Lösung x_0 kommt. Danach wird das obige Verfahren mit einem Intervall $\mathcal{X}_0$ gestartet, das x_0 enthält. Gewöhnlich muss man nur einen Schritt ausführen, wenn das Intervall um x_0 klein genug gewählt wird.

4.62. Beispiel: Wir kommen auf Beispiel 51 zurück. Die eingebaute Funktion `inewton2` von EMT löst das Problem mit einer garantierten Einschließung der Lösung.

```
>xsol,valid=inewton2("f","Df",[1,1,0]); xsol,
 [ ~0.6614684927715179,0.6614684927715191~
 ~0.8232820646967926,0.8232820646967943~
 ~-0.2315045435202889,-0.2315045435202875~ ]
>valid, // valid==1 bedeutet Einschließung
 1
```

Der Browersche Fixpunktsatz ist zur Argumentation eigentlich nicht nötig. Damit nämlich $\tilde{\mathcal{X}} \subseteq \mathcal{X}^\circ$ gelten kann, muss

$$\|I - RL\| < 1$$

für alle $L \in \mathcal{L}$ gelten. Es genügt daher bei sorgfältigerer Argumentation der Banachsche Fixpunktsatz.

Kapitel 5

Numerische Integration

5.1 Newton-Cotes Formeln

5.1. Definition: In einem Intervall $[a, b]$ geben wir uns Punkte

$$a \leq x_0 < \ldots < x_n \leq b$$

vor. Eine Newton-Cotes Formel für die numerische Integration einer stetigen Funktion $f : [a, b] \to \mathbb{R}$ ist eine Formel der Form

$$\int_a^b f(t)\,dt \approx \sum_{k=0}^{n} a_k f(x_k),$$

die für Polynome vom Grade n exakt ist.

Mit den Lagrangeschen Grundpolynomen (3.14) $L_k(x)$ muss daher gelten

$$\int_a^b L_l(t)\,dt = \sum_{k=0}^{n} a_k L_l(x_k) = a_l.$$

für alle $l = 0, \ldots, n$. Man erhält

$$a_l = \int_a^b \prod_{k \neq l} \frac{t - x_k}{x_l - x_k}\,dt$$

und für äquidistante Punkte

$$x_k = x_0 + ah, \quad h = \frac{b-a}{n}$$

folgt mit der Transformation $t = a + sh$

$$a_l = h \int_0^n \prod_{k \neq i} \frac{s - k}{l - k}\,ds$$

5.2. Beispiel: (0) Für $n = 0$ und $x_0 \in [a, b]$ lautet die Newton-Cotes Formel

$$\int_a^b f(t)\,dt = (b-a)\,f(x_0).$$

Dies ist eine einfache Approximation der Fläche unter der Kurve durch ein Rechteck.

(1) Für $n = 1$ und $x_0 = a$, $x_1 = b$ erhalten wir

$$a_0 = h\int_0^1 \frac{s-0}{1-0}\,ds = \frac{h}{2} = a_1.$$

und daher

$$\int_a^b f(t)\,dt \approx (b-a)\frac{f(a)+f(b)}{2}.$$

Dies ist eine Approximation des Integrals durch ein Trapez. Daher nennt man diese Regel die Trapezregel.

(2) Für $n = 2$ und äquidistante Punkte

$$a = x_0, \quad \frac{a+b}{2} = x_1, \quad b = x_1$$

erhalten wir die Koeffizienten

$$a_0 = \frac{h}{3}, \quad a_1 = \frac{4h}{3}, \quad a_2 = \frac{h}{3}.$$

und die Simpson-Regel

$$\int_a^b f(t)\,dt \approx \frac{b-a}{6}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right).$$

(3) Für $n = 3$ ergibt sich die Pulcherima

$$\int_a^b f(t)\,dt \approx \frac{3h}{8}\left(f(a) + 3f(a+h) + 3f(a+2h) + f(b)\right)$$

mit

$$h = \frac{b-a}{3}.$$

5.3 Aufgabe: Rechnen Sie die Koeffizienten der Simpson-Regel per Hand nach. Rechnen Sie die Koeffizienten der Pulcherrima mit Maxima in EMT nach.

5.4. Beispiel: Zur numerischen Berechnung der Koeffizienten kann man das Gleichungssystem

$$\frac{1}{l+1} = \int_0^1 t^l\,dt = \sum_{k=0}^{n} a_k \left(\frac{k}{n}\right)^l \qquad \text{für } l = 0, \ldots, n$$

verwenden, was dem Spezialfall $[a, b] = [0, 1]$ entspricht.

```
>n=3;
>x=linspace(0,1,n); // x[k]=k/n für k=0..n
>l=0:n; // Polynome vom Grad l <= n
>X=x^l'; // Matrix (k/n)^l für Zeilen k und Spalten l
>b=1/(l+1)'; // Spaltenvektor 1/(l+1)
>fracprint(X\b) // Lösung des Gleichungssystems
       1/8
       3/8
       3/8
       1/8
```

Wir Berechnen die Newton-Cotes Formel für $n = 6$, und integrieren Sie $\sin(x)$ auf dem Intervall $[0, \pi]$ mit dieser Formel. Der erste Teil ist wie oben mit $n = 6$.

```
>a=(X\b)'; fracprint(a*840) // Lösen von Xa=b
 [ 41  216  27  272  27  216  41 ]
>x=linspace(0,pi,6);
>sum(a*sin(x))*pi // Test
 2.00001781364
```

Für $n > 6$ versagt der Gauß-Algorithmus, der zur Lösung des Gleichungssystems verwendet wird, da die Matrix sehr schlecht konditioniert ist. Wir können aber auf den besseren Algorithmus `xlgs` von EMT zurückgreifen.

```
>a=xlgs(X,b)'; // Genaueres Lösen von Xa=b
>x=linspace(0,pi,n);
>sum(a*sin(x))*pi // Test
 2.00000000115
```

5.5 Satz: *Bei zu $(a + b)/2$ symmetrischer Punktewahl sind die Newton-Cotes Formeln bei geradem n exakt für Polynome vom Grad $n + 1$.*

Beweis: Die Abbildung $g : [a, b] \to [a, b]$ definiert durch

$$g(s) = a + b - s$$

ist die Spiegelung von g an der Mitte des Intervalls. Wir haben wegen der Symmetrie

$$g(x_k) = x_{n-k}.$$

Für Polynome $p(s)$ vom Grad n ist $p(g(s))$ wieder ein Polynom n-ten Grades. Es folgt

$$\begin{aligned}
\int_a^b p(t)\,dt &= -\int_b^a p(g(s))\,ds \\
&= \int_a^b p(g(s))\,ds \\
&= \sum_{k=0}^{n} a_k p(g(x_k)) \\
&= \sum_{k=0}^{n} a_k p(x_{n-k}) \\
&= \sum_{k=0}^{n} a_{n-k} p(x_k).
\end{aligned}$$

Dies ist also ebenfalls eine exakte Formel für Polynome n-ten Grades. Es folgt

$$a_k = a_{n-k} \qquad \text{für alle } k = 0, \ldots, n.$$

Da für gerades n die Funktion

$$p(t) = \left(t - \frac{a+b}{2}\right)^{n+1}$$

punktsymmetrisch zu $(b+a)/2$ ist, hat man

$$0 = \int_a^b p(t)\,dt.$$

Andererseits aber auch

$$\sum_{k=0}^{n} a_k p(x_k) = -\sum_{k=0}^{n} a_{n-k} p(x_{n-k}) = -\sum_{k=0}^{n} a_k p(x_k)$$

Also ist diese Summe auch 0. **q.e.d.**

Die Newton-Cotes Formeln für gerades n sind für Polynome vom Grad $n+2$ im Allgemeinen nicht exakt. Die Newton-Cotes Formeln für ungerades n sind für Polynome vom Grad $n+1$ im Allgemeinen nicht exakt.

5.6 Aufgabe: Zeigen Sie, dass die Trapezregel für Polynome vom Grad 2 im Allgemeinen nicht exakt ist.Zeigen Sie, dass weder die Simpsonregel, noch die Pulcherrima für Polynome vom Grad 4 im Allgemeinen exakt sind.

Es folgt, dass die Simpson-Regel bis zum Grad 3 exakt ist, ebenso wie die Pulcherrima. Daher wird die Simpson-Regel bevorzugt verwendet.

5.7. Definition: Die zusammengesetzten Newton-Cotes Formeln entstehen durch Unterteilung des Intervalls in gleich lange Teilintervalle.

5.8. Beispiel: Wir setzen für die Länge der einzelnen Intervalle

$$H = \frac{b-a}{N}.$$

Die zusammengesetzte Trapezregel lautet dann

$$\int_a^b f(t)\,dt = \frac{H}{2}\left(f(a) + 2f(a+H) + \ldots + 2f(b-H) + f(b)\right)$$

Die zusammengesetzte Simpson-Regel wird sehr häufig verwendet und lautet

$$\int_a^b f(t)\,dt = \frac{H}{6}\big(f(a) + 4f(a+H/2) + 2f(a+H) + \ldots \\ + 2f(b-H) + 4f(b-H/2) + f(b)\big).$$

5.9. Beispiel: Die zusammengesetzte Simpson-Regel ist in EMT mit der Funktion `simpson` implementiert. Die Anzahl der Teilintervalle N kann dabei gewählt werden.

```
>simpson("sin(x)",0,pi) // Default: 100 Auswertungen
 2.00000001082
>simpson("sin(x)",0,pi,n=400) // 800 Auswertungen
 2
```

Wir implementieren zum Vergleich die zusammengesetzte Trapezregel und die Simpsonregel. Die Trapezregel hat genausoviele Funktionsauswertungen wie die Simpsonregel, wenn man doppelt so viele Teilintervalle nimmt. Die Simpsonregel ist deutlich besser.

```
>function trapez (f,a,b,n) ... // Trapezregel
$  x=linspace(a,b,n);
$  k=2*ones(1,n+1); k[1]=1; k[n+1]=1; // 1,2,2,2,...,1
$  h=(b-a)/n;
$  return sum(k*f(x))*h/2;
$endfunction
>trapez("sin(x)",0,pi,100) // Test
 1.99983550389
>function overwrite simpson (f,a,b,n) ... // Simpsonregel
$  x=linspace(a,b,2*n);
$  k=4*ones(1,2*n+1);
$  k[1]=1; k[2*n+1]=1; k[3:2:2*n-1]=2; // 1,4,2,4,2,...,2,4,1
$  h=(b-a)/n;
$  return sum(k*f(x))*h/6;
$endfunction
>simpson("sin(x)",0,pi,50) // Test
 2.00000001082
```

5.10 Aufgabe: Vergleichen Sie die Simpsonregel mit der Pulcherrima im obigen Beispiel, bei etwa gleich vielen Funktionsauswertungen.

5.11 Satz: *Eine Newton-Cotes Formel*

$$\int_a^b f(t)\,dt \approx \sum_{k=0}^{n} a_k f(x_k)$$

mit $n+1$ *Punkten*

$$a \le x_0 < \ldots < x_n \le b$$

sei exakt für Polynome vom Grad $m \ge n$*. Dann gilt für* $m+1$*-mal stetig differenzierbare Funktionen* f

$$\left| \int_a^b f(t)\,dt - \sum_{k=0}^{n} a_k f(x_k) \right| \le \frac{h^{m+2}}{2^{2(m-n)-1}(N+1)!}\, \|f^{m+1}\|_{[a,b]}$$

mit $h = b - a$*.*

Beweis: Für $N > n$ ergänzen wir die Punkte durch $N - n$ Nullstellen der transponierten Chebyshev-Polynome zu den Punkten $x_0, \ldots, x_m$. Bezeichne $p(x)$ das Interpolationspolynom N-ten Grades an f in diesen Punkten. Dann gilt

$$\begin{aligned}
\left| \int_a^b f(t)\,dt - \sum_{k=0}^{n} a_k f(x_k) \right| &= \left| \int_a^b f(t)\,dt - \int_a^b p(t)\,dt \right| \\
&\le \int_a^b |f(t) - p(t)|\,dt \\
&\le h\,\frac{f^{m+1}(\xi)}{(m+1)!}\,\|\tilde{\omega}\|_{[}a, b]
\end{aligned}$$

mit

$$\tilde{\omega}(x) = \omega(x)\, t_{m-n}(x),$$

wobei t_{m-n} das transformierte Chebyshev-Polynom, normiert mit höchstem Koeffizienten 1, ist. Also

$$t_{m-n}(x) = \frac{h^{m-n}}{2^{m-n}} \frac{1}{2^{m-n-1}} T_{m-n}\left(\frac{t-(a+b)/2}{(h/2)^{m-n}}\right).$$

Zudem verwenden wir die sehr grobe Abschätzung

$$\|\omega(x)\|_{[a,b]} \leq h^{n+1}.$$

Es folgt die Behauptung. **q.e.d.**

5.12 Satz: *Die zusammengesetzte Simpsonregel $S_N(f,a,b)$ mit N Teilintervallen, also $2N+1$ Funktionsauswertungen, hat den Fehler*

$$\int_a^b f(t)\,dt - S_N(f,a,b) = -\frac{(b-a)^5}{2880\,N^4} f^{(4)}(\xi)$$

mit einem $\xi \in]a,b[$.

Beweis: Wir betrachten zunächst ein Intervall. Für die Fehlerabschätzung interpolieren wir f mit Hermite-Interpolation in

$$x_0 = a, \quad x_1 = x_3 = \frac{a+b}{2}, \quad x_2 = b.$$

Der Interpolationsfehler ist dann

$$f(x) - p(x) = \frac{f^{(4)}(\xi_x)}{4!}\omega(x)$$

mit

$$\omega(x) = (x-x_0)\cdot(x-x_1)^2\cdot(x-x_2)$$

für $x \in [a,b]$. Man zeigt mit Hilfe des Satzes von de l'Hospital, dass

$$f^{(4)}(\xi_x) = 4!\,\frac{f(x)-p(x)}{\omega(x)}$$

stetig von x abhängt. Nach dem Mittelwertsatz der Integralrechnung gibt es wegen $\omega(x) \leq 0$ für $x \in [a,b]$ ein $\xi \in]a,b[$ mit

$$\int_a^b f(t)\,dt - S_1(f,a,b) = \int_a^b (f(t)-p(t))\,dt = \frac{f^{(4)}(\xi)}{4!}\int_a^b \omega(t)\,dt.$$

Schließlich berechnet man

$$\int_a^b \omega(t)\,dt = \frac{(b-a)^5}{2^5}\int_{-1}^1 (s-1)\,s^2\,(s+1)\,ds = -\frac{(b-a)^5}{32\cdot 90}.$$

mit der Substitution

$$t = \frac{a+b}{2} + s\frac{b-a}{2}.$$

Es folgt für ein Intervall

$$\int_a^b f(t)\,dt - S_1(f,a,b) = -\frac{(b-a)^5}{2880} f^{(4)}(\xi).$$

Für N Intervalle ergibt sich der Gesamtfehler

$$-\sum_{k=1}^{N} \frac{(b-a)^5}{2880\,N^5} f^{(4)}(\xi_k).$$

Nun existiert ein $\xi \in]a,b[$ mit

$$f^{(4)}(\xi) = \frac{1}{N}\sum_{k=1}^{N} f^{(4)}(\xi_k).$$

wegen der Stetigkeit von $f^{(4)}$ wegen des Zwischenwertsatzes. Es folgt die Behauptung. **q.e.d.**

5.13 Aufgabe: Erläutern Sie, wie der Zwischenwertsatz im letzten Schritt des obigen Beweises verwendet wurde. Begründen Sie insbesondere, warum man ξ im offenen Inneren des Intervalls wählen kann.

5.14. Beispiel: Für $f(x) = x^4$ müsste diese Abschätzung wegen $f^{(4)} = 4!$ exakt sein.

```
>function f(x) &= x^4;
>simpson("f(x)",1,3,5)-2^5/(2880*5^4)*4! // Approximation + Fehler
 48.4
>&integrate(f(x),x,1,3); %() // Evaluiere exaktes Resultat numerisch
 48.4
```

Es gelingt mit etwas Intervallrechnung auch, sehr exakte Abschätzungen für Integrale zu erhalten. Wir sehen in der folgenden Rechnung allerdings der Einfachheit halber von den Rechenfehlern der Simpsonregel ab. Die Werte der 4-ten Ableitung werden einfach intervallmäßig ermittelt.

```
>function f(x) &= x^x; // zu integrierende Funktion
>function d4f(x) &= diff(f(x),x,4); // vierte Ableitung
>simpson("f(x)",1,1.5,1000)-d4f(~1,2~)*0.5^5/(2880*1000^4)
 ~0.67686327879930641,0.67686327879930741~
```

5.15 Aufgabe: Zeigen Sie für eine Newton-Cotes Formel auf $[a,b]$

$$\sum_{k=0}^{n} a_k = 1.$$

Folgern Sie daraus

$$\left| \int_a^b f(t)\,dt - \sum_{k=0}^{n} a_k f(x_k) \right| \le 2(b-a)\,\omega_{b-a}(f) \sum_{k=0}^{n} |a_k|$$

mit dem Stetigkeitsmodul

$$\omega_\delta(f) = \{|f(x) - f(y)| : |x-y| < \delta,\ x,y \in [a,b]\}$$

Folgern Sie, dass die zusammengesetzten Newton-Cotes Formeln für alle stetigen Funktionen gegen das Integral von f konvergieren.

5.2 Gauß-Quadratur

Die Newton-Cotes Formeln mit $n+1$ Parametern a_k sind für Polynome vom Grad n exakt. Es liegt nahe, durch geschickte Wahl von $x_0, \ldots, x_k$ zusätzliche Exaktheit zu gewinnen. Da dies $n+1$ weitere Parameter sind, erwarten wir den Grad $2n+1$.

In der Tat kann eine Newton-Cotes Formel nicht für Polynome vom Grad $2n+2$ genau sein. Denn müsste sonst gelten

$$0 < \int_a^b \omega(t)^2 \, dt = \sum_{k=0}^{n} a_k \omega(x_k)^2 = 0.$$

mit

$$\omega(x) = (x - x_0) \cdot \ldots \cdot (x - x_n).$$

Der Grad $2n+1$ lässt sich aber wirklich erreichen.

Die Newton-Cotes Formeln kann man auch mit einer positiven Gewichtsfunktion einführen. Die Näherung ist dann

$$\int_a^b f(t) w(t) \, dt = \sum_{k=0}^{n} a_k f(x_k).$$

Analog zum Fall $w = 1$ ergibt sich hier

$$a_k = \int_a^b L_k(t) w(t) \, dt$$

für alle $k = 0, \ldots, n$.

5.16 Satz: *Sei $w(x) > 0$ für alle $x \in [a, b]$ eine positive Gewichtsfunktion, und p_{n+1} das orthogonale Polynom vom Grad $n+1$ bezüglich des Skalarprodukts*

$$\langle f, g \rangle = \int_a^b f(t) g(t) w(t) \, dt.$$

Es gelte also $p_{n+1} \perp \mathcal{P}_n$. Dann hat p_{n+1} alle $n+1$ verschiedene Nullstellen

$$a < x_0 < \ldots < x_n < b$$

in $]a, b[$. Stellt man mit diesen Punkten die n-te Newton-Cotes Formel auf, so ist diese Formel für Polynome vom Grad $2n+1$ exakt. Das heißt es gilt

$$\int_a^b f(t) w(t) \, dt = \sum_{k=0}^{n} a_k f(x_k)$$

für alle $f \in \mathcal{P}_{2n+1}$. Diese Formel nennt man Gauß-Quadraturformel.

Beweis: Angenommen p_{n+1} hat nur $m < n+1$ Nullstellen

$$a < x_1 < \ldots < x_m < b$$

in $]a, b[$, in denen p_{n+1} das Vorzeichen wechselt. Dann definieren wir

$$q(t) = (x - x_1) \cdot \ldots \cdot (x - x_m).$$

Es folgt

$$0 = \langle p_{n+1}, q \rangle = \int_a^b p_{n+1}(t) q(t) w(t)\, dt$$

Dies ist aber unmöglich, weil $p_{n+1} q$ das Vorzeichen nicht wechselt. Also hat p_{n+1} in der Tat $n+1$ einfache Nullstellen

$$a < x_0 < \ldots < x_n < b.$$

Wir bestimmen die Koeffizienten a_k der Newton-Cotes Formel in diesen Punkten unter Berücksichtigung der Gewichtsfunktion w. Sei nun $q \in \mathcal{P}_{2n+1}$. Dann kann man p_{n+1} aus q ausdividieren und erhält

$$q = p_{n+1} h + r$$

mit $h, r \in \mathcal{P}_n$. Es gilt dann einerseits

$$\int_a^b q(t) w(t)\, dt = \int_a^b p_{n+1}(t) h(t) w(t)\, dt + \int_a^b r(t)\, dt = \int_a^b r(t)\, dt,$$

und andererseits

$$\sum_{k=0}^{n} a_k q(x_k) = \sum_{k=0}^{n} a_k (p_{n+1}(x_k) h(x_k) + r(x_k)) = \sum_{k=0}^{n} a_k r(x_k).$$

Da die Formel exakt für $r \in \mathcal{P}_n$ ist, folgt

$$\int_a^b q(t) w(t) = \sum_{k=0}^{n} a_k q(x_k)$$

und damit die Behauptung. **q.e.d.**

5.17 Aufgabe: Sei $w = 1$ und $[a, b] = [-1, 1]$. Zeigen Sie, dass die Polynome

$$p_n(x) = \frac{d^n}{dx^n} \left[(x^2 - 1)^n \right]$$

orthogonal bezüglich des entsprechenden Skalarprodukts sind. Man nennt diese Polynome Legendre-Polynome. Beachten Sie, dass Sie nur für gerades $n + m$

$$\langle p_n, p_m \rangle = 0$$

zeigen müssen.

5.18. Beispiel: Wir berechnen mit Maxima in EMT das 11-te Legendre-Polynom, und mit EMT numerisch seine Nullstellen.

```
>function p(x) &= expand(diff((x^2-1)^10,x,10))

                     10                  8                  6
        670442572800 x   - 1587890304000 x  + 1307674368000 x
                                        4               2
                         - 435891456000 x  + 50295168000 x  - 914457600

>p &= makelist(coeff(p(x),x,k),k,0,10) // generiere Liste der Koeffizienten

        [- 914457600, 0, 50295168000, 0, - 435891456000, 0,
                    1307674368000, 0, - 1587890304000, 0, 670442572800]

>xn=sort(real(polysolve(p()))) // finde alle Nullstellen des Polynoms p
 [ -0.973906528517  -0.865063366689  -0.679409568299  -0.433395394129
 -0.148874338982  0.148874338982  0.433395394129  0.679409568299
 0.865063366689  0.973906528517 ]
```

Danach berechnen wir mit dem schon bekannten Code die Koeffizienten der Newton-Cotes Formel.

```
>n=9;
>l=(0:n)'; // Polynome vom Grad l <= n
>X=xn^l; // Matrix (k/n)^l für Zeilen k und Spalten l
>b=(1-(-1)^(l+1))/(l+1); // Integral von x^l auf [-1,1]
>fracprint(b') // Test
 [ 2  0  2/3  0  2/5  0  2/7  0  2/9  0 ]
>a=xlgs(X,b)' // Lösung des Gleichungssystems
 [ 0.0666713443087  0.149451349151  0.219086362516  0.26926671931
 0.295524224715  0.295524224715  0.26926671931  0.219086362516
 0.149451349151  0.0666713443087 ]
```

In der Tat ist diese Formel für Polynome bis zum Grad $2n+1 = 19$ exakt.

```
>sum(a*xn^18); fracprint(%)
 2/19
>sum(a*xn^19),
 0
```

Die Funktion gauss in EMT führt diese Integration auf beliebigen Intervallen durch, wobei das Intervall in Teilintervalle aufgeteilt werden kann.

```
>longestformat;
>gauss("x*sin(x)",0,pi,n=1)
       3.141592653589793
>pi
       3.141592653589793
>gauss("exp(-x^2)/sqrt(pi)",0,10)
      0.4999044573862593
>gauss("exp(-x^2)/sqrt(pi)",0,10,n=10)
      0.5000000000000001
```

Die Genauigkeit ist sehr gut. Das letzte Integral wurde mit nur 100 Funktionsauswertungen berechnet.

Der Satz lässt sich auch auf unendliche Intervalle I ausdehnen, sofern die Gewichtsfunktion stark genug fällt.

5.19 Aufgabe: Zeigen Sie, dass die Polynome

$$p_n(x) = e^x \frac{d^n}{dx^n}\left(x^n e^{-x}\right)$$

in der Tat Polynome sind, und dass sie orthogonal stehen bezüglich des Skalarprodukts

$$\langle f, g\rangle = \int_0^\infty f(t)g(t)e^{-t}\,dt.$$

Die mit $1/n!$ skalierten Polynome nennt man Laguerre-Polynome. Berechnen Sie p_{10}, die Nullstellen von p_{10} und die Koeffizienten der Newton-Cotes Formel. Beachten Sie dabei, dass

$$\int_0^\infty t^k e^{-t}\,dt = k!$$

gilt. Wenden Sie das Ergebnis zur Approximation von

$$\int_0^\infty \sin(t)e^{-t}\,dt = \frac{1}{2}$$

an.

5.20 Satz: *Für die Gauß-Quadraturformel gilt*

$$\int_a^b f(t)w(t) - \sum_{k=0}^{n} a_k f(x_k) = \frac{f^{(2n+2)}(\xi)}{(2n+2)!}\int_a^b \omega(t)^2 w(t)\,dt$$

mit

$$\omega(x) = (x - x_0)\cdot\ldots\cdot(x - x_n),$$

also $\omega(x) = \rho_n p_n(x)$.

Beweis: Wir interpolieren f durch $p \in \mathcal{P}_{2n+1}$ in den Punkten

$$x_0, x_0, x_1, x_1, \ldots, x_n, x_n$$

im Hermiteschen Sinn. Es folgt die Behauptung genau wie im Beweis von Satz 12. **q.e.d.**

5.21 Aufgabe: Zeigen Sie für die Legendrepolynome

$$p_n(x) = \frac{d^n}{dx^n}\left[(x^2-1)^n\right]$$

die Gleichung

$$\int_{-1}^{1} p_n(t)^2\,dt = (2^n n!)^2\,\frac{2}{2n+1}$$

Zeigen Sie für diesen Fall

$$\rho_n = \frac{n!}{(2n)!}.$$

Zeigen sie daraus eine Fehlerformel für die Gauß-Quadratur auf $[-1, 1]$ mit $w = 1$ ab.

5.22. Beispiel: Wir können diese Fehlerabschätzung sehr leicht in EMT für ein gegebenes n herleiten.

```
>function omega(x) &= expand(diff((x^2-1)^10,x,10)/20!*10!)

                     8        6        4        2
              10   45 x    630 x    210 x    315 x     63
             x   - ----- + ------ - ------ + ------ - -----
                    19       323      323      4199    46189

>&integrate(omega(x)^2,x,-1,1); c=%() // Integral von om^2 auf [-1,1]
 2.92559033074e-006
>gauss("x^20",-1,1,n=1)+c // n=9 angewendet auf x^20 plus Fehler
 0.0952380952381
>2/21 // Exaktes Ergebnis
 0.0952380952381
```

5.23. Beispiel: Im Fall $w = 1$ kann man die Gauß-Quadratur sehr leicht vom Intervall $[-1, 1]$ auf ein Intervall $[a, b]$ übertragen, indem man die Knoten linear gemäß

$$\tilde{x}_k = \frac{a+b}{2} + x_k \frac{b-a}{2}$$

verschiebt, und die Gauß-Quadratur mit

$$\tilde{a}_k = \frac{b-a}{2} a_k$$

verwendet. Der Fehler ist dann

$$\int_a^b f(t)w(t)\,dt - \sum_{k=0}^{n} \tilde{a}_k f(\tilde{x}_k) = \frac{f^{(2n+2)}(\xi)}{(2n+2)!} \left(\frac{b-a}{2}\right)^{2n+3} \int_a^b \omega(t)^2 w(t)\,dt$$

mit ω aus dem Intervall $[-1, 1]$.

5.24. Beispiel: Mit dem im vorigen Beispiel hergeleiteten c können wir also eine sehr genaue Einschließung von

$$\int_0^1 e^{-t^2}\,dt$$

gewinnen. Wir vernachlässigen wieder die Fehler bei der Auswertung der Gauß-Quadraturformel.

```
>longestformat;
>Ig=gauss("exp(-x^2)",0,1,n=1)
       0.746824132812427
>d20f &= diff(exp(-x^2),x,20); // 20. Ableitung von exp(-x^2)
>Ig+d20f(~0.1~)/20!*(1/2)^21*c // Integral + Intervall-Fehler
  ~0.74682413281242688,0.7468241328124271~
```

Maxima kann dieses Integral auf beliebig viele Stellen berechnen, da das benötigte Gaußsche Fehlerintegral dort definiert ist. Das Ergebnis liegt in unserem Intervall.

```
>&integrate(exp(-x^2),x,0,1) | bfloat

                  7.46824132812427025399946743613185b-1
```

5.25 Satz: *Sei $p_n \in \mathcal{P}_n$, $n \in \mathbb{N}_0$ eine Folge von Polynomen mit höchstem Koeffizienten 1, die orthogonal bezüglich eines Skalarprodukts*

$$\langle f, g\rangle = \int_a^b f(t)g(t)w(t)\,dt$$

seien. Dann existieren Folgen von Koeffizienten $b_n, c_n \in \mathbb{R}$ mit

$$p_{n+1}(x) = (x + b_n)p_n(x) + c_n p_{n-1}(x)$$

für alle $n \in \mathbb{N}$, $x \in \mathbb{R}$.

Beweis: Wir definieren

$$q_n(x) = p_{n+1}(x) - xp_n(x).$$

Da p_{n+1} und p_n den höchsten Koeffizienten 1 haben, folgt $q_n \in \mathcal{P}_n$. Da $p_0, \ldots, p_n$ eine Basis von $\mathcal{P}_n$ bilden, existieren Koeffizienten mit

$$q_n = \alpha_n p_n + \alpha_{n-1}p_{n-1} + \ldots + \alpha_0 p_0.$$

Für $0 \le k \le n-2$ gilt

$$\langle q_n, p_k\rangle = \langle p_{n+1}, p_k\rangle - \langle xp_n, p_k\rangle = \langle p_{n+1}, p_k\rangle - \langle p_n, xp_k\rangle = 0$$

wegen der Orthogonalität. Es folgt

$$0 = \langle q_n, p_k\rangle = \alpha_k\langle p_k, p_k\rangle.$$

Also $\alpha_k = 0$ für alle $k = 0, \ldots, n-2$. Mit

$$\alpha_n = b_n, \quad \alpha_{n-1} = c_n$$

folgt die Behauptung. **q.e.d.**

5.26. Beispiel: b_n und c_n sind lediglich durch die höchsten Koeffizienten von p_{n+1}, p_n und p_{n-1} festgelegt. Für die Legendre-Polynome mit höchstem Koeffizienten 1 folgt

$$\begin{aligned} p_n(x) &= \frac{n!}{(2n)!}\frac{d^n}{dx^n}\left[(x^2-1)^n\right] \\ &= \frac{n!}{(2n)!}\frac{d^n}{dx^n}\left[x^{2n} - nx^{2n-2} + \ldots\right] \\ &= x^n - \frac{n(n-1)}{2(2n-1)}x^{n-2} + \ldots \end{aligned}$$

Es folgt $b_n = 0$ und

$$c_n - \frac{n(n-1)}{2(2n-1)} = -\frac{(n+1)n}{2(2n+1)}$$

Also lautet die Rekursionsformel für die Legendre-Polynome mit höchstem Koeffizienten 1

$$p_{n+1}(x) = xp_n(x) - \frac{n^2}{4n^2-1}p_{n-1}(x).$$

5.27 Aufgabe: Wir definieren die Hermiteschen Polynome

$$H_n(x) = (-1)^n e^{x^2} \frac{d^n}{dx^n}\left(e^{-x^2}\right).$$

Berechnen Sie $H_0, \ldots, H_3$, sowie deren Nullstellen. Zeigen Sie

$$H_{n+1}(x) = 2xH_n(x) - H_n'(x).$$

Folgern Sie daraus per Induktion

$$H_n'(x) = 2nH_{n-1}(x),$$

sowie die Rekursionsformel

$$H_{n+1}(x) = 2xH_n(x) - 2nH_{n-1}(x).$$

Zeigen Sie, dass Hermitesche Polynome orthogonal bezüglich des Skalarprodukts

$$\langle f, g\rangle = \int_{-\infty}^{\infty} f(t)g(t)e^{-t^2}\,dt$$

sind. Setzen Sie dazu

$$G_m(x) = e^{-x^2}H_m(x)$$

und zeigen Sie durch partielle Integration

$$\int_{-\infty}^{\infty} H_n(t)G_m(t)\,dt = 2n\int_{-\infty}^{\infty} H_{n-1}(t)G_{m-1}(t)\,dt.$$

5.3 Das Romberg-Verfahren

5.28. Definition: Das Romberg-Verfahren verwendet die Trapezregel $T(h)$ mit verschiedenen Schrittweiten $h_0, \ldots, h_m$, und interpoliert T mit einem Polynom $p \in \mathcal{P}_m$ in h^2, also

$$p(h_k^2) = T(h_k) \qquad \text{für } k = 0, \ldots, m$$

Als Approximation für das Integral wird

$$\int_a^b f(t)\,dt \approx p(0)$$

genommen.

Die Idee, die Werte $T(h_0), \ldots, T(h_m)$ zu interpolieren, und so den Grenzwert

$$T(0) = \lim_{h\to 0} T(h)$$

zu ermitteln, ist naheliegend. Es ist aber nicht so einfach zu begründen, warum dazu Polynome der Form $p(h^2)$ günstig sind. Um das zu begründen benötigen wir die Eulersche Summenformel

$$\frac{g(0)}{2} + g(1) + \ldots + g(n-1) + \frac{g(n)}{2} - \int_0^n g(t)\,dt \\ = \sum_{k=1}^{m} a_k \left(g^{(2k-1)}(n) - g^{(2k-1)}(0) \right) + R_m,$$

dir für $2m+2$ mal stetig differenzierbares g auf $[0,n]$ gilt. Dabei ist

$$a_k = \frac{B_{2k}(0)}{(2k)!}$$

mit den Bernoullischen Polynomen, die durch $B_0(x) = 1$ und

$$B_k(x) = B_k(0) + k \int_0^x B_{k-1}(t)\,dt$$

für $k \in \mathbb{N}$ definiert sind. Der Fehler der Formel ist

$$R_m = \frac{1}{(2k+2)!} \int_0^n \left(s_{2m+2}(n) - s_{2m+2}(0) \right) g^{(2m+2)}(t)\,dt$$

mit der stückweise definierten Funktion

$$s_k(x) = B_k(x - l), \qquad \text{für } x \in [l, l+1].$$

Wir geben diese Formel hier ohne Beweis wieder. Insbesondere gilt

$$R_m = 0 \qquad \text{für } g \in \mathcal{P}_{2m+1}.$$

Für $g \in \mathcal{P}_{2m+1}$ setzen wir nun $f(x) = g(nx)$f ür $x \in [0,1]$ und erhalten für $f \in \mathcal{P}_{2m+1}$

$$\frac{f(0)}{2} + f(1/n) + \ldots + f(1-1/n) + \frac{f(1)}{2} - n \int_0^1 f(t)\,dt \\ = \sum_{k=1}^{m} a_k \frac{1}{n^{2k-1}} \left(f^{(2k-1)}(1) - f^{(2k-1)}(0) \right)$$

Mit $h = 1/n$ haben wir also

$$h \left(\frac{f(0)}{2} + f(h) + \ldots + f(1-h) + \frac{f(1)}{2} \right) - \int_0^1 f(t)\,dt = \sum_{k=1}^{m} b_k h^{2k}.$$

Mit Koeffizienten $b_k \in \mathbb{R}$, die nicht von m abhängen. Für das Intervall $[0,1]$ ergibt das eine Entwicklung der Form

$$T(h) = b_0 + b_1 h^2 + b_2 h^4 + \ldots$$

Dies begründet die Verwendung von Polynomen in h^2.

Man kann die Simpsonregel als ein Romberg-Verfahren mit Schrittweiten

$$h_0 = \frac{b-a}{n}, \quad h_1 = \frac{b-a}{2n}$$

auffassen. Denn in diesem Fall gilt für das Interpolationspolynom

$$p(h^2) = T(h_0) + \frac{T(h_1) - T(h_0)}{h_1^2 - h_0^2}(h^2 - h_0^2).$$

Folglich wegen $h_0 = 2h_1$

$$p(0) = T(h_0) + \frac{4}{3}(T(h_1) - T(h_0)) = \frac{4}{3}T(h_1) - \frac{1}{3}T(h_0).$$

Beachtet man nun noch

$$T(h_0) = \frac{h_0}{2}\left(f(a) + 2f(a+h_0) + \ldots + 2f(b-h_0) + f(b)\right)$$
$$T(h_1) = \frac{h_0}{4}\left(f(a) + 2f(a+h_0/2) + 2f(a+h_0) + \ldots + f(b)\right),$$

so erhält man

$$p(0) = \frac{h_0}{6}(f(a) + 4f(a+h_0/2) + 2f(a+2h_0) + \ldots + f(b)),$$

was der Simpsonregel entspricht.

Wählt man in jedem Schritt die halbe Schrittweite, also

$$h_0, \quad \frac{h_0}{2}, \quad \frac{h_0}{4}, \quad \frac{h_0}{8}, \quad \ldots,$$

so kann man die schon berechneten Funktionswerte von f wiederverwenden.

Da wir in jedem Schritt nur an dem Interpolationswert $p(0)$ interessiert sind, ist die Interpolation nach Neville möglich. Diese beruht auf der Darstellung

$$p_{k,\ldots,k+m}(x) = \frac{p_{k+1,\ldots,k+m}(x)\cdot(x - x_k) + p_{k,\ldots,k+m-1}(x)\cdot(x_{k+m} - x)}{x_{k+m} - x_k}.$$

des interpolationspolynoms in den Punkten $x_k, \ldots, x_{k+m}$ aus Übung 3.22. Wir setzen

$$T_{k,0} = T(h_0) \qquad \text{für } k = 0, \ldots, m.$$

und für $l = 1, \ldots, m$

$$T_{k-1,l} = T_{k,l-1} + \frac{T_{k,l-1} - T_{k-1,l-1}}{\frac{h_{l-k}^2}{h_l^2} - 1} \qquad \text{für } k = 1, \ldots, m-l.$$

Das Ergebnis steht dann in $T_{0,m}$. Die Zahlen lassen sich in einem Dreiecksschema

$$\begin{array}{ccccccc}
T_{0,0} & & & & & & \\
 & T_{0,1} & & & & & \\
T_{1,0} & & T_{0,2} & & & & \\
 & T_{0,2} & & \ddots & & & \\
\vdots & & & & T_{0,m} & & \\
 & T_{m-2,1} & & \iddots & & & \\
T_{m-1,0} & & T_{m-2,2} & & & & \\
 & T_{m-1,1} & & & & & \\
T_{m,0} & & & & & &
\end{array}$$

berechnen. Zur Erweiterung des Dreiecksschemas um $T_{m+1,0} = T(h_{m+1})$ benötigt man die letzte Zeile dieses Schemas, die man also speichern muss.

5.29. Beispiel: Das Verfahren ist in EMT vorprogrammiert. Es stoppt, wenn die Differenz $T_{m,0} - T_{m+1,0}$ klein genug wird.

```
>romberg("sin(x)",0,pi)
 2
```

Um das Verfahren mit dem Gauß-Verfahren zu vergleichen, definieren wir eine Funktion `fcount`, die eine andere Funktion aufruft und dabei zählt, wie oft die Funktion f tatsächlich ausgewertet wird. Dabei muss die Auswertung von EMT elementweise auf Matrizen berücksichtigt werden.

```
>function f(x) &= x*exp(x^2); // eigentliche Funktion
>function fcount(x) ... // Zählfunktion
$  global count;
$  count=count+prod(size(x)); // erhöhe um Zeilen*Spalten von x
$  return f(x)
$endfunction
```

Beim Vergleich stellen wir in diesem Beispiel fest, dass Gauß ein besseres Ergebnis mit ebenso vielen Funktionsauswertungen liefert.

```
>longestformat;
>count=0; romberg("fcount(x)",0,1), count,
      0.8591409142296098
                     161
>&integrate(x*exp(x^2),x,0,1)|bfloat

                 8.591409142295226176801437356763 3b-1

>count=0; gauss("fcount(x)",0,1,16), count,
      0.8591409142295227
                     160
```

Das Romberg-Verfahren scheitert allerdings an

$$\int_0^1 \sqrt{t}\, dt,$$

da die Voraussetzungen der Differenzierbarkeit nicht erfüllt sind. Das Gauß-Verfahren mit 1000 Funktionsauswertungen liefert immerhin ein angenähertes Ergebnis.

```
>romberg("sqrt(x)",0,1)
 Stack overflow!
 Error in sqrt
 Error in function %evalexpression

 Error in :
        if maps then return %mapexpression1(x,f);
 ^
 Error in function %evalexpression
 Error in function romberg
>gauss("sqrt(x)",0,1,100)
      0.6666667560429368
```

Kapitel 6

Splines

6.1 Bezier-Kurven

Bezier-Kurven sind ein Spezialfall von Splines mit mehrfachen Knoten, der aber für sich alleine interessant ist.

6.1. Definition: Für Punkte

$$P_0, \dots, P_n \in \mathbb{R}^m$$

definieren wir die Bezierkurve

$$B_{P_0,\dots,P_n}(t) = \sum_{k=0}^{n} B_{k,n}(t) P_k$$

für $t \in [0,1]$, wobei

$$B_{k,n}(t) = \binom{n}{k} t^k (1-t)^{n-k}$$

die Bernstein-Polynome sind.

6.2. Beispiel: Es gilt

$$\begin{aligned} B_{0,3}(t) &= (1-t)^3, \\ B_{1,3}(t) &= 3t(1-t)^2, \\ B_{2,3}(t) &= 3t^2(1-t), \\ B_{3,3}(t) &= t^3. \end{aligned}$$

Mit Punkten

$$\begin{aligned} P_0 &= (0,0), \\ P_1 &= (0,1), \\ P_2 &= (1,1), \\ P_3 &= (1,0) \end{aligned}$$

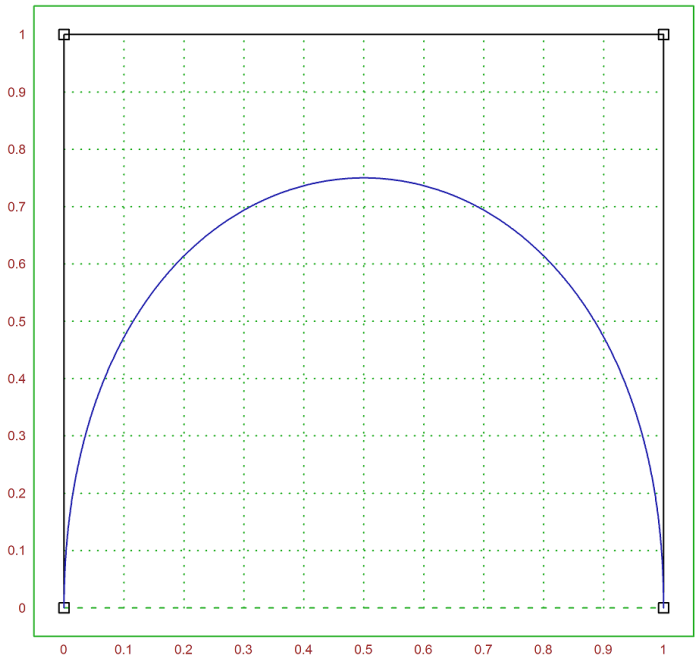

Abbildung 6.1: Bezierkurve mit Kontrollpolygon

ergibt sich eine Kurve von P_0 nach P_3 mit den Gleichungen

$$B_{P_0,P_1,P_2,P_3}(t) = \begin{pmatrix} 3t^2(1-t)+t^3 \\ 3t(1-t)^2+3t^2(1-t) \end{pmatrix} = \begin{pmatrix} t^2(3-2t) \\ 3t(1-t)(2-t) \end{pmatrix}.$$

Diese spezielle Kurve können wir in EMT recht einfach plotten. Wir fügen noch die Punkte P_0, P_1, P_2, P_3 dazu. Den Streckenzug durch diese Punkte bezeichnet man als Kontrollpolygon.

```
>t=linspace(0,1,1000);
>plot2d([0,0,1,1],[0,1,1,0]);
>plot2d([0,0,1,1],[0,1,1,0],>points,>add);
>plot2d(t^2*(3-2*t),3*t*(1-t),color=blue,>add);
```

Man kann die Matrixsprache von EMT effektiv ausnutzen, um die Bezier-Kurve zu plotten. Dazu erzeugen wir die Matrix

$$B = \left(\binom{n}{k} x_j^k (1-x_j)^{n-k} \right)_{k,j}$$

für Punkte

$$0 = x_1 < \ldots < x_l = 1.$$

Dies geschieht durch Kombination des Spaltenvektors $k = (0, \ldots, n)^T$ mit dem Zeilenvektor x. Die Punkte werden in einer Matrix

$$P = (P_0, \ldots, P_n)$$

gespeichert. Die Koordinaten der Kurve stehen dann in der Matrix $P \cdot B$.

Die folgenden Zeilen erzeugen denselben Plot wie im vorigen Beispiel.

```
>function bezier (x, P) ...
$  n=cols(P)-1; k=(0:n)';
$  B=bin(n,k)*x^k*(1-x)^(n-k);
$  return P.B;
$endfunction
>P=[0,0,1,1;0,1,1,0]; fracprint(P)
         0         0         1         1
         0         1         1         0
>t=linspace(0,1,1000); b=bezier(t,P);
>plot2d(b[1],b[2],a=0,b=1,c=0,d=1);
```

Es ist nicht effektiv, die Funktion `bin` und die Potenzen in EMT für jeden Punkt $B_k(x_j)$ erneut aufzurufen. Statt dessen ist es besser die Rekursion

$$B_{k,n}(x) = B_{k-1,n}(x)\,\frac{(n-k+1)x}{k(1-x)}$$

zu verwenden.

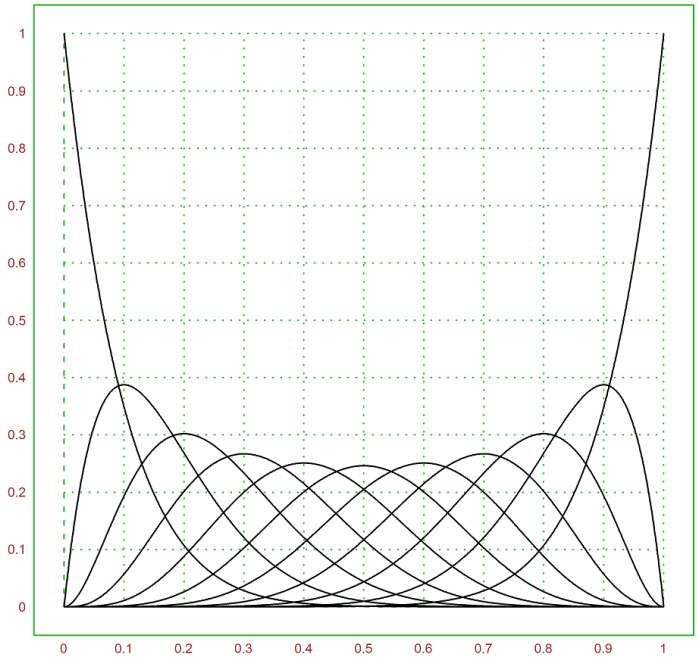

Abbildung 6.2: Bernsteinpolynome $B_{k,10}$

6.3. Beispiel: Die Matrix B lässt sich in EMT direkt wie folgt berechnen. Wir können auf diese Weise alle Bernstein-Polynome schnell plotten.

```
>x=linspace(0,1,1000); n=10; k=(0:10)';
>B=bin(n,k)*x^k*(1-x)^(n-k);
>plot2d(x,B);
```

Statt dessen ist aber auch folgende Konstruktion möglich. Dabei wird das kumulative Produkt der Werte

$$(1-x)^{10}, \quad \frac{10x}{1-x}, \quad \frac{9x}{2(1-x)}, \quad \frac{8x}{3(1-x)}, \quad \dots$$

verwendet.

```
>x=linspace(1/1000,1-1/1000,1000)'; n=10; k=1:10;
>B=cumprod((1-x)^10|(x*(n-k+1)/((1-x)*k)))';
>plot2d(x',B);
```

Die Bernsteinpolyonme sind uns schon im Beweis des Satzes von Weierstraß 3.73 begegnet. Dort war

$$P_k = f(k/n),$$

und wir haben bewiesen, dass für stetiges $f : [0,1] \to \mathbb{R}$

$$B_{P_0,\dots,P_n}(t) \to f(t)$$

gleichmäßig konvergiert. Daraus folgt dass für eine Kurve

$$\gamma : [0,1] \to \mathbb{R}^m$$

die gleichmäßige Konvergenz

$$B_{\gamma(0),\gamma(1/n),\dots,\gamma(1)}(t) \to \gamma(t)$$

gilt.

6.4 Aufgabe: Zeigen Sie

$$B_{P_0,\dots,P_n}(0) = P_0, \quad B_{P_0,\dots,P_n}(1) = P_n,$$

sowie

$$B'_{P_0,\dots,P_n}(0) = n(P_1 - P_0), \quad B'_{P_0,\dots,P_n}(1) = n(P_n - P_{n-1}).$$

Woran erkennt man dieses Ergebnis in der Abbildung des obigen Beispiels?

6.5 Aufgabe: Zeigen Sie

$$\sum_{k=0}^{n} B_k(x,t) = 1 \qquad \text{für alle } t \in \mathbb{R}.$$

Folgern Sie, dass die Bezier-Kurve ganz in der konvexen Hülle der Kontrollpunkte verläuft.

Kubische Bezierkurven der Form

$$B_{P_0,P_1,P_2,P_3}(t)$$

werden gerne zur graphischen Modellierung von Kurven verwendet. Dabei werden die Kurven stückweise aneinander gehängt. Um eine differenzierbare Verbindung zu erreichen, wählt man Punkte

$$P_0, P_1, P_2, P_3, \tilde{P}_0, \tilde{P}_1, \tilde{P}_2, \tilde{P}_3$$

mit

$$P_3 = \tilde{P}_0, \quad P_3 - P_2 = \tilde{P}_1 - \tilde{P}_0.$$

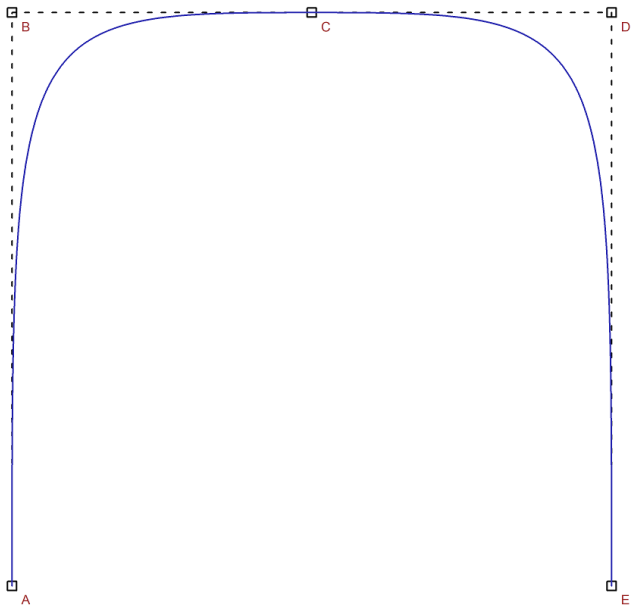

Abbildung 6.3: Zwei zusammengesetzte Bezierkurven

Die Kurve kann dann durch Verschieben von $P2, P3 = \tilde{P}_0, \tilde{P}_1$ modelliert werden. Um die Differenzierbarkeit zu erhalten, muss $P2$ und $\tilde{P}_1$ simultan verschoben werden. Bei der Verschiebung von $P_3 = \tilde{P}_0$ muss P_2 und $\tilde{P}_1$ parallel mit verschoben werden.

6.6. Beispiel: Wir wählen die Punkte

$$A = P_0, \quad B = P_1, \quad D = P_2, \quad E = P_3$$

aus dem vorigen Beispiel, fügen aber noch den Mittelpunkt C der Strecke BD hinzu. Nun zeichnen wir zwei Bezierkurven mit Kontrollpunkten A, B, B, C und C, D, D, E. Die zusammengesetzte Kurve ist wegen $C - B = D - C$ differenzierbar in C.

```
>A=[0,0]'; B=[0,1]'; D=[1,1]'; E=[1,0]';
>C=(B+D)/2;
>P=A|B|C|D|E; plot2d(P[1],P[2],>points,grid=0,);
>plot2d(P[1],P[2],style="--",>add);
>x=linspace(0,1,1000);
>b=bezier(x,A|B|B|C); plot2d(b[1],b[2],>add,color=blue);
>b=bezier(x,C|D|D|E); plot2d(b[1],b[2],>add,color=blue);
>label("A",A[1],A[2]);
>label("B",B[1],B[2]);
>label("C",C[1],C[2]);
>label("D",D[1],D[2]);
>label("E",E[1],E[2]);
```

6.7 Aufgabe: Zeigen Sie

$$B_{P_0,\dots,P_n}(t) = (1-t)B_{P_0,\dots,P_{n-1}}(t) + tB_{P_1,\dots,P_n}(t)$$

für $t \in \mathbb{R}$.

6.8 Aufgabe: Sei für $t \in [0,1]$

$$\tilde{P}_k(t) = (1-t)P_k + tP_{k+1} \qquad \text{für } k = 0, \ldots, n-1.$$

Zeigen Sie

$$B_{P_0,\ldots,P_n}(t) = B_{\tilde{P}_0(t),\ldots,\tilde{P}_{n+1}(t)}(t)$$

für $t \in [0,1]$.

$B_{P_0,\ldots,P_n}(t)$ lässt sich daher auch mit einem interessanten Teilungsalgorithmus berechnen. Wir bestimmen für festes $t \in [0,1]$ ein Dreiecksschema

$$\begin{array}{ccccccc}
P_0 = P_{0,0} & & & & & & \\
 & & P_{0,1} & & & & \\
P_1 = P_{1,0} & & & \ddots & & & \\
\vdots & & & & & P_{0,n} & \\
P_{n-1} = P_{n-1,0} & & & & \iddots & & \\
 & & P_{n-1,1} & & & & \\
P_n = P_{n,0} & & & & & &
\end{array}$$

mit Punkten

$$P_{k,l} = (1-t)P_{k,l-1} + tP_{k+1,l}$$

für Schritte $l = 0, \ldots, n$ und $k = 0, \ldots, n-l$. Dann gilt nach obigen Aufgabe

$$B_{P_0,\ldots,P_n}(t) = P_{0,n}.$$

6.9 Aufgabe: Zeichnen Sie die Punkte $P_{k,l}$ für $t = 1/2$ für die Ausgangspunkte im Beispiel 2.

6.10 Satz: *Mit den in der vorigen Bemerkung eingeführten Bezeichnungen gilt*

$$B_{P_{0,0},\ldots,P_{0,n}}(s) = B_{P_0,\ldots,P_n}(st),$$

sowie

$$B_{P_{0,n},P_{1,n-1},\ldots,P_{n,0}}(s) = B_{P_0,\ldots,P_n}((1-s)t + s),$$

so dass damit die Bezierkurve in zwei Bezierkurven zerlegt wird, die auf den Intervallen $[0,t]$ *und* $[t,1]$ *definiert sind.*

Beweis: Wir verwenden Induktion nach der Anzahl der Punkte. Für einen Punkt ist die Aussage trivial. Für zwei Punkte gilt

$$\begin{aligned}
B_{P_{0,0},P_{0,1}}(s) &= \sum_{k=0}^{1} B_{k,n}(s)P_{0,k} \\
&= (1-s)P_{0,0} + s((1-t)P_{0,0} + tP_{1,0}) \\
&= (1-st)P_{0,0} + stP_{1,0} \\
&= B_{P_0,P_1}(st).
\end{aligned}$$

mit der Gleichung aus Aufgabe 7. Für $n+1$ Punkte haben wir mit der Induktionsvoraussetzung

$$\begin{aligned}
B_{P_{0,0},\ldots,P_{0,n}}(s) &= (1-s)B_{P_{0,0},\ldots,P_{0,n-1}}(s) + sB_{P_{0,1},\ldots,P_{0,n}}(s)\\
&= (1-s)B_{P_{0,0},\ldots,P_{n-1,0}}(st) + sB_{P_{0,1},\ldots,P_{n-1,1}}(st)\\
&= (1-s)B_{P_{0,0},\ldots,P_{n-1,0}}(st)\\
&\quad + sB_{(1-t)P_{0,0}+tP_{1,0},\ldots,(1-t)P_{n-1,0}+tP_{n,0}}(st)\\
&= (1-s)B_{P_{0,0},\ldots,P_{n-1,0}}(st)\\
&\quad + s\left((1-t)B_{P_{0,0},\ldots,P_{n-1,0}}(st) + tB_{P_{1,0},\ldots,P_{n,0}}(st)\right)\\
&= (1-st)B_{P_{0,0},\ldots,P_{n-1,0}}(st) + stB_{P_{1,0},\ldots,P_{n,0}}(st)\\
&= B_{P_{0,0},\ldots,P_{n,0}}(st)
\end{aligned}$$

Für den Beweis der zweiten Gleichung verwenden wir die erste Gleichung, wobei die Reihenfolge der Punkte umgedreht wurde. Man muss dann also $1-t$ statt t einsetzen.

$$\begin{aligned}
B_{P_{0,n},\ldots,P_{n,0}}(s) &= B_{P_{n,0},\ldots,P_{0,n}}(1-s)\\
&= B_{P_{n,0},\ldots,P_{0,0}}((1-t)(1-s))\\
&= B_{P_{0,0},\ldots,P_{n,0}}(1-(1-t)(1-s))\\
&= B_{P_{0,0},\ldots,P_{n,0}}((1-s)t+s).
\end{aligned}$$

Dies ist die Behauptung. **q.e.d.**

6.2 Splines

6.11. Definition: Seien

$$x_1 < \ldots < x_l$$

Knoten in $\mathbb{R}$, $n \in \mathbb{N}_0$. Eine Funktion $s : [x_1, x_l] \to \mathbb{R}$ heißt Spline mit einfachen Knoten vom Grad n, wenn gelten:

- $s \in \mathcal{P}_n$ auf den offenen Teilintervallen $]x_k, x_{k+1}[$ mit $k = 1, \ldots, l-1$.
- Für $n \geq 1$ ist s mindestens $n-1$-mal differenzierbar in den inneren Knoten $x_2, \ldots, x_{l-1}$.

Wir bezeichnen den Raum dieser Splines mit $S_n(x_1, \ldots, x_l)$.

6.12. Beispiel: Für $n = 1$ sind die Splines lineare Funktionen zwischen den Knoten, und stetig in den Knoten. Es handelt sich also um Sreckenzüge. Die Interpolationsaufgabe

$$s(x_1) = y_1, \ldots, s(x_l) = y_l$$

ist in diesem Fall offenbar eindeutig lösbar.

6.13. Beispiel: Für $n = 0$ fordern wir lediglich, dass die Funktion in den offenen Intervallen konstant ist. Es handelt sich also um Treppenfunktionen.

6.14 Aufgabe: Zeigen Sie, dass für $s \in S_n(x_1, \ldots, x_l)$ die Stammfunktion

$$\tilde{s}(x) = \int_{x_1}^{x} s(t)\,dt$$

ein Spline $\tilde{s} \in S_{n+1}(x_1, \ldots, x_l)$ ist. Sei $s \in S_1(0, 1, 2)$ der Spline mit

$$s(0) = 0, \quad s(1) = 1, \quad s(2) = 0.$$

Berechnen und zeichnen Sie den Spline $\tilde{s}$.

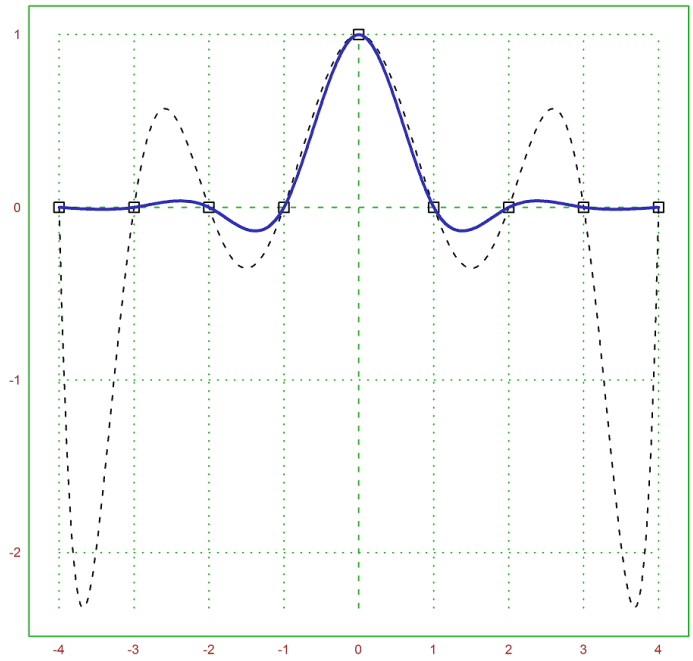

Abbildung 6.4: Vergleich Interpolationspolynom und Spline

Mit Splines lassen sich Funktionen wesentlich glatter interpolieren als mit Polynomen. EMT bringt Funktionen `spline` und `splineeval` mit, die eine Interpolation mit kubischen Splines ($n = 3$) ermöglichen.

```
>n=4; xp=-n:n; yp=(xp==0); // Punkte -4,...,4, Werte 0,0,0,0,1,0,0,0,0
>d=interpolate(xp,yp); // berechne dividierte Differenzen
>plot2d("interpval(xp,d,x)",-n,n,style="--"); // plotte Polynom
>plot2d(xp,yp,>points,>add); // plotte Punkte
>s=spline(xp,yp); // berechne Spline
>plot2d("splineval(xp,yp,s,x)", ...
>  >add,color=blue,thickness=2); // plotte Spline
```

Die Bedingung an die Differenzierbarkeit besagt, dass

$$s_-^{(\nu)}(x_k) = s_+^{(\nu)}(x_k) \qquad \text{für } 0 \le \nu \le n-1.$$

in allen Knoten x_k, $k = 2, \ldots, l-1$. Es gibt also in jedem Knoten $n-1$ Bedingungen zu erfüllen.

6.15 Aufgabe: Zeigen Sie, dass es genau einen Spline $s \in S_n(x_1, \ldots, x_l)$ gibt, der Interpolationsbedingungen

$$s(x_1) = y_1, s'(x_1) = d_1, \ldots, s^{(n-1)}(x_1) = d_{n-1}, s(x_2) = y_2, \ldots, s(x_l) = y_l$$

erfüllt. Berechnen und Zeichnen Sie einen Spline $s \in S_2(0,1,2,3)$ mit

$$s(0) = 0, s'(0) = 0, s(1) = 1, s(2) = 1, s(3) = 0.$$

Zeigen Sie für diesen Spline $s'(3) = 0$, sowie $s'(1) = 2$, $s'(2) = -2$. Nutzen Sie aus, dass der Spline spiegelsymmetrisch zu $x = 3/2$ ist.

6.16 Satz: *Der Splineraum $S_n(x_1, \ldots, x_l)$ hat die Dimension $n + l - 1$.*

Beweis: Die eindeutige Lösbarkeit der linearen Interpolationsaufgabe aus der obigen Aufgabe mit $n + l - 1$ Bedingungen beweist die Behauptung. **q.e.d.**

6.17 Aufgabe: Zeigen Sie, dass die Splines

$$1, x, x^2, \ldots, x^n, (x - x_2)_+^n, \quad , (x - x_{l-1})_+^n$$

eine Basis von $S_n(x_1, \ldots, x_l)$ bilden. Dabei ist

$$(x-a)_+^n = \begin{cases} (x-a)^n, & x \geq a, \\ 0, & x < a, \end{cases}$$

die abgebrochene Potenzfunktion.

6.18. Definition: Für den Splineraum ist es sinnvoll den Begriff der Spline-Nullstelle mit Vielfachheit k zu definieren. Dies ist entweder eine isolierte Nullstelle einer Vielfachheit k, also

$$s(x) = 0, \quad \ldots, \quad s^{(k-1)}(x) = 0,$$

wobei wir am Rand die einseitige Ableitung verwenden, oder ein maximal großes Intervall, auf dem der Spline identisch 0 ist. Das Intervall zählt dann als Nullstelle der Vielfachheit $n + m$, wobei n der Splinegrad ist und m die Anzahl der Knotenintervalle, die das Nullstellenintervall umfasst.

6.19 Satz: *Der Spline $s \in S_n(x_1, \ldots, x_l)$, $n \geq 1$, kann höchstens $n + l - 2$ Spline-Nullstellen in $[x_1, x_l]$ haben, einschließlich Vielfachheit gezählt, oder s ist identisch 0 auf $[x_1, x_l]$.*

Beweis: Wir beweisen den Satz per Induktion nach n. Sei zunächst $n = 1$, der Spline also ein Streckenzug.

Für diesen Fall beweisen wir den Satz durch Induktion nach l. Für $l = 1$ ist der Satz trivial. Falls $s \in S_1(x_1, \ldots, x_{l+1})$ nun

$$n + (l+1) - 1 = l + 1$$

Spline-Nullstellen hat, so gibt es zwei Fälle: Falls der Spline auf $[x_l, x_{l+1}]$ identisch 0 ist, so hat der Spline in $[x_1, x_l]$ immer noch l Nullstellen, ist dort also nach Induktionsvoraussetzung identisch 0. Andernfalls kann er höchstens eine Nullstelle in $[x_l, x_{l+1}]$ haben, und wir erhalten dasselbe Ergebnis.

Der Satz gelte also für $n \geq 1$. Falls der Spline $s \in S_{n+1}(x_1, \ldots, x_l)$ nun

$$(n+1) + l - 1 = n + l$$

Spline-Nullstellen hat, so folgt aus dem Satz von Rolle und der Definition der Spline-Nullstellen, dass $s' \in S_n(x_1, \ldots, x_l)$ mindestens $n + l - 1$ Nullstellen hat. Nach Induktionsvoraussetzung ist $s' = 0$, also s konstant. Es folgt, dass s identisch 0 ist. **q.e.d.**

Ein Spline lässt sich auf ganz $\mathbb{R}$ fortsetzen, möglicherweise sogar identisch 0, wobei man die Spline-Bedingung in x_1 und x_l voraussetzt.. Die Fortsetzung ist bis auf Vielfache von $(x-x_1)^n$ bzw. $(x-x_l)^n$ eindeutig. Der fortgesetzte Spline kann als Spline mit 2 Knoten mehr aufgefasst werden, und hat dann höchstens $n+l$ Spline-Nullstellen, oder ist identisch 0.

6.20. Definition: Ein kubischer Spline $s \in S_3(x_1,\dots,x_l)$ mit

$$s''(x_1) = s''(x_l) = 0$$

wird als natürlicher Spline bezeichnet.

6.21 Satz: *Zu Punkten $x_1 < \dots < x_l$ und Daten $y_1,\dots,y_l$ existiert ein eindeutiger natürlicher Spline $s \in S_3(x_1,\dots,x_l)$, der die Interpolationsbedingungen*

$$s(x_1) = y_1, \quad \dots, \quad s(x_l) = x_l$$

erfüllt.

Beweis: Mit den zusätzlichen zwei Bedingungen für natürliche Splines haben wir $l+2$ lineare Bedingungen, was der Dimension des Splineraums entspricht. Es genügt, also zu zeigen, dass das homogene System nur die Nullfunktion als Lösung hat. Falls x_1 oder x_l in einem Nullstellenintervall liegen, so reduziert sich das Problem auf ein Problem mit weniger Knoten. Wir können das also ausschließen. Sei

$$s''(x_1) = 0, \quad s(x_1) = 0, \quad \dots, \quad s(x_l) = 0, \quad s''(s_l) = 0.$$

Dann hat s'' in $]x_1, x_l[$ noch $l-2$ Spline-Nullstellen, sowie zwei weitere Nullstellen in x_1 und x_l. Es folgt aus dem obigen Satz $s'' = 0$. **q.e.d.**

Ein natürlicher Spline lässt sich mit linearen Funktionen auf ganz $\mathbb{R}$ fortsetzen. Wenn die zweite Ableitung die Biegung des Splines näherungsweise wiedergibt, so entspricht dieser Spline einen Spline, der um die inneren Knoten gebogen wurde.

Er lässt sich in EMT mit der Funktion `spline` berechnen. Diese Funktion gibt den Vektor

$$s''(x_2),\dots,s''(x_{l-1})$$

zurück. Die Funktion `splineval` berechnet aus den Daten dieses Vektors und den Knoten den Spline in jedem Teilintervall, und setzt den Spline außerhalb von $[x_1, x_l]$ linear fort.

```
>xn=-2:2; yn=[-1,1,1,-1,1];
>plot2d(xn,yn,>points,r=3);
>s=spline(xn,yn); plot2d("splineval(xn,yn,s,x)",>add);
```

Zur Berechnung setzen wir

$$M_k = s''(x_k)$$

für $k = 1,\dots,l$ und

$$h_k = x_{k+1} - x_k$$
$$\delta_i = \frac{y_{k+1} - y_k}{h_k}$$

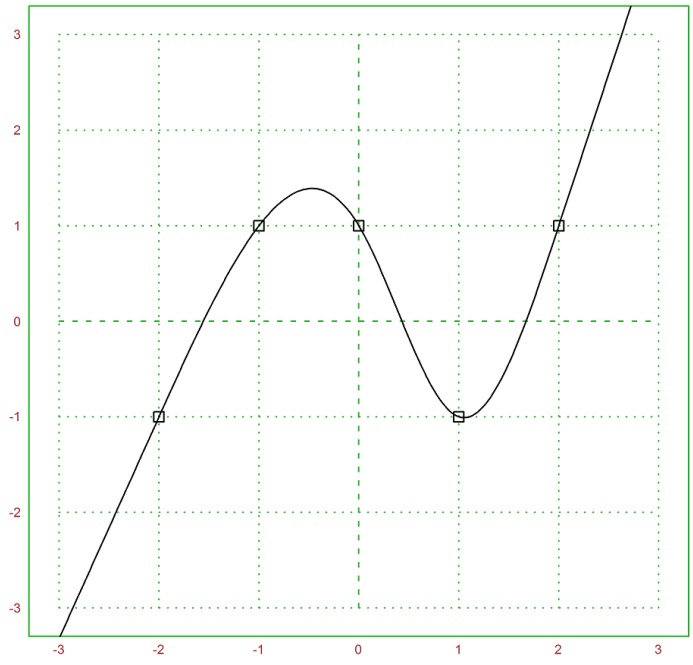

Abbildung 6.5: Natürlicher Spline

für $k = 1, \ldots, l-1$. Gemäß der folgenden Übung gilt dann

$$\frac{h_k}{6} M_k + \frac{h_k + h_{k+1}}{3} M_{k+1} + \frac{h_{k+1}}{6} M_{k+2} = \delta_{k+1} - \delta_k$$

für $k = 1, \ldots, l-2$. Für äquidistante Punkte

$$h_k = h \qquad \text{für alle } k = 1, \ldots, l-1$$

ergibt sich zum Beispiel das Gleichungssystem

$$\frac{h}{6} \begin{pmatrix} 4 & 1 & & & & 0 \\ 1 & 4 & 1 & & & \\ & 1 & 4 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & 4 & 1 \\ 0 & & & & 1 & 4 \end{pmatrix} \cdot \begin{pmatrix} M_2 \\ \vdots \\ M_{l-1} \end{pmatrix} = \begin{pmatrix} \delta_2 - \delta_1 \\ \vdots \\ \delta_{l-1} - \delta_{l-2} \end{pmatrix}$$

6.22 Aufgabe: Ziegen Sie für ein Polynom $p \in \mathcal{P}_3$

$$p'(x) = \frac{p(x+h) - p(x)}{h} - h \left(\frac{p''(x+h)}{6} + \frac{p''(x)}{3} \right),$$
$$p'(x+h) = \frac{p(x+h) - p(x)}{h} + h \left(\frac{p''(x)}{6} + \frac{p''(x+h)}{3} \right).$$

Verwenden Sie das Taylorpolynom vom Grad 2 mit Restglied der Funktion p um x, und

$$\frac{q(x+h)}{3} + \frac{2q(x)}{3} = q(x) + \frac{q'(\xi)\,h}{3}$$

für die lineare Funktion $q = p''$ und alle $\xi \in \mathbb{R}$.

6.23 Satz: *Der natürliche Spline $s \in S_3(x_1, \ldots, x_l)$ mit den Interpolationseigenschaften*

$$s(x_1) = y_1, \quad \ldots, \quad s(x_l) = x_l$$

minimiert die Semi-Norm

$$\|s''\| = \sqrt{\int_{x_1}^{x_l} s''(t)^2\,dt}$$

unter allen Splines $\tilde{s} \in S_3(x_1, \ldots, x_l)$ mit diesen Interpolationseigenschaften.

Dieser Satz von Holladay besagt, dass die mittlere Krümmung durch den natürlichen Interpolationsspline minimiert wird.

Beweis: Die Behauptung folgt, wenn wir

$$\|s'' + \tilde{s}''\|^2 = \|s''\|^2 + \|\tilde{s}''\|^2.$$

für alle $\tilde{s}$ zeigen, die das homogene Problem

$$s(x_1) = 0, \quad \ldots, \quad s(x_l) = 0$$

lösen. Dazu genügt es

$$\int_{x_1}^{x_l} s''(t)\tilde{s}''(t)\,dt = 0$$

nachzuweisen. Mit partieller Integration erhält man

$$\begin{aligned}
\int_{x_1}^{s_l} s''(t)\,\tilde{s}''(t)\,dt &= [s''(t)\,\tilde{s}'(t)]_{x_1}^{x_l} - \int_{x_1}^{s_l} s'''(t)\,\tilde{s}'(t)\,dt \\
&= -\int_{x_1}^{s_l} s'''(t)\,\tilde{s}'(t)\,dt \\
&= -[s'''(t)\,\tilde{s}(t)]_{x_1}^{x_l} + \int_{x_1}^{x_l} s''''(t)\,\tilde{s}(t)\,dt \\
&= 0.
\end{aligned}$$

q.e.d.

6.3 B-Splines

6.24 Satz: *Für Knoten*

$$x_1 < \ldots < x_{n+2}$$

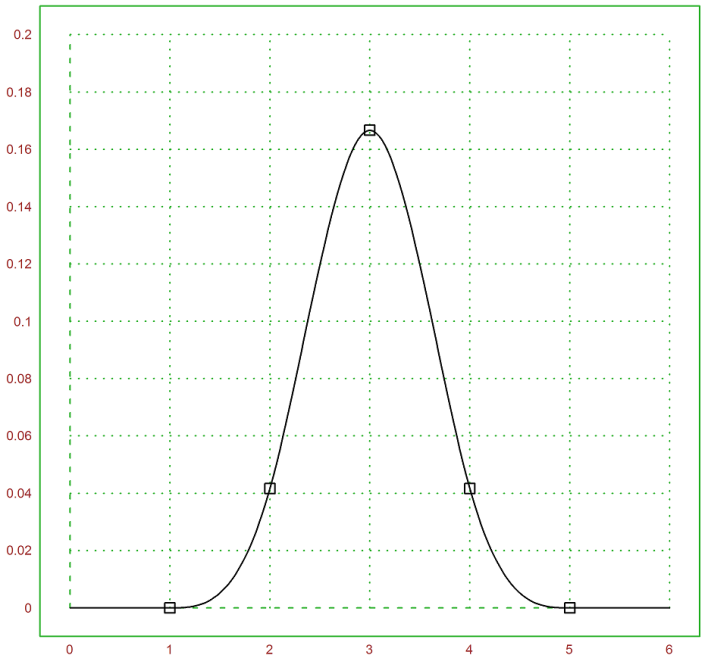

Abbildung 6.6: Kubischer B-Spline $B_3(1,2,3,4,5)$

gibt es genau einen Spline $s \in S_n(x_1,\ldots,x_{n+2})$ *mit*

$$s(x_1) = \ldots = s^{(n-1)}(x_1) = 0,$$
$$s(x_{n+2}) = \ldots = s^{(n-1)}(x_{n+2}) = 0,$$

und

$$\int_{x_1}^{x_{n+2}} s(t)\,dt = \frac{1}{n+1}.$$

Dieser Spline ist im Intervall $]x_1, x_{n+2}[$ *positiv.*

6.25. Definition: Wir bezeichnen den Spline als B-Spline $B(x_1,\ldots,x_{n+2})$.

Beweis: Wir suchen einen Spline $s \in S_n(x_0,\ldots,x_{n+3})$ mit den $2n$ Interpolationsbedingungen

$$s(x_1) = \ldots = s^{(n-1)}(x_1) = 0,$$
$$s(x_{n+2}) = \ldots = s^{(n-1)}(x_{n+2}) = 0,$$

und $s(\xi) = 1$ für ein fest gewähltes $\xi \in]x_1, x_{2n+2}[$. Das sind $2n+1$ lineare Bedingungen, was der Dimension des Spline-Raumes entspricht. Wir betrachten daher das homogene Problem $s(\xi) = 0$. Jede Lösung davon hat eine Spline-Nullstelle der Vielfachheit n an jedem Rand, und eine weitere Nullstelle im Innern, insgesamt als $2n+1$ Spline-Nullstellen. Es folgt $s = 0$ aus Satz 19.

s muss in $]x_1, x_{n+2}[$ positiv sein, da ansonsten s wieder eine Nullstelle zu viel hätte. Wir können s daher so normieren, dass es die zusätzliche Integralbedingung erfüllt. **q.e.d.**

Der B-Spline lässt sich offenbar als Spline außerhalb des Intervalls $[x_1, x_{n+2}]$ identisch 0 fortsetzen. Das bedeutet, dass er eine Darstellung der Form

$$B(x_1, \ldots, x_{n+2})(x) = \sum_{k=1}^{n+2} \lambda_k (x - x_k)_+^n$$

mit

$$\sum_{k=1}^{n+2} \lambda_k (x - x_k)^n = 0$$

haben muss. Wenn wir uns den Spline identisch 0 fortgesetzt denken, so gilt für $x > x_{n+2}$

$$\begin{aligned} \frac{1}{n+1} &= \int_{x_1}^{x} B(x_1, \ldots, x_{n+2})(t)\, dt \\ &= \sum_{k=1}^{n+2} \lambda_k \int_{x_1}^{x} (t - x_k)_+^n \, dt \\ &= \frac{1}{n+1} \sum_{k=1}^{n+2} \lambda_k (x - x_k)^{n+1}. \end{aligned}$$

Es folgt für alle $x \in \mathbb{R}$

$$\sum_{k=1}^{n+2} \lambda_k (x - x_k)^{n+1} = 1.$$

Aufgrund des nächsten Satzes definiert diese Bedingung die λ_k eindeutig. Wir definieren

$$\lambda_k(x_1, \ldots, x_{n+2}) = \lambda_k$$

für $k = 1, \ldots, n+2$, und

$$\lambda_k(x_1, \ldots, x_{n+2}) = 0$$

für $k < 1$ oder $k > n+2$.

6.26 Satz: *Die Funktionen*

$$(x - x_0)^n, \ldots, (x - x_n)^n$$

sind für paarweise verschiedene Punkte $x_0, \ldots, x_n \in \mathbb{C}$ linear unabhängig.

Beweis: Angenommen

$$\sum_k \lambda_k (x - x_k)^n = 0$$

für alle $x \in \mathbb{C}$ mit $\lambda_k \neq 0$ für ein k. Durch Ableiten erhalten wir

$$\sum_k \lambda_k (x - x_k)^l = 0$$

für $l = 0, \ldots, n$. Wir betrachten die Matrix

$$M(x) = \begin{pmatrix} 1 & \ldots & 1 \\ x - x_0 & \ldots & x - x_n \\ \vdots & & \vdots \\ (x - x_0)^n & \ldots & (x - x_n)^n \end{pmatrix}$$

für die dann $\det M(x) = 0$ gelten muss, und zwar für alle $x \in \mathbb{C}$. Aufgrund von Aufgabe 3.17 ist das nicht möglich. **q.e.d.**

6.27 Satz: *Die Koeffizienten lassen sich rekursiv berechnen. Es gilt*

$$\lambda_k(x_1, \ldots, x_{n+2}) = \frac{\lambda_k(x_1, \ldots, x_{n+1}) - \lambda_{k-1}(x_2, \ldots, x_{n+2})}{x_{n+2} - x_1}$$

für alle $k \in \mathbb{Z}$. Der Induktionsanfang ist durch

$$\lambda_1(x_1, x_2) = \lambda_2(x_1, x_2) = \frac{1}{x_2 - x_1}$$

oder durch

$$\lambda_0(x_1) = 1$$

gegeben.

Beweis: Es gilt für alle $x \in \mathbb{R}$

$$\begin{aligned}
1 &= \sum_{k=1}^{n+2} \lambda_k(x_1, \ldots, x_{n+2})(x - x_k)^{n+1} \\
&= \sum_{k=1}^{n+2} (x_{n+2} - x_k)\lambda_k(x_1, \ldots, x_{n+2})(x - x_k)^n \\
&\qquad + (x - x_{n+2}) \sum_{k=1}^{n+2} \lambda_k(x_1, \ldots, x_{n+2})(x - x_k)^n \\
&= \sum_{k=1}^{n+1} (x_{n+2} - x_k)\lambda_k(x_1, \ldots, x_{n+2})(x - x_k)^n.
\end{aligned}$$

Dabei wurde

$$\sum_{k=1}^{n+2} \lambda_k(x_1, \ldots, x_{n+2})(x - x_k)^n = 0$$

verwendet. Wir erhalten wegen der linearen Unabhängigkeit der Funktionen für alle $k \in \mathbb{Z}$

$$(x_{n+2} - x_k)\,\lambda_k(x_1, \ldots, x_{n+2}) = \lambda_k(x_1, \ldots, x_{n+1}).$$

In analoger Weise zeigt man

$$(x_1 - x_k)\,\lambda_k(x_1, \ldots, x_{n+2}) = \lambda_{k-1}(x_2, \ldots, x_{n+2}).$$

Es folgt die Behauptung. **q.e.d.**

Für eine folgenden Beweis benötigen wir noch die Gleichung

$$(x_k - x_1)\lambda_k(x_1, \ldots, x_{n+1}) + (x_{n+2} - x_k)\lambda_{k-1}(x_2, \ldots, x_{n+2}) = 0,$$

die aus dem letzten Teil des Beweises folgt.

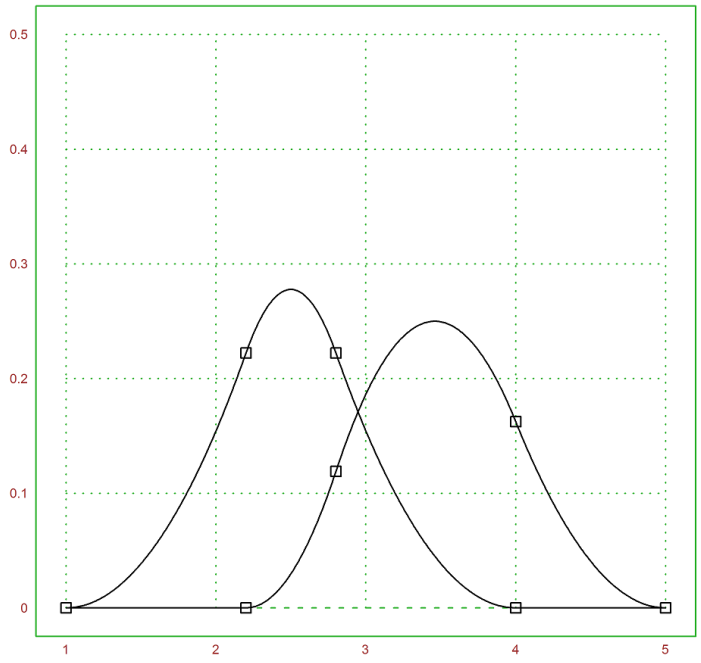

Abbildung 6.7: $B_2(1, 2.2, 2.8, 4)$ und $B_2(2.2, 2.8, 4, 5)$

6.28. Beispiel: Gegeben seien Punkte

$$x_1, \ldots, x_m.$$

Dann können wir rekursiv die Koeffizienten

$$\lambda_k(x_l, x_{l+n+1})$$

für $l = 1, \ldots, m-(n+1)$ berechnen. Wir starten mit der Einheitsmatrix und führen n Schritte lang Zeilenoperationen durch. Die Matrixsprache von Eule sorgt dafür, dass die Differenzen der Zeilen durch die richtigen Differenzen der Knoten dividiert werden.

```
>function lambda (xnode,n) ...
$  m = cols(xnode);
$  M = id(m);
$  for i=1 to n+1
$    r = rows(M);
$    M = (M[1:r-1]-M[2:r])/(xnode[i+1:m]-xnode[1:m-i])';
$  end;
$  return M
$endfunction
```

Das Ergebnis ist eine Matrix M, die in den Zeilen die λ_k der einzelnen Splines enthält. Die Koeffizienten von $B_n(x_l, x_{l+n+1})$ beginnen beim l-ten Element der Zeile. In unserem Fall sind des 2 B-Splines vom Grad 2.

```
>xnode := [1,2.2,2.8,4,5]
 [ 1  2.2  2.8  4  5 ]
>la := lambda(xnode,2)
      0.154321     -0.771605      0.771605     -0.154321             0
             0      0.330688     -0.631313      0.462963     -0.162338
```

Zur Berechnen des Splines auf einem Zeilenvektor $t_1, \ldots, t_p$ können wir wegen der Matrixsprache von EMT den folgenden Code verwenden.

```
>function lambdaeval (t,xnode,la,n) := la.(max(t-xnode',0)^n)
```

Dies stellt zunächst eine Matrix mit den Einträgen

$$\left((t_j - x_i)_+^n\right)_{i,j}$$

her, und addiert die Zeilen mit den Koeffizienten in den Zeilen von λ.

Zum Zeichnen der beiden B-Splines vom Grad 2 verwenden wir schließlich folgenden Code.

```
>t := linspace(1,5,500); s := lambdaeval (t,xnode,la,2);
>plot2d(t,s,a=1,b=5,c=0,d=0.5);
>plot2d(xnode,lambdaeval(xnode,xnode,la,2),>points,>add);
```

Nebenbei können wir mit Hilfe einer einfachen Riemann-Summe die Integrationsbedingung überprüfen.

```
>sum(s)*(4/500)
      0.333333
      0.333333
```

6.29 Satz: *Es gilt die Rekursion*

$$B_{n+1}(x_1, \ldots, x_{n+3}) = \frac{(x - x_1)B_n(x_1, \ldots, x_{n+2})(x) + (x_{n+3} - x)B_n(x_2, \ldots, x_{n+3})(x)}{x_{n+3} - x_1}.$$

Als Induktionsanfang kann

$$B_0(x_1, x_2)(x) = \begin{cases} \dfrac{1}{x_2 - x_1}, & x_1 \le x \le x_2, \\ 0, & \text{sonst}, \end{cases}$$

verwendet werden.

Man benennt diese Rekursionsformel nach de Boor. Es ist keineswegs selbstverständlich, dass die Funktionen auf der rechten Seite tatsächlich n mal in den Knoten differenzierbar sind.

Beweis: Zur Abkürzung setzen wir

$$\lambda_{k,1,n+l} = \lambda_{k-1}(x_1, \ldots, x_{n+l})$$

Dann erhalten wir der obigen Rekursionsformel und der darauf folgenden Bemerkung

$$\begin{aligned}((x-x_1)\,\lambda_{k,1,n+2} + (x_{n+3}-x)\,\lambda_{k-1,2,n+3})\,(x-x_k)_+^n \\ = ((x_k-x_1)\,\lambda_{k,1,n+2} + (x_{n+3}-x_k)\,\lambda_{k-1,2,n+3})\,(x-x_k)_+^n \\ + ((x-x_k)\,\lambda_{k,1,n+2} + (x_k-x)\,\lambda_{k-1,2,n+3})\,(x-x_k)_+^n \\ = (\lambda_{k,1,n+2} - \lambda_{k-1,2,n+3})\,(x-x_k)_+^{n+1} \\ = (x_{n+3}-x_1)\,\lambda_{k,1,n+3}\,(x-x_k)_+^{n+1}.\end{aligned}$$

Durch Summation über alle k folgt die Behauptung. **q.e.d.**

6.30 Satz: *Führt man zusätzliche Knoten*

$$x_{-n+1} < \ldots < x_1 < \ldots < x_l < \ldots < x_{l+n}$$

ein, so bilden die B-Spline

$$B(x_{-n+1},\ldots,x_2),\ldots,B(x_{l-1},\ldots,l+n+1)$$

eine Basis von $S_n(x_1,\ldots,x_l)$.

Man beachte, dass gerade die B-Splines ausgewählt sind, deren Träger ein Intervall mit $[x_1, x_l]$ gemeinsam haben.

Beweis: Die Anzahl der B-Splines ist

$$(l-1)-(-n+1)+1 = l+n-1$$

und stimmt der Dimension von $S_n(x_1,\ldots,x_l)$ überein. Es muss daher nur noch die lineare Unabhängigkeit gezeigt werden.

Sei eine Linearkombination s diese B-Splines auf $[x_1, x_l]$ identisch 0. s lässt sich dann auch als Spline auf $]-\infty, x_{-n+1}[$ und $]x_{l+n}, \infty[$ identisch 0 fortsetzen. Eine Nullstellenzählung mit Spline-Nullstellen in den Räumen $S_n(x_{-n+1},\ldots,x_2)$ und $S_n(x_{l-1},\ldots,x_{n+l})$ ergibt, dass s überall identisch 0 sein muss.

Es dann aber einfach zu zeigen, dass dann die Koeffizienten der Darstellung von s alle gleich 0 sein müssen. **q.e.d.**

Mit Hilfe der B-Splines versteht man den folgenden Satz von Schönberg-Whitney besser. Wir verwenden zum Beweis allerdings keine B-Splines, sondern eine einfache Nullstellenzählung.

6.31 Satz: *Seien die zusätzlichen Knoten aus der Basisdarstellung mit $B-Splines$*

$$x_{-n+1} < \ldots < x_1 < \ldots < x_l < \ldots < x_{l+n}$$

gegeben, sowie Punkte

$$x_{-n+1} < t_1 < \ldots < t_{n+l-1} < x_{l+n}$$

Dann ist das Interpolationsproblem

$$s(t_k) = s_k \qquad \text{für } k = 1,\ldots,n+l-1$$

durch Linearkombination der B-Spline-Basis genau dann für alle Werte s_k eindeutig lösbar, wenn

$$x_{-n+k} < t_k < x_{k+1}$$

für alle $k = 1, \ldots, n + l - 1$ gilt.

Die Interpolationsbedingung besagt also, dass man die Interpolationspunkte im Innern der Träger der B-Spline-Basis wählen muss.

Beweis: Angenommen die Bedingung ist erfüllt. Die Anzahl der linearen Interpolationsbedingungen entspricht der Dimension des Raumes. Wir müssen daher lediglich das homogene Problem untersuchen. Sei also

$$s(t_k) = 0 \qquad \text{für } k = 1, \ldots, n + l - 1$$

für eine Linearkombination der B-Spline-Basis. s hat dann schon zwei Nullstellen der Vielfachheit n an den Rändern von $[x_{-n+1}, x_{n+l}]$. Also hat s im Innern dieses Intervalls höchstens

$$n + ((l + n) - (-n + 1) + 1) - 2 - 2n = l + n - 2$$

Spline-Nullstellen, oder s ist identisch 0.

Wir müssen noch zeigen, dass die Nullstellen $t_1, \ldots, t_{n+l-1}$ tatsächlich für $n + l - 1$ Spline-Nullstellen gezählt werden dürfen. Wenn sie isoliert sind, so ist das der Fall. In jedem Nullstellen-Intervall von s, das m Intervalle umfasst, können aber aufgrund der Interpolationsbedingung höchstens $n + m$ der t_k liegen. Da wir aber ein solches Nullstellenintervall als $n + m$-fache Nullstelle zählen, hat s tatsächlich $n + l - 1$ zusätzliche Nullstellen, und ist damit identisch 0.

Angenommen die Interpolationsbedingung ist nicht erfüllt, also etwa

$$t_K < x_{-n+k}.$$

Dann müssen wir das Interpolationsproblem

$$s(t_k) = s_k \qquad \text{für } k = 1, \ldots, K$$

mit den Linearkombination der ersten $K - 1$ B-Splines lösen. Aus Dimensiongründen ist dies nicht immer möglich. **q.e.d.**

6.32. Definition: Wir definieren die normalisierten B-Splines als

$$N(x_1, \ldots, x_{n+2}) = (x_{n+2} - x_1)\, B(x_1, \ldots, x_{n+2})$$

für Knoten $x_1 < \ldots < x_{n+2}$.

6.33 Aufgabe: Zeigen Sie mit Hilfe der Rekursionformel für die B-Splines die Formel

$$N(x_1, \ldots, x_{n+3})(x) = \frac{x - x_1}{x_{n+3} - x_1}\, N(x_1, \ldots, x_{n+2})(x) + \frac{x_{n+3} - x}{x_{n+3} - x_1}\, N(x_2, \ldots, x_{n+3})(x)$$

für die normalisierten B-Splines. Auf der rechten Seite steht eine Konvex-Kombination der normalisierten B-Splines von einem Grad niedriger. Folgern Sie per Induktion

$$\sum_{k=1}^{n+l-1} N(x_{-n+k}, \ldots, x_{k+1})(x) = 1$$

für $x \in [x_1, x_l]$ mit Knoten

$$x_{-n+1} < \ldots < x_1 < \ldots < x_l < \ldots < x_{l+n}$$

wie in der Basisdarstellung der B-Splines.

Die normalisierten B-Splines bilden also, genau wie die Bernstein-Polynome eine Zerlegung der Eins.

6.4 Mehrfache Knoten

6.34. Definition: Im Falle von mehrfachen Knoten in dem Knotentupel

$$x_1 \leq \ldots \leq x_l$$

definieren wir einen Spline $s \in S_n(x_1, \ldots, x_l)$ dadurch, dass s in den Intervallen zwischen den Knoten in $\mathcal{P}_n$ ist, und in einem m-fachen Knoten $n - m$-mal differenzierbar für $n > m$ bzw. stetig für $n = m$. Falls wir s auf ganz $\mathbb{R}$ fortsetzen, so muss dies auch in x_1 und x_l gelten. Wir lassen maximal die Vielfachheit $m = n + 1$ zu. Die B-Splines

$$B(x_1, \ldots, x_{n+2})$$

mit mehrfachen Knoten definieren wir als Splines mit der Integralbedingung

$$\int_{x_1}^{x_l} B(x_1, \ldots, x_{n+2})(t)\, dt = \frac{1}{n+1}$$

die auf ganz $\mathbb{R}$ mit 0 als Splines fortgesetzt werden können. Dabei muss $x_1 \neq x_{n+2}$ sein. Analog setzen wir wieder

$$N(x_1, \ldots, x_{n+2}) = (x_{n+2} - x_1)\, B(x_1, \ldots, x_{n+2}).$$

Die Sätze für einfache Knoten gelten mit entsprechenden Modifikationen für mehrfache Knoten. Die Dimension des Splineraumes ist weiterhin

$$\dim S_n(x_1, \ldots, x_l) = n + l - 1,$$

wobei wir für die Basis in einem m-fachen Knoten die m Funktionen

$$(x - x_k)_+^n, \quad \ldots, \quad (x - x_k)_+^{n-m+1}$$

als Basisfunktionen nehmen. Im Fall $m = n + 1$ enthält diese Basis die unstetige Funktion $(x - x_k)_+$.

6.35 Aufgabe: Zeigen Sie, dass der Satz über die maximale Anzahl von Spline-Nullstellen richtig bleibt. Beachten Sie, dass beim Ableiten möglicheweise die Vielfachheit von Knoten reduziert werden muss.

6.36. Beispiel: Die Bernstein-Polynome $B_{k,n}$ können als Spline mir mehrfachen Knoten auf ganz $\mathbb{R}$ fortgesetzt werden. Setzt man das Bernstein-Polynom

$$B_{k,n}(x) = \binom{n}{k} t^k (1-t)^{n-k}$$

mit 0 fort, so ist es in 0 genau $k-1$ mal differenzierbar, in 1 genau $n-k-1$ mal differenzierbar. Folglich haben wir

$$B_{k,n} \in S_n(\underbrace{0,\ldots,0}_{n-k+1},\underbrace{1,\ldots,1}_{k+1})$$

also

$$B_{0,n} \in S_n(0,\ldots,0,1), \quad \ldots, \quad B_{n,n} \in S_n(0,1,\ldots,1).$$

Es gilt

$$B_{k,n} \in N(\underbrace{0,\ldots,0}_{n-k+1},\underbrace{1,\ldots,1}_{k+1}).$$

6.5 Rationale Kurven

6.37. Definition: Rationale Kurven sind Quotienten von Splines oder Bezierkurven mit eindimensionalen Splines oder Bezierkurven. Quotienten von Splines bezeichnet man auch als Nurbs.

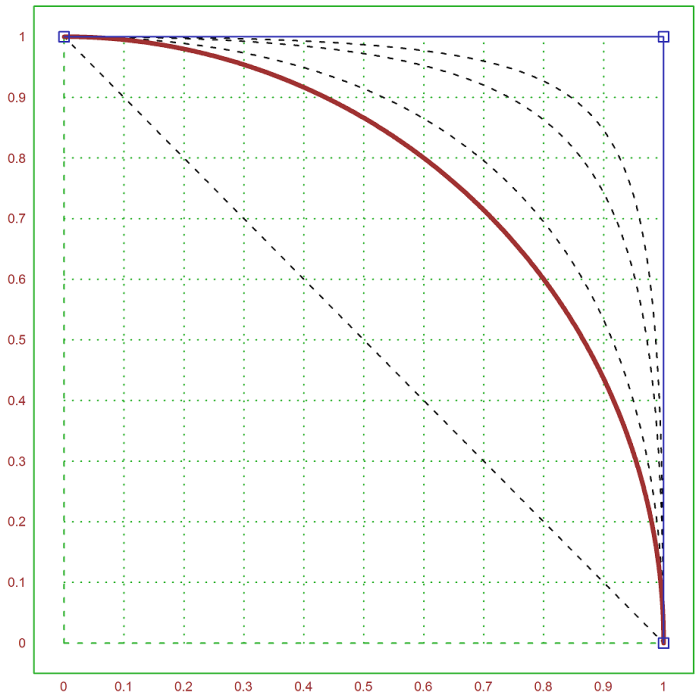

Abbildung 6.8: Rationale Bezierkurven

6.38. Beispiel: Für Bezierkurven erhält man

$$\gamma(t) = \frac{1}{\sum_{k=0}^{n} \alpha_k B_{k,n}(z)} \sum_{k=0}^{n} B_{k,n}(z) P_k$$

Dies kann als Projektion der Bezierkurve mit den Kontrollpunkten

$$\tilde{P}_k = \begin{pmatrix} P_k \\ \alpha_k \end{pmatrix} \in \mathbb{R}^{m+1}$$

in den $\mathbb{R}^m$ mit dem Projektionsoperator

$$\mathcal{P}(x) = \begin{pmatrix} x_1/x_{m+1} \\ \vdots \\ x_m/x_{m+1} \end{pmatrix}$$

angesehen werden. Das entspricht der zentrischen Projektion von 0 auf die Ebene $x_{m+1} = 1$. Es ist daher kein Wunder, dass sich Kreise und Ellipsen als rationale Kurven darstellen lassen. Sie sind schließlich Projektionen von Kegelschnitten.

Die rationale Kurve beginnt in (P_0) und endet in (P_n). Es ist deswegen günstig,

$$\alpha_0 = \alpha_n = 1$$

zu fordern. Wählt man darüber hinaus

$$\tilde{P}_1 = \begin{pmatrix} \alpha_1 x_1 \\ \vdots \\ \alpha_1 x_m \\ \alpha_1 \end{pmatrix},$$

so dass also

$$\mathcal{P}(P_1) = P_1$$

gilt, so ist die rationale Bezierkurve als Projektion einer tangentialen Kurve wieder tangential zu $P_1 - P_0$. Im folgenden Beispiel erzeugen wir daher quadratische rationale Kurven der Form

$$\gamma(t) = \frac{1}{(1-t)^2 + 2rt(1-t) + t^2} \left((1-t)^2 P_0 + 2rt(1-t)P_1 + t^2 P_2 \right).$$

Dies ergibt mit einer speziellen Wahl von Kontrollpunkten die Kurven in Abbildung 6.8.

```
>P0 := [1,0]'; P1 := [1,1]'; P2 := [0,1]';
>function gd(t,r) := (1-t)^2+2*r*t*(1-t)+t^2
>function g1(t,r) := ((1-t)^2*P0[1]+2*r*t*(1-t)*P1[1]+t^2*P2[1])/gd(t,r)
>function g2(t,r) := ((1-t)^2*P0[2]+2*r*t*(1-t)*P1[2]+t^2*P2[2])/gd(t,r)
>x=0:0.001:1;
>r=[0,1,2,3]';
>plot2d(g1(x,r),g2(x,r),a=0,b=1,c=0,d=1,style="--");
>plot2d(g1(x,1/sqrt(2)),g2(x,1/sqrt(2)),>add,color=red,thickness=3);
>M=P0|P1|P2;
>plot2d(M[1],M[2],>points,>add,color=blue);
>plot2d(M[1],M[2],>add,color=blue);
```

Wir rechnen schließlich noch mit Maxima nach, dass der Fall $r = 1/\sqrt{2}$ tatsächlich einen Kreisbogen ergibt.

```
>function gd(t,r) &= (1-t)^2+2*r*t*(1-t)+t^2;
>function g1(t,r) &= ((1-t)^2+2*r*t*(1-t))/gd(t,r);
>function g2(t,r) &= (2*r*t*(1-t)+t^2)/gd(t,r);
>&g1(t,1/sqrt(2))^2+g2(t,1/sqrt(2))^2|ratsimp
                                  1
```

Rationale Bezierkurven oder B-Splines können also verwendet werden, um Kurven enger an die Kontrollpolygone anzuschmiegen. Letztendlich approximieren die Kurven das Kontrollpolygon.

Kapitel 7

Lineare Gleichungssysteme

7.1 Das Gauß-Verfahren

Zu lösen ist das Lineare Gleichungssystem (LGS)

$$Ax = b$$

mit regulärer Matrix $A \in \mathbb{K}^{n\times n}$ und $b \in \mathbb{K}^n$ ($\mathbb{K} = \mathbb{R}$ oder $\mathbb{K} = \mathbb{C}$). Das Gauß-Verfahren führt Äquivalenzumformungen im Schema

$$\left.\begin{array}{ccc|c} a_{1,1} & \dots & a_{1,n} & b_1 \\ \vdots & & \vdots & \vdots \\ a_{n,1} & \dots & a_{n,n} & b_n \end{array}\right.$$

durch. Zulässig sind die folgenden Operationen.

- Vertauschungen von Zeilen,
- Addition des λ-fachen einer Zeile zu einer anderen,
- Multiplikation einer Zeile mit $\lambda \neq 0$.

7.1 Aufgabe: Zeigen sie, dass diese Änderungen die Lösungsmenge des Gleichungssystems nicht ändern und mit zulässigen Operationen wieder rückgängig gemacht werden können.

Ziel ist, ein Schema der Form

$$\left.\begin{array}{ccc|c} \tilde{a}_{1,1} & \dots & * & \tilde{b}_1 \\ & \ddots & \vdots & \vdots \\ 0 & & \tilde{a}_{n,n} & \tilde{b}_n \end{array}\right.$$

zu erreichen. Das System

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

ist dann äquivalent zu $Ax = b$. Es hat also dieselbe Lösungsmenge, lässt sich aber leicht nach x auflösen. Man berechnet rekursiv

$$x_n = \tilde{b}_n/\tilde{a}_{n,n}, \quad x_{n-1} = (\tilde{b}_{n-1} - \tilde{a}_{n-1,n}x_n)/\tilde{a}_{n-1,n-1}, \quad \dots$$

Wenn A regulär ist, ist es auch $\tilde{A}$ und deswegen

$$\tilde{a}_{1,1} \neq 0, \quad \dots, \quad \tilde{a}_{n,n} \neq 0.$$

Alternativ kann mit weiteren Umformungen das Schema in die Form

$$\left(\begin{array}{ccc|c} 1 & & 0 & \tilde{\tilde{b}}_1 \\ & \ddots & & \vdots \\ 0 & & 1 & \tilde{\tilde{b}}_n \end{array}\right)$$

gebracht werden. Damit lässt sich

$$x = I_n x = \tilde{\tilde{b}}.$$

sofort ablesen.

Im i-ten Schritt sieht unser Schema so aus.

$$\left(\begin{array}{ccccc|c} a_{1,1} & & & & * & b_1 \\ & \ddots & & & & \\ & & a_{i,i} & * & & \vdots \\ & & \vdots & & & \\ 0 & & a_{n,i} & * & & b_n \end{array}\right)$$

Falls $a_{i,i} \neq 0$ ist, so wird das $(-a_{k,i}/a_{i,i})$-fache der i-ten Zeile zur k-ten addiert, also

$$\tilde{a}_{k,j} = a_{k,j} - \frac{a_{k,i}}{a_{i,i}} a_{i,j}$$

Es stellt sich heraus, dass das System am stabilsten bleibt, wenn

$$\left|\frac{a_{k,i}}{a_{i,i}}\right|$$

möglichst klein ist. Diese Strategie verursacht die geringsten Änderungen am Ausgangssystem. Eine genauere Begründung dafür lernen wir später bei der Untersuchung der Fehlerfortpflanzung kennen. Man wird also versuchen, $|a_{i,i}|$ möglichst groß zu wählen.

Dazu kann man vor dem n-ten Schritt Zeilen und sogar Spalten vertauschen. Das Element, das man mit $a_{i,i}$ vertauscht, nennt man Pivot.

Es gibt verschiedene Heuristiken.

(1) Vertausche die l-te Zeile mit der i-ten, wobei

$$|a_{l,i}| = \max_{k \geq i} |a_{k,i}|$$

Die Pivotsuche findet hier also nur in der i-ten Spalte statt.

(2) Wie (1). Jedoch aufwändiger

$$\frac{|a_{l,i}|}{\sum_j |a_{l,j}|} = \max_{k \geq i} \frac{|a_{k,i}|}{\sum_j |a_{k,j}|}.$$

Dadurch sollen verschieden skalierte Zeilen ausgeglichen werden.

(3) Suche

$$|a_{l,j}| = \max_{k,j \geq i} |a_{k,j}|.$$

Dies nennt man vollständige Pivotsuche. Man muss dann auch Spalten vertauschen, und damit Variablen. Die Vertauschung der Variablen muss natürlich am Schluss berücksichtigt werden.

(4) Nivelliere das Gleichungssystem zu Anfang, so dass alle Zeilen dieselbe L_1-Norm

$$\sum_j |a_{i,j}| = 1$$

haben und wende die teilweise Pivotsuche aus (1) an. Dies ist eine einfache und wirkungsvolle Strategie.

7.2. Beispiel: Wir wollen das Gleichungssystem

$$\begin{pmatrix} 0.005 & 1 \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0.5 \\ 1 \end{pmatrix}$$

mit 2-stelliger Rechnung lösen. Die exakte Lösung ist $x = 0.503$, $y = 0.497$, was auf zwei Stellen gerundet $x = y = 0.5$ ergibt.

(1) Ohne Pivotsuche erhält man mit einem Schritt das System

$$\begin{array}{cc|c} 0.005 & 1 & 0.5 \\ 0 & -200 & -99 \end{array}$$

Also $y = 0.5$ und $x = 0$, wenn man von unten nach oben auflöst. Man beachte, dass man selbst bei exakter Rechnung ausgehend von diesem System die falsche Lösung $y = 0.495$ und $x = 1$ erhält.

(2) Mit Pivotsuche werden zunächst die Zeilen vertauscht und man erhält nach einem Schritt.

$$\begin{array}{cc|c} 1 & 1 & 1 \\ 0 & 1 & 0.5 \end{array}$$

und die richtige Lösung $x = y = 0.5$.

7.3. Beispiel: Der folgende Algorithmus implementiert das Gauß-Verfahren mit Pivotsuche in Java. Man beachte, dass das Vertauschen der Zeilen einfach durch Vertauschen der Zeiger auf die Zeilen realisiert wird.

```
/**
 * Löst Ax=b mit Hilfe des Gauß-Algorithmus
 * @param A Eine quadratische reguläre Matrix
 * @param b Ein Vektor
 * @param x Ergebnisvektor
 */
public static void solveGauss (double[][] A, double[] b, double[] x)
   int n=A.length; // A muss quadratisch sein!
    for (int j=0; j<n-1; j++) // Loop über die Spalten
       // Finde Maximum in der Spalte:
        double max=Math.abs(A[j][j]);
        int imax=j;
        for (int i=j+1; i<n; i++)
           if (Math.abs(A[i][j])>max)
               max=Math.abs(A[i][j]); imax=i;


        // Vertausche Zeilen von A und b
        double[] h=A[j]; A[j]=A[imax]; A[imax]=h;
        double hb=b[j]; b[j]=b[imax]; b[imax]=hb;
        // Multipliziere Vielfache der j-ten Zeile zu
        // den darunter liegenden Zeilen:
        for (int i=j+1; i<n; i++)
           double f=-A[i][j]/A[j][j];
            for (int k=j+1; k<n; k++) A[i][k]+=f*A[j][k];
            b[i]+=f*b[j];


    // Berechne rekursiv x[n-1],...,x[0]
    for (int j=n-1; j>=0; j--)
       x[j]=b[j];
        for (int k=j+1; k<n; k++) x[j]-=A[j][k]*x[k];
        x[j]/=A[j][j];
```

In EMT ist der Gauß-Algorithmus zum Lösen von $Ax = b$ durch $A\backslash b$ implementiert. EMT besitzt jedoch auch bessere Algorithmen.

```
>A := normal(100,100);  // 100x100 Zufallsmatrix
>b := sum(A);  // Summe der Zeilen, so dass die Lösung (1,...1) ist
>longestformat; norm(A*1-1)
  8.623567564250007e-014
```

7.4 Aufgabe: Testen Sie das Gauß-Verfahren mit Hilfe des Java-Programms für Zufallsmatrizen $A \in \mathbb{R}^{n\times n}$ und

$$n = 10, 20, 30, \ldots, 500.$$

Dabei sei

$$a_{i,j} = \texttt{Math.random()}$$

und die rechte Seite sei

$$b_i = \sum_{j=1}^{n} a_{i,j}, \qquad i = 1, \ldots, n.$$

Geben Sie den Fehler bei der Lösung von $Ax = b$ in der Maximumsnorm aus.

Man kann sich sämtliche Vielfache, die verwendet wurden, unterhalb der Diagonale notieren. In einem speziellen Array merkt man sich die Vertauschungen, die man in jedem Schritt vornehmen

musste. Auf diese Weise kann man ein neues Gleichungssystem

$$Ax = \tilde{b}$$

mit anderer rechter Seite leicht lösen, indem man die Vertauschungen und die Zeilenoperationen auf $\tilde{b}$ anwendet.

7.5. Beispiel: Wir führen das anhand der Matrix

$$A = \begin{pmatrix} 2 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 2 \end{pmatrix}$$

vor, und zwar zunächst ohne rechte Seite. In diesem Fall ist eine Pivotsuche wegen der exakten Rechnung nicht erforderlich. Es gibt also auch keine Vertauschungen. Die Faktoren $-1/2$ und $-2/3$ notieren wir an den entsprechenden Stellen unterhalb der Diagonalen.

$$\begin{array}{ccc} 2 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 2 \end{array} \quad \to \quad \begin{array}{ccc} 2 & 1 & 0 \\ -1/2 & 3/2 & 1 \\ 0 & 1 & 2 \end{array} \quad \to \quad \begin{array}{ccc} 2 & 1 & 0 \\ -1/2 & 3/2 & 1 \\ 0 & -2/3 & 4/3 \end{array}$$

Nun wenden wir dieselben Umformungen auf $b = (1, 1, 1)^T$ an.

$$\begin{array}{c} 1 \\ 1 \\ 1 \end{array} \quad \to \quad \begin{array}{c} 1 \\ 1/2 \\ 1 \end{array} \quad \to \quad \begin{array}{c} 1 \\ 1/2 \\ 2/3 \end{array}$$

und erhalten das System

$$\left(\begin{array}{ccc|c} 2 & 1 & 0 & 1 \\ 0 & 3/2 & 1 & 1/2 \\ 0 & 0 & 4/3 & 2/3 \end{array}\right)$$

Man berechnet rekursiv

$$x_3 = 1/2, \quad x_2 = 0, \quad x_1 = 1/2.$$

Natürlich lässt sich das Gaußverfahren auch über jedem anderen Körper K durchführen. Im Fall $K = \mathbb{Q}$ wird man dann exakt rechnen und eine Pivotsuche ist unnötig. Vertauschungen von Zeilen können natürlich trotzdem erforderlich sein.

Startet man mit einer ganzzahligen Matrix $A \in \mathbb{Z}^{n\times n}$, so kann man alle Rechnungen ganzzahlig halten, indem man in jedem Schritt das $a_{k,i}$-fache der i-ten Zeile vom $a_{i,i}$-fachen der k-ten Zeile abzieht. Es empfiehlt sich, dauernd durch den größten gemeinsamen Teiler der Zeilen zu dividieren. Erst ganz am Ende entstehen Brüche.

7.6 Satz: *Wenn die Matrix $A \in \mathbb{K}^{n\times n}$ positiv definit ist, dann kommt das Gauß-Verfahren bei der Berechnung der Lösung von $Ax = b$ ohne Zeilenvertauschungen aus.*

Beweis: Bezeichne A_i die linke obere $i \times i$-Untermatrix von A. Nach dem Satz von Hurwitz sind die Determinanten der Matrizen A_i immer reell und größer als 0.

Wir führen den Beweis durch Induktion über die Schritte des Gaußverfahrens. Zunächst gilt $a_{1,1} = \det A_1 > 0$. Also benötigt man im ersten Schritt keine Vertauschung.

Im i-ten Schritt sieht das Schema so aus.

$$\begin{array}{cccc|c} \tilde{a}_{1,1} & & & * & \tilde{b}_1 \\ & \ddots & & & \\ & & \tilde{a}_{i,i} & & \vdots \\ & & \vdots & & \\ 0 & & \tilde{a}_{n,i} & & \tilde{b}_n \end{array}$$

Die linke obere $i \times i$-Untermatrix bezeichnen wir mit $\tilde{A}_i$. Nach Induktionsvoraussetzung wurde bis zum $i-1$-ten Schritt noch keine Zeilenvertauschung vorgenommen. Das bedeutet aber, dass

$$\det \tilde{A}_i = \det A_i > 0$$

für alle $i = 1, \ldots, n$ gilt. Denn die Addition eines λ fachen der i-ten Zeile zur k-ten Zeile ändert die linken oberen Unterdeterminanten nicht, wenn $i < k$ ist.

Daher gilt

$$\tilde{a}_{1,1} \cdot \ldots \cdot \tilde{a}_{i,i} = \det \tilde{A}_i = \det A_i > 0.$$

Es folgt $\tilde{a}_{i,i} \neq 0$. Also benötigt man auch im i-ten Schritt keine Vertauschung. **q.e.d.**

Offenbar genügt es zu fordern, dass alle linken oberen Untermatrizen eine Determinante ungleich 0 haben.

7.2 Gauß-Jordan-Verfahren

Wir schreiben das System $Ax = b$ mit Variablen x und b in der Form

$$\begin{array}{ccc|c} x_1 & \ldots & x_n & \\ \hline a_{1,1} & \ldots & a_{1,n} & b_1 \\ \vdots & & \vdots & \vdots \\ a_{n,1} & \ldots & a_{n,n} & b_n \end{array}$$

Die Zeilen des Systems sind als

$$a_{i,1}x_1 + \ldots + a_{i,n}x_n = b_i$$

zu lesen $(i = 1, \ldots, n)$.

Nun werden Variablen b_i gegen x_j ausgetauscht, so dass die enstehenden Gleichungen äquivalent sind. Man wählt ein Pivot-Element $a_{i,j} \neq 0$, löst die i-te Gleichung nach x_j auf und setzt den Wert für x_j in die übrigen Zeilen ein. Also

$$x_j = -\frac{a_{i,1}}{a_{i,j}} - \cdots - \frac{a_{i,j-1}}{a_{i,j}}x_{j-1} + \frac{1}{a_{i,j}}b_i - \frac{a_{i,j+1}}{a_{i,j}}x_{j+1} - \cdots - \frac{a_{i,n}}{a_{i,j}}x_n.$$

Man erhält ein Schema

x_1	$\dots$	x_{j-1}	b_i	x_{j+1}	$\dots$	x_n	
							b_1
							$\vdots$
							b_{i-1}
			$\tilde{a}_{i,j}$				x_j
							b_{i+1}
							$\vdots$
							b_n

mit

$$\begin{aligned}
\tilde{a}_{i,j} &= \frac{1}{a_{i,j}}, \\
\tilde{a}_{i,\mu} &= -\frac{a_{i,\mu}}{a_{i,j}}, \qquad \mu \neq j, \\
\tilde{a}_{\nu,j} &= \frac{a_{\nu,j}}{a_{i,j}}, \qquad \nu \neq i, \\
\tilde{a}_{\nu,\mu} &= a_{\nu,\mu} - \frac{a_{\nu,j}a_{i,\mu}}{a_{ij}}, \qquad \nu \neq i, \quad \mu \neq j.
\end{aligned}$$

Dies wird solange fortgeführt, bis alle Variablen $x_1, \dots, x_n$ rechts stehen. Das Pivot-Element $a_{i,j}$ wird stets so gewählt, dass $a_{i,j} \neq 0$ ist und in der Spalte eine x-Variable, in der Zeile aber eine b-Variable steht.

7.7 Satz: *Ein Paar x, b erfüllt genau dann die Gleichungen eines Schrittes des Verfahrens, wenn $Ax = b$ gilt. Wenn die Ausgangs-Matrix A regulär ist ($\det A \neq 0$), so ist das Gauß-Jordan-Verfahren bis zum Ende durchführbar. Sortiert man im letzten Schritt die Zeilen und Spalten, so steht im Schema die inverse Matrix zu A. Es sind höchstens n Schritte notwendig.*

Beweis: Die Äquivalenz rechnet man leicht nach.

Wenn das Verfahren nicht mehr durchführbar ist, dann gibt es eine Zeile, in der eine Variable b_i nur von anderen b-Variablen abhängig ist. Offenbar kann das Gleichungssystem dann nicht immer gelöst werden. Das bedeutet, dass A nicht regulär sein kann.

Im letzten Schritt hat nach der Sortierung die Gleichung $x = \tilde{A}b$. Da diese Gleichung genau dann für alle x und b gilt, wenn $x = A^{-1}b$ ist, folgt $A^{-1} = \tilde{A}$. **q.e.d.**

Auch hier kann man vollständige Pivot-Suche durchführen. Allerdings darf man nur solche $a_{i,j}$ in Betracht ziehen, die eine x-Variable in der Spalte und eine b-Variable in der Zeile haben. Man wählt $|a_{i,j}|$ maximal.

7.8. Beispiel: Mit zweistelliger Rechnung

x_1	x_2	
0.005	1	b_1
1	1	b_2

$\rightarrow$

x_1	b_2	
-1	1	b_1
-1	1	x_2

$\rightarrow$

b_1	b_2	
-1	1	x_1
1	0	x_2

Das Pivot-Element ist jeweils rot markiert. Wir haben also die Gleichungen

$$-b_1 + b_2 = x_1, \quad b_1 = x_2.$$

Einsetzen von $b_1 = 0.5$ und $b_2 = 1$ ergibt die korrekt gerundete Lösung $x_1 = x_2 = 0.5$.

Man kann diesen Algorithmus auch anders darstellen. Das System $Ax = b$ ist nämlich äquivalent zu

$$\left(\begin{array}{c|c} A & I_n \end{array} \right) \cdot \begin{pmatrix} x \\ -b \end{pmatrix} = 0.$$

Der Algorithmus führt nun im Prinzip an $(A|I_n)$ Zeilenoperationen aus, bis $(P|\tilde{A})$ ensteht mit einer Permutationsmatrix P (Die Spalten von P sind die permutierten Einheitsvektoren). Führt man noch Zeilenvertauschungen durch so ensteht (I_n, A^{-1}).

Der Gauß-Jordan-Algorithmus entspricht also der üblichen Berechnung der inversen Matrix. Der Aufwand ist $O(n^3)$, n Schritte mit je n^2 Berechnungen der $\tilde{a}_{\nu,\mu}$.

Beide Darstellung sind in der Optimierung üblich. Dort allerdings gibt es noch Nebenbedingungen, die bei der Wahl des Pivot-Elements berücksichtigt werden müssen.

7.9. Beispiel: In dieser Schreibweise ist das letzte Beispiel so zu lesen

$$\begin{array}{cc|cc} x_1 & x_2 & -b_1 & -b_2 \\ \hline 0.005 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{array} \quad \rightarrow \quad \begin{array}{cc|cc} x_1 & x_2 & -b_1 & b_2 \\ \hline -1 & 0 & 1 & -1 \\ 1 & 1 & 0 & 1 \end{array}$$

$$\rightarrow \quad \begin{array}{cc|cc} x_1 & x_2 & -b_1 & -b_2 \\ \hline 1 & 0 & -1 & 1 \\ 0 & 1 & 1 & 0 \end{array}$$

Dies sind wieder die Gleichungen $x_1 = -b_1 + b_2$ und $x_2 = b_1$.

7.3 LR-Zerlegung

7.10. Definition: Wir definieren die Frobenius-Matrizen

$$F_{i,j}(\lambda) = \begin{pmatrix} 1 & & & \\ & \ddots & & \\ & \lambda & \ddots & \\ & & & 1 \end{pmatrix} \in \mathbb{K}^{n \times n}$$

Dabei steht das das λ in der i-ten Zeile und der j-ten Spalte, und in der Hauptdiagonalen steht 1, sonst überall 0.

Die Matrix

$$\tilde{A} = F_{i,j}(\lambda) \cdot A$$

entsteht aus A, indem man das λ-fache der j-ten Zeile von A zur i-ten Zeile von A addiert. Analog ensteht die Matrix

$$\tilde{A} = A \cdot F_{i,j}(\lambda)$$

aus A, indem man das λ-fache der i-ten Spalte von A zur j-ten Spalte von A addiert.

7.11 Aufgabe: Zeigen Sie

$$F_{i+1,i}(\lambda_{i+1}) \cdot \ldots \cdot F_{n,i}(\lambda_n) = \begin{pmatrix} 1 & & & \\ & \ddots & & \\ & & 1 & \\ & & \lambda_{i+1} & \\ & & \vdots & \ddots \\ & & \lambda_n & 1 \end{pmatrix}.$$

Zeigen Sie außerdem, dass es dabei auf die Reihenfolge der Multiplikation nicht ankommt.

7.12 Aufgabe: Sei

$$F_i(\lambda_{i+1}, \ldots, \lambda_n) := F_{i+1,i}(\lambda_{i+1}) \cdot \ldots \cdot F_{n,i}(\lambda_n)$$

Zeigen Sie

$$F_i(\lambda_{i+1}, \ldots, \lambda_n)^{-1} = F_i(-\lambda_{i+1}, \ldots, -\lambda_n).$$

7.13. Definition: Wir definieren die Permutationsmatrizen $P_{i,j}$ durch die Einheitsmatrix I_n, bei der die i-te und die j-te Spalte (oder Zeile) vertauscht sind.

Die Matrix

$$\tilde{A} = P_{i,j} \cdot A$$

entsteht aus A, indem man die i-te Zeile mit der j-ten Zeile von A vertauscht. Die Matrix

$$\tilde{A} = A \cdot P_{i,j}$$

entsteht aus A, indem man die i-te Spalte mit der j-ten Spalte von A vertauscht. Offenbar gilt

$$P_{i,j}^{-1} = P_{i,j}.$$

Man beachte aber, dass die Reihenfolge von Vertauschungen *nicht* beliebig ist.

7.14 Satz: *(LR-Zerlegung) Wenn man, ausgehend von A, den Gauß-Algorithmus durchführt, indem man die negativen Vielfachen der Zeilenadditionen unterhalb der Diagonalen speichert und die Vertauschungen im gesamten Schema vornimmt, so ensteht eine Matrix*

$$\begin{pmatrix} \tilde{a}_{1,1} & \ldots & \ldots & \tilde{a}_{1,n} \\ \tilde{\lambda}_{2,1} & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ \tilde{\lambda}_{n,1} & \ldots & \tilde{\lambda}_{n,n-1} & \tilde{a}_{n,n} \end{pmatrix}.$$

Seien $P_1, \ldots, P_{n-1}$ die im entsprechenden Schritt verwendete Permutationsmatrix und

$$P = P_{n-1} \cdot \ldots \cdot P_1.$$

Sei außerdem

$$L = \begin{pmatrix} 1 & \ldots & \ldots & 0 \\ \tilde{\lambda}_{1,1} & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ \tilde{\lambda}_{n,1} & \ldots & \tilde{\lambda}_{n,n-1} & 1 \end{pmatrix},$$

und

$$R = \begin{pmatrix} \tilde{a}_{1,1} & \dots & \dots & \tilde{a}_{1,n} \\ 0 & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & \tilde{a}_{n,n} \end{pmatrix}.$$

Dann gilt

$$L \cdot R = P \cdot A.$$

Beweis: Bezeichne

$$F_i = F_i(\lambda_{i+1,i}, \dots, \lambda_{n,i}) := F_{i+1,i}(\lambda_{i+1,i}) \cdot \dots \cdot F_{n,i}(\lambda_{n,i})$$

die Matrix der Zeilenoperationen im i-ten Schritt. Dann ist der Gauß-Algorithmus äquivalent zu der Darstellung

$$F_{n-1}P_{n-1} \cdot \dots \cdot F_1 P_1 A = R.$$

Also

$$P_1 A = F_1^{-1} P_2 \cdot \dots \cdot P_{n-1} F_{n-1}^{-1} R.$$

Wir müssen also zeigen, dass

$$L = P_{n-1} \cdot \dots \cdot P_2 F_1^{-1} P_2 \cdot \dots \cdot P_{n-1} F_{n-1}^{-1}$$

gilt.

Nun bezeichnen wir mit $\tilde{F}_k$ die linke untere Dreiecksmatrix mit 1 in der Diagonalen und den $-\lambda_{i,j}$ bis zum k-ten Schritt, inklusive der Vertauschungen, die in den vorigen Schritten an diesen Konstanten vorgenommen wurden. Zunächst gilt

$$\tilde{F}_1 = F_1^{-1}.$$

Im k-ten Schritt wird eine Vertauschung von zwei Zeilen der vorhandenen $-\lambda_{i,j}$ in $\tilde{F}_{k-1}$ vorgenommen, und man erhält die Matrix

$$P_k \tilde{F}_{k-1} P_k$$

(Durch die Multiplikation mit P_k von rechts wird die Vertauschung der Einheitsvektoren rechts der k-ten Spalte rückgängig gemacht.) Insgesamt sieht man

$$\tilde{F}_k = P_k \tilde{F}_{k-1} P_k F_k^{-1}.$$

Im letzten Schritt erhalten wir $\tilde{F}_{n-1} = L$. Also

$$P_{n-1} \cdot \dots \cdot P_2 F_1^{-1} P_2 F_2^{-1} \cdot \dots \cdot P_{n-1} F_{n-1}^{-1} = \tilde{F}_{n-1} = L.$$

q.e.d.

Hat man eine LR-Zerlegung

$$LR = PA$$

dann ist ein Gleichungssystem $Ax = b$ äquivalent zu

$$LRx = Pb,$$

also zu einem gestaffelten Gleichungssystem

$$Ly = Pb, \quad Rx = y.$$

Beide Gleichungssysteme sind mit einem Aufwand von $O(n^2)$ lösbar, weil die beteiligten Matrizen untere und obere Dreiecksmatrizen sind. Der Aufwand des Gaußalgorithmus ist dagegen $O(n^3)$.

Zusammengefasst kann man also, wenn man die Zerlegung kennt, die Lösung auf folgende Weise ermitteln:

1. Führe die Vertauschungen $P_1, \ldots, P_{n-1}$ an b aus.
2. Löse $Ly = Pb$ von y_1 bis y_n rekursiv auf.
3. Löse $Rx = y$ von x_n bis x_1 rekursiv auf.

7.15. Beispiel: Sei

$$A = \begin{pmatrix} 0 & 1 & 2 \\ 1 & 2 & 1 \\ 2 & 1 & 0 \end{pmatrix}$$

Wir führen eine Vertauschung mit der Permutationsmatrix $P_{1,3}$ aus, und erhalten die Matrix A aus Beispiel 7.5. Also

$$\begin{pmatrix} 1 & 0 & 0 \\ 1/2 & 1 & 0 \\ 0 & 2/3 & 1 \end{pmatrix} \cdot \begin{pmatrix} 2 & 1 & 0 \\ 0 & 3/2 & 1 \\ 0 & 0 & 4/3 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 & 2 \\ 1 & 2 & 1 \\ 2 & 1 & 0 \end{pmatrix}$$

Das Verfahren ist in EMT in `lu` so implementiert, dass die Rückgabe möglichst wenig Platz verbraucht. Eine komfortablere Funktion, die die Matrizen zurückgibt, ist `LU`.

```
>fracformat(10);
>A := [0,1,2;1,2,1;2,1,0]
         0         1         2
         1         2         1
         2         1         0
>B,r,c,det=lu(A); det,
        -4
>P := id(3)[r]
         0         0         1
         0         1         0
         1         0         0
>L := band(B[r],-3,-1)+id(3)
         1         0         0
       1/2         1         0
         0       2/3         1
>R := band(B[r],0,3)
         2         1         0
         0       3/2         1
```

```
         0         0       4/3
>L.R-P.A
         0         0         0
         0         0         0
         0         0         0
>L,R,P=LU(A);  // one step
```

Die Rückgabe von `lu` kann wiederum verwendet werden, um Gleichungssysteme zu lösen.

```
         0         0         0
>b := sum(A);
>lusolve(B[r],b[r])  // solve after permutation
         1
         1
         1
```

7.16. Beispiel: Das folgende Java-Unterprogramm berechnet die LR-Zerlegung einer regulären Matrix $A \in \mathbb{R}^{n\times n}$.

```
/**
 * Zerlegt A in der Form LR=PA.
 * L hat 1 in der Diagonalen und die Elemente links unten von A.
 * R besteht aus die Diagonalen und den Elementen oberhalb in A.
 * Die Vertauschungen von P stehen in perm.
 *
 * @param A Eine quadratische reguläre Matrix
 * @param perm Nimmt die Permutationen auf (n-1 Indizes)
 */
public static void constructLR (double[][] A, int[] perm)
   int n=A.length; // A muss quadratisch sein!
    for (int j=0; j<n-1; j++) // Loop über die Spalten
       // Finde Maximum in der Spalte:
        double max=Math.abs(A[j][j]);
        int imax=j;
        for (int i=j+1; i<n; i++)
           if (Math.abs(A[i][j])>max)
               max=Math.abs(A[i][j]); imax=i;


        // Vertausche Zeilen von A und b,
        // einschließlich alter Faktoren:
        double[] h=A[j]; A[j]=A[imax]; A[imax]=h;
        // Notiere Permutation:
        perm[j]=imax;
        // Multipliziere Vielfache der j-ten Zeile zu
        // den darunter liegenden Zeilen:
        for (int i=j+1; i<n; i++)
           double f=-A[i][j]/A[j][j];
            for (int k=j+1; k<n; k++) A[i][k]+=f*A[j][k];
            A[i][j]=-f; // Speichere den Faktor
```

Ein weiteres Unterprogramm berechnet die Lösung von $Ax = b$ mit Hilfe dieser Zerlegung.

```
/**
 * Löst ein LR-zerlegtes Gleichungssystem.
 *
 * @param A Matrix mit L unterhalb der Diagonalen
 * @param b rechte Seite des LGS.
 * @param perm Notwendige Permutationen (n-1 Indizes)
 * @param x Vektor für die Lösung
 */
public static void solveLR (double[][] A, double[] b,
    int[] perm, double[] x)
   int n=A.length;
    // Führe Permutationen an b aus:
    for (int i=0; i<n-1; i++)
       double h=b[perm[i]]; b[perm[i]]=b[i]; b[i]=h;

    // Löse Ly=b, wobei y auf x gespeichert wird:
    for (int j=0; j<n; j++)
       x[j]=b[j];
        for (int i=0; i<j; i++) x[j]-=A[j][i]*x[i];

    // Löse Rx=y:
    for (int j=n-1; j>=0; j--)
       for (int k=j+1; k<n; k++) x[j]-=A[j][k]*x[k];
        x[j]/=A[j][j];
```

Für positiv definite Matrizen ist eine einfachere Zerlegung der Form $A = R^T R$ möglich. Gleichungssysteme mit dieser Zerlegung zu lösen, wird Cholesky-Verfahren genannt.

7.17 Satz: *Für jede positiv definite Matrix $A \in \mathbb{R}^{n\times n}$ existiert eine rechte obere Dreiecksmatrix R, so dass*

$$A = R^T R$$

ist.

Beweis: Nach Satz 7.6 ist es möglich, A ohne Zeilenvertauschungen auf Diagonalgestalt zu bringen. Nach dem Beweis der Existenz der LR-Zerlegung bedeutet dies, dass sich A in der Form

$$A = \tilde{L}\tilde{R}$$

darstellen lässt mit linken unteren bzw. rechten oberen Dreiecksmatrizen $\tilde{L}$ und $\tilde{R}$, wobei $\tilde{L}$ nur 1 in der Diagonalen hat. Bezeichnet A_i, $\tilde{L}_i$ und $\tilde{R}_i$ die linke obere $i \times i$-Untermatrix dieser Matrizen, so gilt offenbar

$$A_i = \tilde{L}_i \tilde{R}_i.$$

Für die Diagonalelemente $r_1, \ldots, r_n$ von $\tilde{R}$ folgt

$$0 < \det(A_i) = \det(\tilde{R}_i) = r_1 \cdot \ldots \cdot r_i.$$

Also $r_1, \ldots, r_n > 0$. Man definiert nun die Diagonalmatrix

$$D = \begin{pmatrix} \sqrt{r_1} & & 0 \\ & \ddots & \\ 0 & & \sqrt{r_n} \end{pmatrix}$$

und

$$L = \tilde{L}D, \quad R = D^{-1}\tilde{R}.$$

Dann gilt $A = LR$. R ist eine linke untere Dreiecksmatrix, L eine rechte obere Dreiecksmatrix. Die Diagonalen von L und R stimmen überein und enthalten beide die Elemente $\sqrt{r_1}, \ldots, \sqrt{r_n}$. Aus der folgenden Aufgabe folgt $L = R^T$ **q.e.d.**

7.18 Aufgabe: Sei $L \in \mathbb{R}^{n\times n}$ eine linke untere Dreiecksmatrix, $R \in \mathbb{R}^{n\times n}$ eine rechte obere Dreiecksmatrix, die in der Diagonale mit L übereinstimmt. Sei LR symmetrisch. Zeigen Sie $L = R^T$.

7.19. Beispiel: Der Sachverhalt läuft also darauf hinaus, eine gewöhnliche LR-Zerlegung einer symmetrischen Matrix A zu berechnen. Wenn keine Zeilenpermutation nötig war, und die Diagonalelemente von R positiv sind, so ist auch A positiv definit. Dividiert man R zeilenweise durch die Wurzeln entsprechenden Diagonalelemente, so hat man eine Zerlegung $A = R^T R$ gefunden.

Diese Zerlegung ist daher in EMT mit Hilfe von `lu` implementiert.

```
>shortestformat;
>A := [2,1,0;1,2,1;0,1,2]
         2         1         0
         1         2         1
         0         1         2
>L := cholesky(A)
      1.41         0         0
     0.707      1.22         0
         0     0.816      1.15
>L.L'
         2         1         0
         1         2         1
         0         1         2
>lsolve(L,sum(A))
         1
         1
         1
```

7.4 Fehlerfortpflanzung

Wir verwenden in diesem Abschnitt die in Kapitel 4 hergeleiteten Matrixnormen.

7.20 Satz: *Sei $A \in \mathbb{K}^{n\times n}$ regulär, und $x \in \mathbb{K}^n$ die Lösung des Gleichungssystems*

$$Ax = b$$

Sei nun $x + \Delta x \in \mathbb{K}^n$ die Lösung des Gleichungssystems

$$A(x + \Delta x) = b + \Delta b$$

für $\Delta b \in \mathbb{K}^n$, dann gilt

$$\frac{||\Delta x||}{||x||} \leq ||A||\,||A^{-1}||\frac{||\Delta b||}{||b||}.$$

Dieser Satz gibt an, wie stark sich ein relativer Messfehler oder anderer Fehler der rechten Seite auf den relativen Fehler der Lösung auswirkt.

Beweis: Offenbar

$$\Delta x = A^{-1}\Delta b.$$

Also

$$\|\Delta x\| \leq \|A^{-1}\|\,\|\Delta b\|.$$

Wegen $\|b\| \leq \|A\|\,\|x\|$ folgt

$$\frac{\|\Delta x\|}{\|x\|} \leq \frac{\|A^{-1}\|\|\Delta b\|}{\|A\|^{-1}\|b\|} = \|A\|\,\|A^{-1}\|\frac{\|\Delta b\|}{\|b\|}.$$

q.e.d.

7.21. Definition: Man bezeichnet für reguläre quadratische Matrizen die Größe

$$\mathrm{cond}(A) := \|A\|\,\|A^{-1}\|$$

als Konditionszahl von A.

7.22 Satz: *Seien $\lambda_1, \ldots, \lambda_k$ die* komplexen *Eigenwerte der regulären Matrix A. Dann gilt*

$$\mathrm{cond}(A) \geq \frac{\max_\nu |\lambda_\nu|}{\min_\nu |\lambda_\nu|}.$$

Beweis: Nach Aufgabe 4.42 gilt

$$\|A\| \geq \sigma(A) = \max_\nu |\lambda_\nu|.$$

Die Matrix A^{-1} hat die komplexen Eigenwerte $1/\lambda_i$, $i = 1, \ldots, k$. Also

$$\|A^{-1}\| \geq \max_\nu |\frac{1}{\lambda_\nu}| = \frac{1}{\min_\nu |\lambda_\nu|}.$$

Es folgt die Behauptung. **q.e.d.**

7.23. Beispiel: Ein Beispiel für gut konditionierte Matrizen sind orthogonale Matrizen. Denn, wenn man die Euklidsche Norm zugrunde legt, so gilt für eine orthogonale Matrix

$$\|Ax\| = \|x\| \qquad \text{für alle } x \in \mathbb{R}^m$$

Also $\|A\| = \|A^{-1}\| = 1$. Die Konditionszahl ist also 1.

7.24. Beispiel: Eine schlecht konditionierte Matrix ist die Hilbertmatrix

$$H_5 = \begin{pmatrix} 1 & 1/2 & \ldots & 1/n \\ 1/2 & 1/3 & & \\ \vdots & & & \vdots \\ 1/n & & \ldots & 1/(2n-1) \end{pmatrix} = \left(\frac{1}{i+j-1}\right)_{1\leq i,j\leq n}$$

Mit der Zeilensummennorm berechnet EMT

$$\mathrm{cond}(H_5) = 943656, \quad \mathrm{cond}(H_{10}) \approx 2.53 \cdot 10^{13}.$$

Im folgenden EMT-Code werden zur Sicherheit noch Einschließungen für die Konditionszahlen berechnet.

```
>norm(inv(hilbert(5)))*norm(hilbert(5))
       943656
>norm(iinv(hilbert(5)))*norm(hilbert(5))
      ~943655.999999999,943656.000000001~
>norm(inv(hilbert(10)))*norm(hilbert(10))
 3.53576e+013
>norm(iinv(hilbert(10)))*norm(hilbert(10))
    ~35357439251991.96,35357439251992.04~
```

7.25 Satz: *Sei $F \in \mathbb{K}^{n\times n}$ mit $\|F\| < 1$, und I die Einheitsmatrix in $\mathbb{K}^n$. Dann ist $I - F$ invertierbar und*

$$\|(I-F)^{-1}\| \le \frac{1}{1-\|F\|}.$$

Beweis: Angenommen, $(I - F)x = 0$, $x \neq 0$. Es folgt $x = Fx$ und

$$\|x\| = \|Fx\| \le \|F\|\,\|x\| < \|x\|.$$

Also $x = 0$. Daher ist $I - F$ invertierbar.

Durch Nachrechnen sieht man

$$(I-F)(I+F+F^2+\cdots+F^n) = I - F^{n+1}.$$

Es gilt

$$\|F^\nu\| \le \|F\|^\nu$$

Also

$$\begin{aligned}\|(I-F)^{-1}(I-F^{n+1})\| &\le \|I+F+\cdots+F^n\| \\ &\le 1+\|F\|^2+\cdots+\|F^n\| \\ &\le \frac{1}{1-\|F\|}.\end{aligned}$$

Mit Hilfe der Dreiecksungleichung folgt

$$\begin{aligned}\|(I-F)^{-1}(I-F^{n+1})\| &\ge \|(I-F)^{-1}\| - \|(I-F)^{-1}F^{n+1}\| \\ &\ge \|(I-F)^{-1}\| - \|(I-F)^{-1}\|\,\|F\|^{n+1}\end{aligned}$$

Die rechte Seite konvergiert gegen $\|(I - F)^{-1}\|$ für $n \to \infty$. Es folgt die Behauptung. **q.e.d.**

Der folgende Satz gibt an, wie stark sich ein relativer Fehler der Matrix auf die Lösung auswirkt. Wieder spielt die Konditionszahl von A die entscheidende Rolle.

7.26 Satz: *Sei $A \in \mathbb{K}^{n\times n}$ regulär, und $x \in \mathbb{K}^n$ die Lösung des Gleichungssystems*

$$Ax = b.$$

Sei außerdem

$$(A + \Delta A)(x + \Delta x) = b$$

mit einer Störungsmatrix ΔA, die so klein sei, dass

$$\|A^{-1}\|\,\|\Delta A\| < 1$$

ist. Dann gilt

$$\frac{\|\Delta x\|}{\|x\|} \leq \frac{\mathrm{cond}(A)}{1 - \|A^{-1}\|\,\|\Delta A\|} \frac{\|\Delta A\|}{\|A\|}.$$

Beweis: Wir definieren

$$F = A^{-1}\,\Delta A$$

und $B = A + \Delta A$. Also

$$B = A(I + F).$$

Dann gilt

$$\Delta x = B^{-1}b - A^{-1}b = B^{-1}(A - B)A^{-1}b = B^{-1}(A - B)x.$$

Also wegen Satz 7.25

$$\frac{\|\Delta x\|}{\|x\|} \leq \|B^{-1}(A - B)\| = \|(I + F)^{-1}A^{-1}\Delta A\| = \|(I + F)^{-1}F\|.$$

Wir haben nun vorausgesetzt, dass

$$\|-F\| = \|F\| \leq \|A^{-1}\|\,\|\Delta A\| < 1$$

ist. Es folgt, dass $I - (-F) = I + F$ invertierbar ist und

$$\|(I + F)^{-1}\| \leq \frac{1}{1 - \|F\|}.$$

Also

$$\frac{\|\Delta x\|}{\|x\|} \leq \frac{\|F\|}{1 - \|F\|}$$

Wegen $\|F\| \leq \|A^{-1}\|\,\|\Delta A\|$ folgt die Behauptung. **q.e.d.**

Die im Laufe des Gauß-Verfahrens auftretenden Rundungsfehler kann man als Störungen der Ausgangsmatrix A interpretieren. Die Qualität des Ergebnisses hängt also auch hier von der Konditionszahl ab. Die beim Gauß-Verfahren verwendeten Frobenius-Matrizen liegen nun möglichst nahe bei I_n, wenn die Faktoren

$$|a_{k,i}|/|a_{i,i}|$$

möglichst klein sind. Damit erklärt man den Erfolg der Heuristiken für die Pivotsuche.

7.27. Beispiel: Das in EMT implementierte Gauß-Verfahren mit teilweiser Pivotsuche liefert sehr schlechte Ergebnisse, obwohl die Hilbertmatrix so skaliert wird, dass die ganzzahlig, also exakt, im Rechner vorliegt. Dass es nicht an der Matrix oder der rechten Seite liegt, beweist eine Vergleichsrechnung mit Intervalleinschließung, einem Verfahre, dass wir später behandeln werden.

```
>b=sum(hilbert(10)); hilbert(10)_          1
            1
            1
     0.999996
      1.00002
      0.99995
      1.00008
     0.999921
      1.00004
     0.999991
>ilgs(hilbert(10),b)
 ~0.99999999999999978,1.0000000000000002~
 ~0.99999999999999978,1.0000000000000002~
 ~0.99999999999999978,1.0000000000000002~
 ~0.99999999999999978,1.0000000000000002~
 ~0.99999999999999978,1.0000000000000002~
 ~0.99999999999999989,1.0000000000000002~
 ~0.99999999999999978,1.0000000000000002~
 ~0.99999999999999978,1.0000000000000002~
 ~0.99999999999999978,1.0000000000000002~
 ~0.99999999999999978,1.0000000000000002~
```

7.28 Aufgabe: Sei x eine Lösung von $Ax = b$ mit regulärem $A \in \mathbb{R}^{n\times n}$, $A + \Delta A$ ebenfalls regulär, und

$$(A + \Delta A)(x + \Delta x) = b + \Delta b.$$

Wir nehmen an, dass

$$\frac{\|\Delta b\|}{\|b\|} < \epsilon, \quad \frac{\|\Delta A\|}{\|A\|} < \epsilon.$$

Beweisen Sie

$$\frac{\|\Delta x\|}{\|x\|} \leq C \ \mathrm{cond}(A)\, \epsilon.$$

mit

$$C = \frac{2}{1 - \|A^{-1}\|\,\|\Delta A\|}.$$

7.5 Residuen-Iteration

Sei $x_0 \in \mathbb{R}^m$ eine Näherungslösung von $Ax = b$, wobei A wieder eine reelle $m \times m$-Matrix und $b \in \mathbb{R}^m$ sei.

7.29. Definition: Wir berechnen das Residuum

$$r_0 = Ax_0 - b.$$

Das Residuum wird nicht exakt 0 sein, weil x_0 nur eine Näherungslösung ist. Wir berechnen nun eine Näherungslösung d_0 von

$$Ad = r_0$$

und setzen

$$x_1 = x_0 - d_0.$$

Dieses Verfahren kann man iterativ fortsetzen. Man nennt das Verfahren Residueniteration.

Man kann den Rechenvorgang in Matrixschreibweise wiedergeben, wenn man zum Auflösen des Gleichungssystems $Ad = r_0$ eine Näherungsinverse R von A verwendet, also $d_0 = Rr_0$. Damit gilt

$$\begin{aligned} x_1 &= x_0 - Rr_0 \\ &= x_0 - R(Ax_0 - b) \\ &= x_0 - RAx_0 + Rb \\ &= (I - RA)x_0 + Rb. \end{aligned}$$

Für die Abbildung

$$\phi(x) := (I - RA)x + Rb$$

gilt

$$\|\phi(x) - \phi(y)\| = \|(I - RA)(x - y)\| \leq \|I - RA\|\|x - y\|.$$

Das Iterationsverfahren

$$x_{n+1} = \phi(x_n)$$

konvergiert also nach dem Banachschen Fixpunktsatz, wenn

$$\|I - RA\| < 1$$

ist. Bei einer exakten Inversen wäre diese Norm natürlich 0.

7.30 Aufgabe: Zeigen Sie mit Hilfe von Satz 7.25, dass A regulär sein muss, wenn $\|I - RA\| < 1$ gilt.

Wenn d_0 die korrekte Lösung von $Ad = r_0$ und r_0 das exakte Residuum wäre, so hätte man

$$Ax_1 = Ax_0 - Ad_0 = Ax_0 - r_0 = b.$$

Also wäre auch x_1 die korrekte Lösung von $Ax = b$.

Die Residueniteration führt nur dann zum Erfolg, wenn das Residuum exakt berechnet wird. Wenn x_0 hinreichend genau ist, so ist

$$Ax_0 \approx b_0$$

und beim Berechnen des Residuums tritt Auslöschung ein. Ein schlecht berechnetes Residuum bringt aber die Konvergenz zum Scheitern. Es genügt, das Residuum mit doppelter Genauigkeit auszurechnen.

Die Residueniteration ist also ein Verfahren, bei dem man die höhere Genauigkeit nur an einer Stelle benötigt und sonst mit einfacher Genauigkeit rechnen kann.

7.31. Beispiel: Wir berechnen die Residueniteration mit EMT und der eingebauten exakten Funktion residuum(). Diese Funktion berechnet das Residuum exakt und rundet es erst zur Ausgabe.

```
>longformat; H=hilbert(12); b=sum(H);
>x0=H_          1.000000014847
        0.9999980996484
        ...
        0.9911483089111
>r0=residuum(H,x0,b)
     3.15413643745e-005
    6.901069340565e-006
    ...
   -4.320282939729e-007
>x1=x0-HÔ
         1.000000000104
        0.9999999865406
        ...
        0.9999338762051
>r1=residuum(H,x1,b);
>x2=x1-Hİ;
>r2=residuum(H,x2,b);
>x3=x2-Hˆ2
                      1
        0.9999999999993
        ...
        0.9999999963622
```

Dieser Algorithmus ist in EMT in der Funktion `xlgs` implementiert.

```
>xlgs(H,b)
                    1
                    1
                    1
                    1
                    1
                    1
                    1
                    1
                    1
                    1
                    1
                    1
```

7.32 Aufgabe: Sei

$$A = \begin{pmatrix} 1 & 1 \\ c & 1 \end{pmatrix}, \quad R = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$$

Zeigen Sie, dass die Residueniteration

$$x_{n+1} = x_n - R^{-1}(Ax_n - b)$$

für $|c| < 1$ konvergiert.

7.6 Intervall-Einschließung

Wir rechnen in diesem Abschnitt mit reellen kompakten Intervallen $I \subseteq \mathbb{R}$. Statt Vektoren betrachten wir Intervall-Vektoren

$$\mathcal{X} = I_1 \times \ldots \times I_n \subseteq \mathbb{R}^n,$$

mit reellen kompakten Intervallen $I_1, \ldots, I_n$. Analog betrachten wir Matrizen $\mathcal{A}$ von Intervallen, die einfach als Mengen von Matrizen $A_0 \in \mathcal{A}$ betrachtet werden können.

7.33. Definition: Sei $\mathcal{A}$ eine reelle Intervallmatrix und $\lfloor$ ein reeller Intervallvektor. $\mathcal{X}$ heißt Einschließung für die Lösungen des Gleichungssystems

$$\mathcal{A}X = \mathcal{B},$$

wenn

$$\{x \in \mathbb{R}^n : Ax = b \text{ für ein } A \in \mathcal{A} \text{ und } b \in \mathcal{B}\} \subseteq \mathcal{X}$$

gilt.

Nicht jedes Element $x \in \mathcal{X}$ muss nach dieser Definition Lösung eines Gleichungssystems $Ax = b$, $A \in \mathcal{A}$ und $b \in \mathcal{B}$, sein. Wir verlangen lediglich, dass alle Lösungen eingeschlossen werden.

7.34. Beispiel: Wir suchen die Lösung des Gleichungssystem $\mathcal{A}x = \mathcal{B}$ mit

$$\mathcal{A} = \begin{pmatrix} [0.99, 1.01] & [0.99, 1.01] \\ [0.99, 1.01] & [1.99, 2.01] \end{pmatrix}, \quad \mathcal{B} = \begin{pmatrix} [0.99, 1.01] \\ [0.99, 1.01] \end{pmatrix}.$$

Man kann versuchen, dieses System mit einem intervallmäßig durchgeführten Gauß-Verfahren lösen, indem man die erste Zeile von der zweiten Zeile intervallmäßig subtrahiert.

```
>iformat(20);
>a := ~0.99,1.01~
 ~0.99,1.01~
>A := [a,a;a,a+1]
          ~0.99,1.01~          ~0.99,1.01~
          ~0.99,1.01~          ~1.99,2.01~
>b := [a;a]
          ~0.99,1.01~
          ~0.99,1.01~
>M := A|b
          ~0.99,1.01~          ~0.99,1.01~          ~0.99,1.01~
          ~0.99,1.01~          ~1.99,2.01~          ~0.99,1.01~
>M[2] := M[2]-M[1]
          ~0.99,1.01~          ~0.99,1.01~          ~0.99,1.01~
         ~-0.02,0.02~          ~0.98,1.02~         ~-0.02,0.02~
```

Um dieses System allerdings aufzulösen, müssen wir eine Abschätzung für x_1 machen. Wir nehmen $|x_1| \leq 1.5$ an. Dadurch ist es möglich, eine Einschließung für x_2 und dann für x_1 zu bekommen.

```
>x2 := (M[2,3]-M[2,1]*~-1.5,1.5~)/M[2,2]
 ~-0.051,0.051~
>x1 := (M[1,3]-M[1,2]*x2)/M[1,1]
 ~0.92,1.1~
```

7.35 Aufgabe: Beweisen Sie mit Hilfe des Browerschen Fixpunktsatzes für $n = 1$, warum die obige Rechnung zeigt, dass tatsächlich eine Lösungseinschließung vorliegt.

7.36. Beispiel: Alternativ kann man eine exakte Formel für die Lösung verwenden, etwa die Cramersche Regel.

```
>detA := A[1,1]-A[2,2]-A[1,2]*A[2,1]
 ~-2.05,-1.96~
>x1 := (b[1]-A[2,2]-A[1,2]*b[2])/detA
 ~0.96,1.05~
>x2 := (A[1,1]-b[2]-b[1]*A[2,1])/detA
 ~0.47,0.531~
```

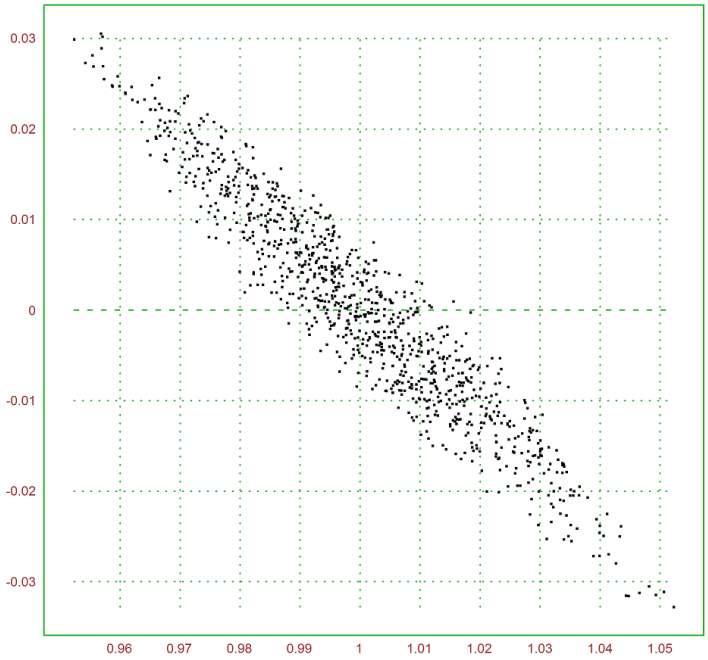

Abbildung 7.1: Zufallslösungen eines Intevallsystems

7.37. Beispiel: Als weitere Möglichkeit, die Lösungsmenge abzuschätzen, kann man sehr viele zufällig ausgewählte Gleichungssysteme $Ax = b$ mit $A \in \mathcal{A}$, $b \in \mathcal{B}$ lösen, und die Maxima und Minima von x_1, x_2 als Intervallgrenzen für $\mathcal{X}$ nehmen. Man ermittelt auf diese Weise die Punktwolke in Abbildung 7.1, die die Lösungseinschließung

$$x_1 \in [-0.04, 0.04], \quad x_2 \in [-0.95, 1.055]$$

nahe legt.

```
>function randomAb (A,b,n)...
$  res=zeros(0,2);
$  loop 1 to n;
$    A0=random(size(A))*diameter(A)+left(A);
$    b0=random(size(b))*diameter(b)+left(b);
$    res=res_(A0()');
$  end;
$  return res;
$endfunction
>x := randomAb(A,b,1000)';
>plot2d(x[1],x[2],>points,style=".");
```

7.38. Beispiel: Wir betrachten unser Intervallsystem als gestörtes System

$$\begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix} \cdot x = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

mit exakter Lösung

$$x = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

Die Störungen der Matrix alleine bewirkt in diesem Fall nach Aufgabe 7.28 einen relativen Fehler von

$$\frac{\|\Delta x\|}{\|x\|} \leq \frac{2\,\mathrm{cond}(A)}{1 - \|A^{-1}\|\,\|\Delta A\|} \frac{\|\Delta A\|}{\|A\|} = \frac{2\|A^{-1}\|\,\|\Delta A\|}{1 - \|A^{-1}\|\,\|\Delta A\|} \leq 0.15,$$

wobei mit der Zeilensummennorm $\|A^{-1}\| = 3$ und $\|\Delta A\| = 0.02$ gerechnet wurde. Leider ist eine solche Abschätzung der Kondition im allgemeinen nicht möglich.

7.39 Satz: *Sei $\mathcal{B}$ ein Intervallvektor und $\mathcal{A}$ eine Intervallmatrix, die nur reguläre Matrizen enthalte. Wir bestimmen eine reguläre Näherungsinverse R_0 zu einer festen Matrix $A_0 \in \mathcal{A}$. Falls dann für einen Intervallvektor $\mathcal{X}$*

$$(I - R_0\mathcal{A})\mathcal{X} + R_0\lfloor \subseteq \mathcal{X}$$

ist, so enthält $\mathcal{X}$ für jedes $A \in \mathcal{A}$ und $b \in \mathcal{B}$ eine Lösung von $Ax = b$.

Beweis: Wir fixieren ein $A \in \mathcal{A}$ und ein $b \in \mathcal{B}$. Dann gilt

$$(I - R_0 A)\mathcal{X} + R_0 b \subseteq \mathcal{X}.$$

Die stetige Abbildung

$$x \mapsto (I - R_0 A)x + R_0 b$$

hat nach dem Browerschen Fixpunktsatz einen Fixpunkt $x \in \mathcal{X}$. Es folgt

$$(I - R_0 A)x + R_0 b = x.$$

Also

$$R_0(-Ax + b) = 0.$$

Weil R_0 invertierbar vorausgesetzt ist, ist dies äquivalent zu $Ax = b$. Also enthält $\mathcal{X}$ eine Lösung von $Ax = b$. **q.e.d.**

7.40 Satz: *Falls im vorgehenden Satz $\mathcal{A}$ eine allgemeine Intervallmatrix ist und R_0 eine beliebige Matrix, und falls*

$$(I - R_0\mathcal{A})\mathcal{X} + R_0\mathcal{B} \subseteq \mathcal{X}^o$$

gefordert wird, so folgt automatisch, dass $\mathcal{A}$ nur reguläre Matrizen enthält, und R_0 invertierbar ist. Dabei bezeichnet $\mathcal{X}^o$ das offene Innere von $\mathcal{X}$.

Beweis: Sei $A \in \mathcal{A}$, $b \in \mathcal{B}$ und x wieder ein Fixpunkt von

$$x \mapsto (I - R_0 A)x + R_0 x$$

Falls dann $R_0 A$ nicht invertierbar ist, so existiert ein $h \neq 0$ mit $R_0 A h = 0$. Es folgt

$$(I - R_0 A)(x + \lambda h) + R_0 b = x + \lambda h.$$

für alle $\lambda \in \mathbb{R}$. Wir wählen λ so, dass $x + \lambda h$ auf dem Rand von $\mathcal{X}$ liegt. Man erhält sofort einen Widerspruch zu

$$(I - R_0 A)\mathcal{X} + R_0 b \subseteq \mathcal{X}^o$$

q.e.d.

7.41. Beispiel: Wir wählen

$$A_0 = \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix} \in \mathcal{A} = \begin{pmatrix} [0.99, 1.01] & [0.99, 1.01] \\ [0.99, 1.01] & [1.99, 2.01] \end{pmatrix}.$$

und

$$\lfloor = \begin{pmatrix} [0.99, 1.01] \\ [0.99, 1.01] \end{pmatrix}.$$

Wir verwenden

$$R = A_0^{-1} = \begin{pmatrix} 2 & -1 \\ -1 & 1 \end{pmatrix}.$$

Nun iterieren wir

$$\mathcal{X}_{n+1} = (I - R\mathcal{A})\mathcal{X}_n + R\mathcal{B}.$$

bis

$$\mathcal{X}_{n+1} \subseteq \mathcal{X}_n$$

mit

$$\mathcal{X}_0 = \begin{pmatrix} [-5, 5] \\ [-5, 5] \end{pmatrix}$$

ist dies nach einem Schritt der Fall. Man kann von nun an

$$\mathcal{X}_{n+1} = ((I - R\mathcal{A})\mathcal{X}_n + R\mathcal{B}) \cap \mathcal{X}_n$$

setzen, bis $\mathcal{X}_{n+1} = \mathcal{X}_n$ gilt.

```
>R := inv(middle(A))
             2            -1
            -1             1
>X := [~-5,5~;~-5,5~]
               ~-5,5~
               ~-5,5~
>X := (id(2)-R.A).X+R.b
          ~0.67,1.33~
         ~-0.22,0.22~
>repeat Xn=((id(2)-R.A).X+R.b)&&X; until all(Xn==X); X=Xn; end;
>X
       ~0.9368,1.064~
       ~-0.042,0.042~
```

Dieser Algorithmus ist in EMT als `ilgs` implementiert.

```
>ilgs(A,b)
       ~0.9368,1.064~
       ~-0.042,0.042~
```

Es ist wichtig festzustellen, dass es mit diesem Verfahren möglich ist auf dem Computer zu beweisen, dass jedes Gleichungssystem $Ax = b$ mit $A \in \mathcal{A}$ und $b \in \lfloor$ eine Lösung hat, und dass jede dieser Lösungen in $\mathcal{X}$ liegt.

7.7 Auswertung von Polynomen

Mit Hilfe dieser Verfahren kann man auch Einschließungen für Werte von Polynomen gewinnen. Sei

$$p(x) = a_0 + a_1 x + \ldots + a_n x^n$$

ein reelles Polynom. Wertet man p mit dem Hornerschema aus, so berechnet man

$$s_0 = a_n, \quad s_1 = x s_0 + a_{n-1}, \quad \ldots, \quad s_n = x s_{n-1} + a_0.$$

Es gilt dann

$$s_n = p(x).$$

Als Gleichungssystem geschrieben, hat man

$$\begin{pmatrix} 1 & & & \\ -x & \ddots & & \\ & \ddots & \ddots & \\ 0 & & -x & 1 \end{pmatrix} \begin{pmatrix} s_0 \\ \vdots \\ s_n \end{pmatrix} = \begin{pmatrix} a_n \\ \vdots \\ a_0 \end{pmatrix}$$

Die Lösung dieses Gleichungssystems mit Residuen-Iteration oder mit Intervall-Einschließung ergibt den Wert $s_n = p(x_n)$.

7.42 Aufgabe: Zeigen Sie

$$\begin{pmatrix} 1 & & & \\ -x & \ddots & & \\ & \ddots & \ddots & \\ 0 & & -x & 1 \end{pmatrix}^{-1} = \begin{pmatrix} 1 & & & \\ x & \ddots & & \\ & \ddots & \ddots & \\ x^n & & x & 1 \end{pmatrix}$$

7.43. Beispiel: Wir berechnen das Polynom

$$p(x) = -945804881 + 1753426039x - 1083557822x^2 + 223200658x^3$$

im Intervall

$$I = [1.61801916, 1.61801917].$$

Mit dem Hornerschema scheitert man an der Auslöschung. Das Beispiel stammt von Rump et. al.

Verwendet wurde folgender Code in EMT. Die Funktion `xevalpoly` enthält eine Residuen-Iteration und die Funktion `ievalpoly` eine Intervall-Einschließung.

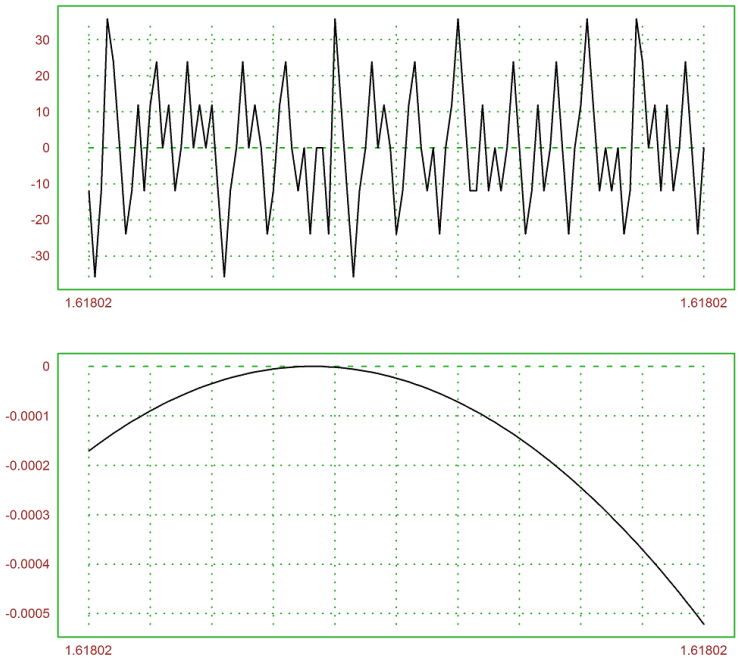

Abbildung 7.2: Hornerschema und Residueniteration für $p(x)$

```
>p=[-945804881,1753426039,-1083557822,223200658];
>t=linspace(1.61801916,1.61801917,100);
>s=evalpoly(t,p);
>figure(2,1);
>figure(1); plot2d(t,s*1e8);
>figure(2); plot2d(t,xevalpoly(t,p,eps=1e-14)*1e8);
>figure(0);
>ievalpoly(1.618019166,p)
 ~-7.1759613078e-013,-7.1759613055e-013~
```

7.8 Orthogonale Transformationen

Verwendet man orthogonale Matrizen $H_1, \ldots, H_k$ zur Umformung von A zu einer rechten oberen Dreiecksmatrix

$$H_k \cdot \ldots \cdot H_1 \cdot A = R$$

so bleibt die Euklidsche Matrixnorm von A und der Inversen von A konstant (siehe Beispiel 7.23). Insbesondere gilt

$$\text{cond}(A) = \text{cond}(R).$$

7.44. Definition: Wenn $\|x\| = 1$ ist, so definieren wir die Householder-Matrix $H(x) \in \mathbb{K}^n$ als

$$H(x) := I_n - 2xx^*.$$

7.45 Aufgabe: Zeigen Sie, dass die Matrizen $H(x)$ orthogonal bzw. unitär sind.

7.46 Satz: *Sei $a \in \mathbb{R}^n$, $x \in \mathbb{R}^n$ und $e_i \in \mathbb{R}^n$ der i-te Einheitsvektor, sowie a kein Vielfaches von e_i. Dann ist*

$$H(x)a = ke_i$$

für ein $k \in \mathbb{R}$ äquivalent zu

$$x = \frac{1}{\|a - ke_i\|}(a - ke_i).$$

und $k = \pm\|a\|$.

Beweis: Wir wollen erreichen, dass für ein $k \in \mathbb{R}$ und ein $x \in \mathbb{R}^n$ mit $\|x\| = 1$ gilt

$$ke_i = H(x)a = a - 2(xx^T)a = a - 2(x^T a)x$$

gilt. Dies ist äquivalent zu

$$(2x^T a)x = a - ke_i,$$

Damit x normiert ist, muss also in der Tat

$$x = \frac{1}{\|a - ke_i\|}(a - ke_i)$$

gesetzt werden, und es muss

$$|k| = \|ke_i\| = \|H(x)a\| = \|a\|$$

sein. Setzt man andererseits k und x in dieser Weise, so gilt

$$\begin{aligned}\|a - ke_i\|^2 &= \|a\|^2 - 2ka_i + |k|^2 \\ &= 2(\|a\|^2 - ka_i) \\ &= 2(a - ke_i)^T a \\ &= 2\|a - ke_i\|(x^T a).\end{aligned}$$

Wegen $a \neq ke_i$ folgt

$$\|a - ke_i\| = 2(x^T a).$$

Also

$$2(x^T a)x = a - ke_i.$$

q.e.d.

Um Auslöschung bei der Berechnung von $a - ke_i$ zu vermeiden, wählt man

$$k = (-\operatorname{sign} a_i)\|a\|.$$

Damit kann man auf die Voraussetzung verzichten, dass a kein Vielfaches von e_i ist.

Der Satz gilt auch im Komplexen, wenn man

$$k = \pm\|a\|e^{-i\alpha}$$

wählt, wobei

$$a_i = re^{i\alpha}, \qquad r \in \mathbb{R},$$

sei. In diesem Fall ist nämlich $ka_i \in \mathbb{R}$.

Die Zerlegung von A in eine orthogonale und eine rechte obere Dreiecksmatrix wird OR-Zerlegung von A genannt. Diese Zerlegung ist eine Verbesserung gegenüber der LR-Zerlegung. Man kann auch damit sehr einfach Gleichungssystem auflösen. Denn es gilt

$$ORx = b \Leftrightarrow Rx = O^*b$$

wegen $O^{-1} = O^*$.

Givens-Rotationen sind eine andere Möglichkeit, eine Matrix A mit Hilfe von orthogonalen Rotationen zu einer rechten oberen Dreiecksmatrix umzuformen.

7.47. Definition: Wir definieren die Matrix $G_{i,j}(\phi) \in \mathbb{R}^{n\times n}$ als die Rotationsmatrix

$$G_{i,j}(\phi) := \begin{pmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & \cos(\phi) & & -\sin(\phi) & & \\ & & & \ddots & & & \\ & & \sin(\phi) & & \cos(\phi) & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{pmatrix},$$

die dadurch ensteht, dass die Drehmatrix

$$D_\phi := \begin{pmatrix} \cos(\phi) & -\sin(\phi) \\ \sin(\phi) & \cos(\phi) \end{pmatrix}$$

in der i-ten und j-ten Zeile und Spalte der Einheitsmatrix I_n eingetragen wird.

$G_{i,j}(\phi)$ bewirkt im $\mathbb{R}^n$ eine Drehung in der Ebene, die durch e_i und e_j aufgespannt wird.

7.48 Satz: *Zu einer Matrix $A \in \mathbb{R}^{n\times k}$ gibt es Givensrotationen*

$$G_1, \ldots G_l$$

so dass

$$G_1 \cdot \ldots \cdot G_l \cdot A = \begin{pmatrix} R \\ 0 \end{pmatrix}$$

mit einer rechten oberen Dreiecksmatrix R in Zeilenstufenform ist.

Die Rotationen sind so beschaffen, dass sie nacheinander die Elemente

$$a_{2,1}, \ldots, a_{n,1}, a_{3,2}, \ldots, a_{n,m-1}$$

zu 0 machen. Dies ist dieselbe Reihenfolge wie beim gewöhnlichen Gauß-Algorithmus. In der Tat ähnelt die Givens-Rotation $G_{i,j}(\phi)$ der Frobeniusmatrix $F_{i,j}(\lambda)$.

Beweis: Der Satz ist nicht schwer zu beweisen, wenn man sich überlegt, welchen Effekt die Rotation

$$\tilde{A} = G_{i,j}(\phi)A$$

hat. Zunächst ändert sich nur die i-te und die j-te Zeile von A. Dort gilt

$$\begin{pmatrix} \tilde{a}_{i,k} \\ \tilde{a}_{j,k} \end{pmatrix} = D_\phi \begin{pmatrix} a_{i,k} \\ a_{j,k} \end{pmatrix} \qquad \text{für alle } k = 1, \ldots, n$$

Um also im ersten Schritt $a_{2,1}$ zu 0 zu machen, muss man lediglich erreichen, dass

$$D_\phi \begin{pmatrix} a_{1,1} \\ a_{2,1} \end{pmatrix} = \begin{pmatrix} \tilde{a}_{1,1} \\ 0 \end{pmatrix}$$

wird. Es kommen dafür zwei Drehwinkel ϕ in Frage, die sich um π unterscheiden. Übrigens gilt dann

$$\tilde{a}_{1,1} = \pm\sqrt{a_{1,1}^2 + a_{2,1}^2}.$$

Dies wiederholt man mit Matrizen $G_{i,1}(\phi_{i,1})$ für $i = 3, \ldots, n$. Man erreicht

$$\tilde{a}_{2,1} = \ldots = \tilde{a}_{n,1} = 0.$$

Das gleiche wird nun mit der zweiten bis zur $n-1$-ten Spalte wiederholt. Die unteren Einträge der vorderen Spalten bleiben dabei gleich 0. **q.e.d.**

Es sieht so aus, als müsste man den Winkel ϕ mit trigonometrischen Funktionen berechnen. Das ist aber nicht der Fall. Um D_ϕ zum Beispiel im ersten Schritt zu finden, benötigt man lediglich

$$c = \cos\phi, \quad s = \sin\phi$$

so dass

$$\begin{aligned} c a_{1,1} - s a_{1,2} &= \tilde{a}_{1,1}, \\ s a_{1,1} + c a_{1,2} &= 0. \end{aligned}$$

Dies ist erfüllt, wenn

$$\begin{aligned} c &= \pm \frac{a_{1,1}}{\sqrt{a_{1,1}^2 + a_{2,1}^2}}, \\ s &= \pm \frac{-a_{1,2}}{\sqrt{a_{1,1}^2 + a_{2,1}^2}} \end{aligned} \tag{7.1}$$

gesetzt wird, wobei das Vorzeichen $\pm$ für c und s natürlich gleich gewählt werden muss. Problematisch ist dabei, wenn $r = 0$ wird. In diesem Fall ist eigentlich nichts zu tun. Die obige Rechnung führt jedoch zum Überlauf, oder zu unsinnigen Werten für c und r. Deswegen muss man $r < \epsilon$ gesondert abfangen.

7.49. Beispiel: Givensrotationen sind in EMT verfügbar, und zwar Schritt für Schritt, sowie als vollständige QR-Zerlegung. Zunächst ein Schritt der Rotation, wobei die rechte Seite b ebenfalls übergeben, und das Ergebnis $Q.b$ zurückgegeben wird.

```
>shortformat;
>A := [1,2,3;4,5,6;7,8,9]
             1             2             3
             4             5             6
             7             8             9
>C,D=givensrot(1,1,2,A,id(3));
>C  // Ergebnis der Rotation
      -4.12311      -5.33578      -6.54846
             0      0.727607       1.45521
             7             8             9
>D.D'  // Rotationsmatrix
             1             0             0
             0             1             0
             0             0             1
>D.A  // Rotation
      -4.12311      -5.33578      -6.54846
             0      0.727607       1.45521
             7             8             9
```

Nun eine QR-Zerlegung für eine reguläre Matrix A. Anschließend wird ein Gleichungssystem gelöst.

```
>A := [1,2,3;4,5,6;7,8,10]
             1             2             3
             4             5             6
             7             8            10
>C,D=givensqr(A,id(3));
>C
      -8.12404      -9.60114      -11.9399
             0     -0.904534      -1.50756
             0             0     -0.408248
>b=sum(A);
>lusolve(C,D.b)
             1
             1
             1
```

Will man die Rotationen abspeichern, um sie später auf andere rechte Seiten b anwenden zu können, so genügt es wegen

$$s = \pm\sqrt{1-c^2}$$

den Wert von c abzuspeichern, wenn man das Vorzeichen in (7.1) so wählt, dass $s > 0$ ist. Als Speicherort kann man die Elemente unterhalb der Diagonalen wählen.

Das folgende Programm löst Gleichungssysteme $Ax = b$ mit regulären Matrizen A mit Hilfe von Givens-Rotationen auf.

```
/**
 * Löst Ax=b mit Hilfe von Givens-Rotationen.
 * @param A Eine quadratische, reelle Matrix
 * @param b Ein Vektor
 * @param x Ergebnisvektor
 */
public static void solveGivens (double[][] A, double[] b, double[] x)
   int n=A.length; // A muss quadratisch sein!
    for (int j=0; j<n-1; j++) // Loop über Spalten
       for (int i=j+1; i<n; i++) // Loop über Zeilen
```

```
        // Berechne c=cos(phi) und s=sin(phi):
         double r=Math.sqrt(A[j][j]*A[j][j]+A[i][j]*A[i][j]);
         if (r<1e-16) continue; // Nichts zu tun!
         double c=A[j][j]/r,s=-A[i][j]/r;
         // Rotiere j-te und i-te Zeile von A
         for (int k=j; k<n; k++)
            double h=A[i][k]*c+A[j][k]*s;
             A[j][k]=A[i][k]*s-A[j][k]*c;
             A[i][k]=h;

         // Rotiere j-te und i-te Zeile von b
         double h=b[i]*c+b[j]*s;
         b[j]=b[i]*s-b[j]*c;
         b[i]=h;


// Berechne x[n-1],...,x[0] aus oberer Dreiecksmatrix:
for (int j=n-1; j>=0; j--)
   x[j]=b[j];
    for (int k=j+1; k<n; k++) x[j]-=A[j][k]*x[k];
    x[j]/=A[j][j];
```

Als Ausgleichsproblem oder Lineare Regression bezeichnet man das Problem

$$\min_x \|Ax - b\|$$

zu einer gegebenen Matrix A und einer gegebenen rechten Seite b zu finden, wobei $Ax = b$ natürlich exakt lösbar sei.

Wir haben schon in Satz 3.87 das Problem mittels der Normalgleichung

$$A^T \cdot A \cdot x = A^T \cdot b$$

gelöst. Diese Lösung ist allerdings nicht sehr stabil, da sie die Kondition verschlechtert. Es wird sich herausstellen, dass sich solche Ausgleichsprobleme sehr stabil mit Hilfe von orthogonalen Transformationen lösen lassen.

7.50 Satz: *Sei $A \in \mathbb{R}^{n\times k}$ als*

$$HA = \begin{pmatrix} R \\ 0 \end{pmatrix}.$$

zerlegt, und

$$Rx = b_1$$

für $x \in \mathbb{R}^k$, wobei

$$Hb = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

mit $b_1 \in \mathbb{R}^k$ und $b_2 \in \mathbb{R}^{n-k}$. Dann ist x das eindeutige Minimum der Funktion

$$\phi(x) = \|Ax - b\|.$$

Beweis: Es gilt für alle $x \in \mathbb{R}^n$

$$\|Ax - b\|^2 = \|HAx - Hb\|^2 = \|Rx - b_1\|^2 + \|b_2\|^2 \geq \|b_2\|^2.$$

Gleichheit gilt nur, wenn $Rx = b_1$ ist. **q.e.d.**

7.51. Beispiel: Die Funktion `fit` von EMT nutzt die Givens-Rotation um das Ausgleichsproblem zu lösen. Das funktioniert auch, wenn A keinen vollen Rang hat. `givensqr` gibt einen zusätzlichen Zeilenvektor zurück, der die Spalten und Zeilen der Zerlegung markiert, die eine eindeutige Lösung garantieren.

```
>shortformat;
>A := [1,2,3;4,5,6;7,8,9]
             1             2             3
             4             5             6
             7             8             9
>b=sum(A)+normal(3,1)*0.1;
>x=fit(A,b)
    0.00901619
       2.99617
             0
>norm(A.x-b)
 0.148316
```

7.52. Definition: Die Matrix

$$A^I := \begin{pmatrix} R^{-1} \\ 0 \end{pmatrix}^T H$$

wird Pseudoinverse von A genannt.

7.53 Aufgabe: Zeigen Sie, dass die Pseudoinverse das Ausgleichsproblem löst.

7.54. Beispiel: Man kann das Ausgleichsproblem zur diskreten Approximation benutzen. Seien $v_1, \ldots, v_m \in C[a,b]$ lineare Funktionen. Gesucht ist eine Funktion

$$v(x) = \sum_{j=1}^{m} \lambda_j v_j(x),$$

die eine Funktion $f \in C[a,b]$ möglichst gut annähert. Dazu minimieren wir

$$\sum_{i=0}^{n} |f(x_i) - v(x_i)|^2$$

mit Punkten

$$a = x_0 < x_1 < \ldots < x_n = b,$$

etwa

$$x_i = a + \frac{i}{n}(b-a), \qquad i = 0, \ldots, n.$$

Das ist dasselbe wie die Minimierung von

$$\|A\lambda - b\|$$

mit

$$A = \begin{pmatrix} v_1(x_0) & \dots & v_m(x_0) \\ \vdots & & \vdots \\ v_1(x_n) & \dots & v_m(x_n) \end{pmatrix}$$

und

$$b = \begin{pmatrix} f(x_0) \\ \vdots \\ f(x_n) \end{pmatrix}.$$

Man nennt diese Art der diskreten Approximation auch lineare Regression.

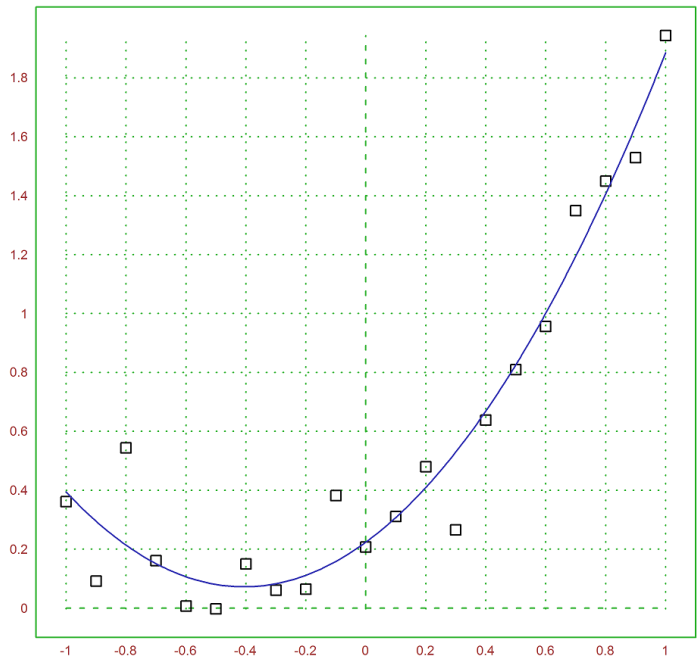

Abbildung 7.3: Fit mit quadratischem Polynom

7.55. Beispiel: Wir verwenden Chebyshev-Polynome als Polynombasis, und berechnen ein quadratisches Regressionspolynom.

```
>function f(x) := x^2+0.8*x+0.2;
>xp := -1:0.1:1; yp := f(xp)+normal(size(xp))*0.1;
>M := cheb(xp',0:2);
>a := fit(M,yp')'
 [ 0.681898073311  0.745402894071  0.457942531622 ]
>function map p(x) := a.cheb(x,(0:2)')
>plot2d(xp,yp,>points);
>plot2d("p",>add,color=blue);
```

Die entsprechenden Funktionen sind in EMT schon implementiert. Bei Verwendung der normalen Polynombasis sind dies `fit` und `evalpoly`, und bei Verwendung der Chebyshev-Polynome

`chebfit` und `chebfitval`.

```
>xp := 0:0.1:1; yp := xp^2+normal(size(xp))*0.1;
>plot2d(xp,yp,>points);
>p := polyfit(xp,yp,2);
>plot2d("evalpoly(x,p)",>add,color=blue);
>a := chebfit(xp,yp,2,0,1)
 [ 0.341494201168  0.487904758828  0.0563458231151 ]
>plot2d("chebval(x,a,0,1)",0,1,color=red,>add);
```

Man kann direkt nachrechnen, dass die Lösung x_0 der Normalgleichung das Ausgleichsproblem löst. Wenn nämlich

$$A^T A x_0 = A^T b$$

ist, so folgt

$$\begin{aligned} \|Ax - b\|^2 &= x^T A^T A x - 2x^T A^T b + b^T b \\ &= x^T A^T A x - 2x^T A^T A x_0 + b^T b \\ &= (x - x_0)^T A^T A (x - x_0) - x_0^T A^T A x_0 + b^T b. \end{aligned}$$

Da $A^T A$ positiv definit ist, wird dieser Ausdruck für $x = x_0$ minimal.

Zur Berechnung muss man die Matrix $A^T A \in \mathbb{R}^{m \times m}$ berechnen, was $O(nm^2)$ Operationen benötigt. Das ist dieselbe Größenordnung, wie die Lösung der Aufgabe mit Givens-Rotationen. Allerdings sind Givens-Rotationen wesentlich stabiler.

7.9 Iterationsverfahren

Wir haben Iterationsverfahren der Form

$$x_{m+1} = (I - RA)x_m + Rb, \qquad m \in \mathbb{N}_0 \tag{7.2}$$

schon bei der Residueniteration und der Intervalleinschließung angetroffen. In diesem Abschnitt geht es darum, Ideen zur Bestimmung der Näherungsinversen R zu gewinnen.

Iterationsverfahren haben den prinzipiellen Vorteil, stabil zu sein. Rundungsfehler verschwinden im Laufe der Iteration wieder, wenn die Iteration gut konvergiert. Darüber hinaus kann die Güte des Ergebnisses allein durch Vergleich von x_{m+1} mit x_m beurteilt werden.

7.56 Satz: *Das obige Iterationsverfahren konvergiert für alle Anfangswerte $x_0 \in \mathbb{R}^n m$, wenn*

$$\|I - RA\| < 1$$

für die zu einer Norm $\|\cdot\|$ auf dem $\mathbb{R}^n$ gehörige Matrixnorm ist. In diesem Fall ist A und R regulär.

Beweis: Wenn die Normabschätzung gilt, so folgt die Konvergenz unmittelbar aus dem Banachschen Fixpunktsatz. In diesem Fall ist RA nach Satz 7.25 regulär, also wegen $\det RA = \det R \det A$ sowohl R als auch A. **q.e.d.**

Die Umkehrung ist nicht richtig. Man muss in diesem Fall den Spektralradius $\sigma(I - RA)$ verwenden, der als Betrag des größten komplexen Eigenwerts von $I - RA$ definiert ist. In der Tat ist dann

$$\sigma(I - RA) < 1$$

äquivalent zur Konvergenz für alle Startwerte.

7.57. Definition: Setzt man die Näherungsinverse

$$R = B^{-1}$$

mit einem $B \sim A$, dann schreibt sich die Iteration als

$$Bx_{m+1} = b - (A - B)x_m.$$

Natürlich macht dies nur Sinn, wenn sich das Gleichungssystem $Bx = y$ leicht nach x auflösen lässt. Wir zerlegen dazu A in die Diagonale D von A, den Anteil E unter der Diagonalen und F über der Diagonalen

$$A = E + D + F.$$

Das Gesamtschrittverfahren, das auch Jacobi-Verfahren genannt wird, verwendet

$$B = D.$$

Natürlich ist in diesem Fall $Bx = y$ leicht zu berechnen. Die Iteration lautet

$$x_{m+1} = D^{-1}(b - (E + F)x_m).$$

Für die i-te Komponente $x_{i,m+1}$ von x_{m+1} gilt also

$$x_{i,m+1} = \frac{1}{a_{i,i}} \left(b_i - \sum_{j \neq i} a_{i,j} x_{j,m} \right), \qquad i = 1, \ldots, m. \tag{7.3}$$

Die Zeilensummennorm von $I - RA$ ist in diesem Fall

$$\|I - D^{-1}A\| = \max_i \sum_{j \neq i} \frac{|a_{i,j}|}{|a_{i,i}|}$$

Sie ist kleiner als 1, wenn die Matrizen stark diagonaldominant sind, wenn also gilt

$$|a_{i,i}| > \sum_{j \neq i} |a_{i,j}|, \qquad \text{für alle } i = 1, \ldots, n.$$

7.58 Satz: *Das Gesamtschrittverfahren konvergiert für stark diagonaldominante Matrizen A.*

Das Gesamtschrittverfahren macht, wie die anderen Iterationsverfahren auch, am meisten Sinn, wenn die Matrix A sehr schwach besetzt ist, wenn also die meisten Element gleich 0 sind. Der Aufwand in jedem Iterationsschritt ist dann proportional zur Anzahl der $a_{i,j} \neq 0$.

7.59. Beispiel: Die $m \times m$-Matrix

$$A = \begin{pmatrix} 4 & 1 & & & \\ 1 & 4 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & 4 & 1 \\ & & & 1 & 4 \end{pmatrix}$$

ist stark diagonaldominant. In diesem Fall ist (mit der Zeilensummennorm)

$$\|I - D^{-1}A\| = \| \begin{pmatrix} 0 & 1/4 & & & \\ 1/4 & 0 & 1/4 & & \\ & \ddots & \ddots & \ddots & \\ & & 1/4 & 0 & 1/4 \\ & & & 1/4 & 0 \end{pmatrix} \| = 1/2.$$

Für $m = 10$ berechnet man in diesem Fall

$$\sigma(I - D^{-1}A) \approx 0.479746$$

Die Größenordung der Konvergenz ergibt sich aus dem Banachschen Fixpunktsatz als $(1/2)^n$. In der Tat findet man die Lösung mit EMT nach etwa 16 Schritten auf 6 Stellen genau.

```
>n=10; D=diag([10,10],0,4); E=diag([10,10],-1,1); F=E';
>x=zeros(10,1); b=ones(10,1);
>for i=1 to 16; x=(b-(E+F).x)/4; end;
>x'
 [ 0.211324083386  0.15470082825  0.169867260149  0.165822551586
 0.166833516909  0.166833516909  0.165822551586  0.169867260149
 0.15470082825  0.211324083386 ]
>(E+D+F).x; %'
 [ 0.999997161794  0.999994656537  0.999992420431  0.9999909834
 0.99999013613  0.99999013613  0.9999909834  0.999992420431
 0.999994656537  0.999997161794 ]
>max(abs(eigenvalues(id(n)-inv(D).(E+D+F))))
 0.479746486807
```

Der Iterationsschritt muss einige Male ausgeführt werden. Das Beispiel ist nicht sehr geeignet, weil es effektiver ist, eine Tridiagonalmatrix mit einen Aufwand von $O(m)$ mit dem Gauß-Verfahren aufzulösen.

7.60. Definition: Für das Einzelschrittverfahren, das auch Gauß-Seidel-Verfahren genannt wird, wählt man

$$B = E + D = A - F.$$

Das Gleichungssystem lautet in diesem Fall

$$(D + E)x_{m+1} = b - Fx_m.$$

Dieses System ist leicht aufzulösen, weil $D + E$ eine untere Dreiecksmatrix ist. Man kann das System als

$$x_{m+1} = D^{-1}(b - (Fx_m + Ex_{m+1}))$$

schreiben. $x_{m+1,i}$ kann nun in der Reihenfolge $i = 1, \ldots, n$ berechnet werden. Ausgeschrieben gilt

$$x_{i,m+1} = \frac{1}{a_{i,i}} \left(b_i - \left(\sum_{j<i} a_{i,j} x_{j,m+1} + \sum_{j>i} a_{i,j} x_{j,m} \right) \right)$$

für $i = 1, \ldots, n$.

Das Verfahren heißt deswegen Einzelschrittverfahren, weil zur Berechnung des Wertes $x_{i,m+1}$ die schon berechneten Werte $x_{k,m+1}$, $k < i$ herangezogen werden.

7.61 Satz: *Das Einzelschrittverfahren konvergiert für stark diagonaldominante Matrizen A.*

Beweis: Es gilt

$$I - RA = I - (E + D)^{-1}A = I - (E + D)^{-1}(E + D + F) = -(E + D)^{-1}F.$$

Wir schätzen die Zeilensummennorm von $(E + D)^{-1}F$ ab. Sei

$$(E + D)^{-1}Fx = y.$$

und die Maximumsnorm $\|x\|_\infty \leq 1$. Man hat wegen $(E + D)y = Fx$

$$y_i = \frac{1}{a_{i,i}} \left(\sum_{j>i} a_{i,j}x_j - \sum_{j<i} a_{i,j}y_j \right).$$

Für stark diagonaldominante Matrizen A gilt also

$$|y_1|, \ldots, |y_{i-1}|, |x_{j+1}|, \ldots, |x_n| \leq 1 \Rightarrow |y_i| < 1.$$

Es folgt induktiv $|y_i| < 1$ für $i = 1, \ldots, n$, also $\|y\|_\infty < 1$. Daraus schließt man $\|(E + D)^{-1}F\|_\infty < 1$. **q.e.d.**

7.62. Beispiel: Für die obige Matrix ist die Iterationsnorm nur halb so groß. Man spart etwa die Hälfte der Schritte. Zur Berechnung des Iterationsschrittes verwenden wir `lusolve`, wobei allerdings $E + F$ eine linke untere Dreiecksmatrix ist. Mit `flipy` vertauschen wir die Zeilen, so dass `lusolve` angewendet werden kann.

```
>n=10; D=diag([n,n],0,4); E=diag([n,n],-1,1); F=E';
>x=zeros(n,1); b=ones(n,1);
>for i=1 to 8; x=xlgs(flipy(D+E),flipy(b-F.x)); end;
>x'
 [ 0.211324342526  0.15470013814  0.169872545172  0.165819321483
 0.166839068286  0.166832707531  0.165825369639  0.169867979667
 0.154701924564  0.211324518859 ]
>(E+D+F).x; %'
 [ 0.999997508246  0.99999744026  1.00000964031  0.999988899388
 1.00000830216  0.99999526805  1.00000216575  0.99999921287
 1.00000019678  1 ]
>max(abs(eigenvalues(id(n)-inv(D).(E+D+F))))
 0.479746486807
```

7.63. Beispiel: Wir lösen ein sehr dünn besetztes Gleichungssystem $Ax = b$ mit EMT. Dünn besetzte Matrizen werden in EMT mit den Funktionen `cpx...` bearbeitet. Es werden nur die Indizes der Elemente gespeichert die nicht 0 sind, so wie deren Wert. Solche Matrizen werden entweder aus vorhandenen Matrizen mit `cpx`, oder mittels einer Liste der Einträge mit `cpxset` erzeugt.

```
>shortestformat;
>A := setdiag(setdiag(4*id(5),1,1),-1,1)
         4         1         0         0         0
         1         4         1         0         0
         0         1         4         1         0
         0         0         1         4         1
         0         0         0         1         4
>cpx(A)
 Compressed 5x5 matrix
     1     1                   4
     1     2                   1
     2     1                   1
     2     2                   4
     2     3                   1
     3     2                   1
     3     3                   4
     3     4                   1
     4     3                   1
     4     4                   4
     4     5                   1
     5     4                   1
     5     5                   4
```

Im folgenden Beispiel ist H ist eine 1000×1000-Matrix, die in der Diagonale immer den Wert 20 hat. Außerdem werden zufällig ausgewählte Elemente gleich 1 gesetzt. Die rechte Seite wird mit Zufallszahlen gefüllt. EMT implementiert das Verfahren in der Funktion `seidelX`.

```
>H := cpxzeros(1000,1000); // leere 1000x1000 Matrix
>k := intrandom(1000,2,1000) | 1;  // Zufalls-Indizes mit 1 als Wert
>shortformat; k[1:5]  // die ersten 5 Indices
           858           745             1
           446           243             1
           136           888             1
           212           407             1
           740           932             1
>H := cpxset(H,k);  // einsetzen in die Matrix
>H := cpxsetdiag(H,0,20);  // Diagonale gleich 20
>b := random(1000,1);  // zufällige rechte Seite
>x := seidelX(H,b);  // Gauß-Seidel Algorithmus
>longestformat; norm(cpxmult(H,x)-b)  // Norm des Residuums
  1.065992463977793e-014
```

Das Gesamtschrittverfahren und das Einzelschrittverfahren konvergieren auch für schwach diagonaldominante Matrizen. Das sind Matrizen mit

$$|a_{i,i}| \geq \sum_{j \neq i} |a_{i,j}|, \qquad \text{für alle } i = 1, \ldots, n,$$

aber „$>$“ für einen Index i_0, mit der zusätzlichen Voraussetzung, dass die Matrizen unzerlegbar sind. Das heißt, es existiert keine Permutationsmatrix P, so dass

$$P^T A P = \begin{pmatrix} A_1 & A_2 \\ 0 & A_3 \end{pmatrix}$$

wird. Wir gehen darauf nicht weiter ein.

7.64. Definition: Im Relaxationsverfahren verwendet man

$$B_\omega = \frac{1}{\omega} D + E.$$

Das Gleichungssystem lautet dann

$$(\frac{1}{\omega}D + E)x_{m+1} = b - (F + D - \frac{1}{\omega}D)x_m.$$

oder, äquivalent,

$$x_{m+1} = (1 - \omega)x_m + \omega D^{-1}(b - (Fx_m + Ex_{m+1})).$$

Man bestimmt also zunächst wie beim Einzelschrittverfahren im i-ten Schritt

$$\tilde{x}_{i,m+1} = \frac{1}{a_{i,i}}\left(b_i - \left(\sum_{j<i} a_{i,j}x_{j,m+1} + \sum_{j>i} a_{i,j}x_{j,m}\right)\right)$$

Danach setzt man

$$x_{i,m+1} = (1 - \omega)x_{i,m} + \omega\tilde{x}_{i,m+1}.$$

Für $\omega = 1$ erhält man das Einzelschrittverfahren. Für $\omega > 1$ geht man stärker in Richtung der neuen Werte $\tilde{x}_{i,m+1}$. Man spricht dann von Überrelaxation. Wenn $0 < \omega < 1$ ist, so werden die Änderungen der x_m gedämpft.

Es gilt der folgende Satz von Ostrowski und Reich, den wir hier nicht beweisen.

7.65 Satz: *Das Relaxationsverfahren konvergiert für alle positiv definiten Matrizen A und* $0 < \omega < 2$.

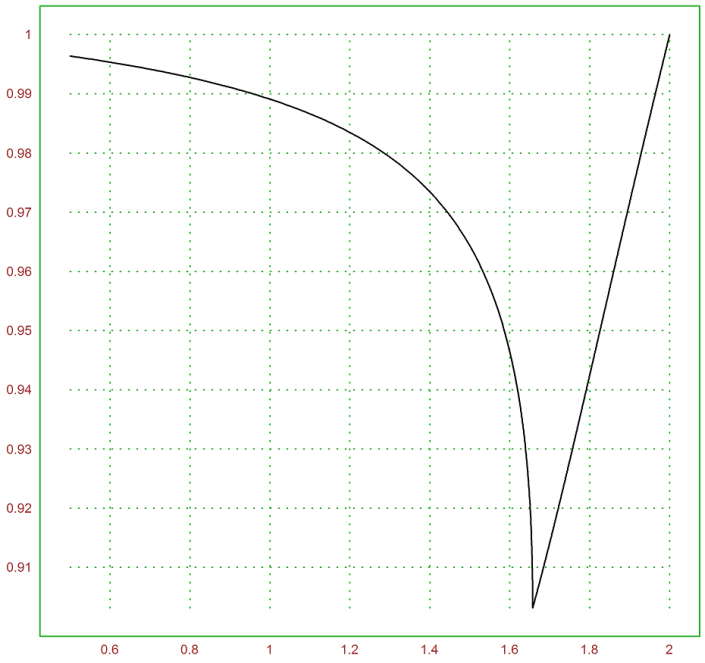

Abbildung 7.4: Norm der Iterationsmatrix für verschiedene ω

7.66. Beispiel: Für eine positiv definite Zufallsmatrix bestimmen wir den optimalen Relaxationsparameter ω, indem wir den Spektralradius von

$$I - \left(\frac{1}{\omega}D + E\right) \cdot A$$

numerisch berechnen. Das Bild zeigt, dass $\omega = 1$ keineswegs optimal ist. Es gibt Strategien, wie man die Relaxationsparameter ω während der Berechnung adaptiv anpasst, so dass das Verfahren möglichst schnell konvergiert.

```
>n := 10;
>A := normal(n,n); A := A.A';
>D := id(n)*A; E := ((1:n)<(1:n)')*A;
>function M(om) := id(n)-inv(D/om+E).A
>function map relaxom(om) := max(abs(eigenvalues(M(om))))
>plot2d("relaxom",0.5,2);
```

7.67 Aufgabe: Sei

$$A_t = \begin{pmatrix} t & 1 & 1 \\ 1 & t & 1 \\ 1 & 1 & t \end{pmatrix}.$$

(a) Für welche $t \in \mathbb{R}$ gilt

$$\sigma(I_3 - RA_t) < 1,$$

wobei R wie im Gesamtschrittverfahren definiert sei.

(b) Testen Sie das Gesamtschrittverfahren für $t = 1, 2, 3, 4$ bei der Lösung von

$$A_t x = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}.$$

mit Startpunkt $x = 0$ auf dem Computer.

(c) Berechnen Sie $\sigma(I_3 - RA_2)$, wobei R wie im Einzelschrittverfahren definiert sei.

(d) Testen Sie mit den Werten aus (b) das Einzelschrittverfahren.

(e) Zeigen Sie durch Berechnung des Spektralradius der Iterationsmatrizen, dass das Einzelschrittverfahren für alle $t > 1$ konvergiert.

7.10 CG-Verfahren

Das CG-Verfahren von Hestenes und Stiefel ist für positiv definite Matrizen geeignet, die schwach besetzt sind.

7.68. Definition: Für eine positiv definite Matrix $M \in \mathbb{R}^{m \times m}$ erhalten wir durch

$$\langle v, w \rangle_M = v^T \cdot M \cdot w$$

ein Skalarprodukt auf dem $\mathbb{R}^m$, mit zugehöriger Norm

$$\|v\|_M = \sqrt{\langle v, v \rangle_M},$$

und wir schreiben

$$v \perp_M w$$

wenn v und w bezüglich dieses Skalarproduktes senkrecht stehen.

Die Idee ist nun für eine positiv definite Matrix A das Problem $Ax = b$ dadurch zu lösen, dass man $\|b - Ax\|_{A^{-1}}$ minimiert. Man beachte, dass A^{-1} ebenfalls positiv definit ist, so dass also die beiden Probleme äquivalent sind.

7.69. Definition: Zu gegebenem $x_0 \in \mathbb{R}^m$ definieren wir den Krylov-Raum

$$K_n = \text{span}\,\{r_0, Ar_0, \ldots, A^{n-1}r_0\}$$

für $n \in \mathbb{N}_0$ und wählen $x_n \in x_0 + K_n$ so dass

$$\|b - Ax_n\|_{A^{-1}} = \min_{x \in K_n} \|b - Ax\|_{A^{-1}}.$$

Wir definieren das Residuum

$$r_k = b - Ax_k.$$

Es ensteht eine Folge von x_k, von denen sich herausstellt, dass man die Glieder durch eine einfache Rekursion berechnen kann, und dass für ein $M \leq m$ die Lösung x_M des Gleichungssystems gefunden wird.

7.70 Satz: *x_n ist dadurch charakterisiert, dass $r_n \perp K_n$ ist. Sei*

$$r_1, \ldots, r_{M-1} \neq 0$$

Dann ist $M \leq m$. Wenn dann $r_M = 0$ ist, so ist x_M Lösung des Gleichungssystems. Das Verfahren findet also nach spätestens M Schritten eine Lösung. Die Vektoren

$$r_0, \ldots, r_{n-1}$$

bilden für $n \leq M$ eine Orthogonalbasis von K_n.

Beweis: Es gilt

$$f(x) = \|b - Ax\|^2_{A^{-1}} = b^T A^{-1} b - 2b^T x + x^T Ax.$$

also

$$\text{grad}\ f(x) = 2(b - Ax)^T.$$

Aufgrund des Lagrange-Kriteriums für Extrema unter einer Nebenbedingung wird diese Funktion wird genau dann in $x_n \in K_n$ unter alle $x \in K_n$ minimal, wenn

$$2r_n = \text{grad}\ f(x_n) \perp K_n.$$

Sei $r_n \neq 0$ für $n < M$. Wir haben aufgrund der Konstruktion

$$r_n = b - Ax_n \in b - Ax_0 - AK_n = r_0 - AK_n \subseteq K_{n+1},$$

und es gilt jeweils $r_n \perp K_n$. Also

$$K_{n+1} = \text{span}\,\{r_0, \ldots, r_n\}.$$

und diese Vektoren bilden, da sie ungleich 0 sind, eine Orthogonalbasis von K_{n+1} für $n < M$. Es folgt sofort $M \leq m$, und damit die Behauptung. **q.e.d.**

7.71 Satz: *Die Iterierten x_n und die Residuen r_n können aus den Anfangswerten*

$$p_0 = r_0 = b - Ax_0$$

rekursiv mittels

$$\begin{aligned} x_{n+1} &= x_n + a_n p_n, \qquad a_n = \frac{r_n^T r_n}{p_n^T A p_n} \\ r_{n+1} &= r_n - a_n A p_n \\ p_{n+1} &= r_{n+1} + b_n p_n, \qquad b_n = \frac{r_{n+1}^T r_{n+1}}{r_n^T r_n} \end{aligned}$$

berechnet werden.

Beweis: Wir nehmen $r_n \neq 0$ für $n < M$ an, sowie $r_M = 0$. Dann setzen wir

$$d_0 = x_1 - x_0, \quad \ldots, \quad d_{M-1} = x_M - x_{M-1}.$$

Da $r_0, \ldots, r_{M-1}$ orthogonal sind, sind $r_0, \ldots, r_M$ paarweise verschieden, und daher

$$Ad_n = Ax_{n+1} - Ax_n = r_n - r_{n+1} \neq 0.$$

Da wir wissen, dass $r_0, \ldots, r_{n-1}$ eine Orthogonalbasis von K_n bilden, erhalten wir

$$d_n \perp_A K_n$$

für alle $n < M$. Andererseits gilt offenbar $d_n \in K_{n+1}$. Also ist

$$d_0, \ldots, d_n$$

eine Orthonormalbasis von K_{n+1} bezüglich des Skalarprodukts $\langle v, w \rangle_A$. Wir entwickeln $r_{n+1} \in K_{n+2}$ bezüglich dieses Skalarprdukts und erhalten

$$r_{n+1} = \beta_n d_{n+1} - \gamma_n d_n.$$

Die anderen Koeffizienten müssen 0 sein, da

$$\langle r_{n+1}, d_l \rangle_A = \langle r_{n+1}, Ad_l \rangle = 0$$

wegen $Ad_l \in K_{l+1}$ für alle $l \leq n-1$. Außerdem gilt $\beta_n \neq 0$, wegen $r_{n+1} \notin K_{n+1}$. Wir setzen nun

$$p_n = \beta_n d_n.$$

für alle $n < M$ und erhalten in der Tat eine Darstellung

$$\begin{aligned} x_{n+1} &= x_n + a_n p_n, \\ r_{n+1} &= r_n - a_n A p_n, \\ p_{n+1} &= r_{n+1} + b_n p_n. \end{aligned}$$

Es bleibt zu zeigen, dass die Koeffizienten a_n, b_n wie im Satz angegeben sind. Aus der zweiten Beziehung erhalten wir wegen der Orthogonalität der r_l sofort

$$a_n = \frac{r_n^T r_n}{p_n^T A r_n}$$

Wegen $r_n - p_n \in K_n$ gilt aber $p_n \perp_A r_n - p_n$ und daher

$$p_n^T A r_n = p_n^T A p_n.$$

Durch Multiplikation der letzten Zeile mir r_n erhalten wir

$$b_k = \frac{p_{n+1}^T r_n}{p_n^T r_n}$$

Es gilt

$$p_n^T r_n - r_n^T r_n = (p_n - r_n)^T r_n = b_{n-1}(p_{n-1}^T r_n) = 0$$

also

$$p_n^T r_n = r_n^T r_n$$

wegen $p_{n-1} \in K_n$. Ebenso wegen $r_{n+1} - r_n = a_n A p_n$

$$p_{n+1}^T r_n = p_{n+1}^T r_{n+1} = r_{n+1}^T r_{n+1}.$$

Es folgt die Behauptung. **q.e.d.**

Zur Fehleranalyse beachte man, dass

$$x_{n+1} = x_0 + \sum_{k=0}^{n} \alpha_k A^k r_0$$

für optimal gewählte α_k gilt. Also

$$r_{n+1} = b - A x_{n+1} = r_0 + \sum_{k=0}^{n} \alpha_k A^{k+1} r_0 = p_{n+1}(A) r_0$$

mit einem Polynom $p_{n+1} \in \mathcal{P}_{n+1}$ mit

$$p_{n+1}(0) = 1,$$

das so gewählt wird, dass $\|r_{n+1}\|_{A^{-1}}$ minimal wird. Wir können daher durch Angabe eines "guten" Polynoms $p_n \in \mathcal{P}_n$ mit $p_n(0) = 1$ eine obere Abschätzung des Fehlers gewinnen. Sei

$$r_0 = \sum_{k=1}^{m} \beta_k v_k$$

wobei v_k eine Orthonormalbasis des $\mathbb{R}^m$ bezüglich des Skalarproduktes $\langle v, w \rangle_{A^{-1}}$ mit Eigenwerten

$$0 < \lambda_1 \leq \ldots \leq \lambda_m$$

sei. Dann haben wir also

$$r_n = p_n(A) r_0 = \sum_{k=1}^{n} \beta_k p(\lambda_k) v_k$$

und folglich

$$\|r_n\|^2_{A^{-1}} = \sum_{k=1}^{n} \beta_k^2 p(\lambda_k)^2 \leq \|p\|^2_{[\lambda_1,\lambda_m]} \|r_0\|^2_{A^{-1}}.$$

Ein gutes Polynom kann man also finden, indem man $\|p\|_I$ auf $I = [\lambda_1, \lambda_m]$ unter der Nebenbedingung $p(0) = 1$ minimiert. Die Lösung dieses Problem sind transformierte Chebyshev-Polynome. Wir wählen

$$\tilde{p}_n(x) = T_n\left(\frac{(\lambda_m + \lambda_1)/2 - x}{(\lambda_m - \lambda_1)/2}\right)$$

und dann $p_n(x) = \tilde{p}_n(x)/\tilde{p}_n(0)$. Man hat dann

$$\|p_n\|_{[\lambda_1,\lambda_m]} \leq \frac{2}{\rho^n + \rho^{-n}}$$

mit

$$\rho = \kappa + \sqrt{\kappa^2 - 1}, \quad \kappa = \frac{\lambda_1 + \lambda_m}{\lambda_m - \lambda_1} > 1$$

gemäß Aufgabe 3.69. Diese obere Abschätzung entspricht einer geometrischen Konvergenz. Natürlich annulliert das Minimalpolynom von A den Fehler für ein $n \leq m$.

7.72 Aufgabe: Zeigen Sie, dass A bezüglich des Skalarproduktes $\langle v, w\rangle_A$ selbstadjungiert ist, und dass es also tatsächlich eine Orthonormalbasis bezüglich dieses Skalarproduktes aus Egenwerten gibt.

7.73 Aufgabe: Zeigen Sie, dass das Verfahren nach einem Schritt endet, wenn r_0 ein Eigenwert von A ist. Zeigen Sie weiter, dass das Verfahren nach k Schritten endet, wenn r_0 in einem k-dimensionalen Raum liegt, der eine Basis aus Eigenwerten hat.

7.74. Beispiel: Das CG-Verfahren ist in EMT für normale und für schwach besetzte Matrizen implementiert. Die Funktion `cgX` führt das Verfahren für schwach besetzte, positiv definite Matrizen durch. Für allgemeine Matrizen sollte man statt dessen `cpxfit` verwenden, was die Normalgleichung

$$A^T \cdot A \cdot x = A^T \cdot b$$

löst. Man beachte, dass dann $A^T \cdot A$ positiv semi-definit ist. Falls A vollen Rang hat, so ist es sogar positiv-definit.

Im Beispiel erzeugen wir wieder dieselbe unsymmetrische Zufallsmatrix wie im Beispiel 63, und lösen sie mit `cpxfit`

```
>H := cpxzeros(1000,1000); // leere 1000x1000 Matrix
>k := intrandom(1000,2,1000) | 1;  // Zufalls-Indizes mit 1 als Wert
>H := cpxset(H,k);  // einsetzen in die Matrix
>H := cpxsetdiag(H,0,20);  // Diagonale gleich 20
>b := random(1000,1);  // zufällige rechte Seite
>x := cpxfit(H,b);  // CG-Verfahren für H'.H.x=H'.b
>longestformat; norm(cpxmult(H,x)-b)  // Norm des Residuums
  1.571766870343371e-014
```

Man beachte, dass das Programm wegen der Iteration $H^T \cdot H$ nicht berechnen muss, sondern die Multiplikation jedesmal explizit ausführen kann.

Im folgenden den Code für einen Schritt des Verfahrens. Man beachte, dass im Beispiel aufgrund von Rechenfehlern mehr Schritte notwendig werden.

```
>function cgstep (x,r,p) ...
$  global A;
$  a=(r'.r)/(p'.A.p);
$  xn=x+a*p;
$  rn=r-a*A.p;
$  b=(rn'.rn)/(r'.r);
$  pn=rn+b*p;
$  return xn,rn,pn
$endfunction
>n=20;
>A=random(n,n); A=A.A';
>b=random(n,1);
>x=random(n,1);
>r=b-A.x;
>p=r;
>count=0; repeat x,r,p=cgstep(x,r,p); until all(r~=0); count=count+1; end;
>count,
 28
>norm(A.x-b)
 0
```

Wenn A nicht regulär ist, aber noch positiv semi-definit, so scheitert das Verfahren nur, falls zufällig $r_0 \in \operatorname{Kern} A$ liegt. Denn dann haben wir

$$x_1 = x_0.$$

Falls dies nicht der Fall ist, so Ar_0 im zum Kern orthogonalen Raum, der von Eigenvektoren zu Eigenwerten ungleich 0 aufgespannt ist. Das Verfahren liefert dann nach endlich vielen Schritten einen besten Fit. Lässt man es darüber hinaus laufen, so wird allerdings der Nenner in a_k gleich 0.

```
>n=3;
>A &= [2,1,1;1,2,1;1,1,x]; &solve(det(A)); A := (&A with %)();
>fracprint(A)
         2         1         1
         1         2         1
         1         1       2/3
>b=random(n,1);
>x=random(n,1);
>r=b-A.x;
>p=r;
>loop 1 to 2; x,r,p=cgstep(x,r,p); until all(r~=0); end;
>norm(A.x-b)
 0.0578903465777
>x=fit(A,b); norm(A.x-b)  // bester Fit mit Givens-Rotationen
 0.0543909760898
```

Kapitel 8

Gewöhnliche Differentialgleichungen

8.1 Einführung

Wir wiederholen kurz die wesentlichen Fakten aus der Theorie der Differentialgleichungen. Eine gewöhnliche (nicht partielle) Differentialgleichung hat die Gestalt

$$y'(x) = f(x, y(x)).$$

Dabei ist $f : U \to \mathbb{R}^n$ eine stetige Abbildung, die auf einer offenen Teilmenge

$$U \subseteq \mathbb{R} \times \mathbb{R}^n$$

definiert ist. Gesucht ist eine differenzierbare Funktion $y : I \to \mathbb{R}^n$, die die Differentialgleichung mit gewissen Anfangswerten

$$y(x_0) = y_0$$

erfüllt. I ist ein Intervall und natürlich ist $x_0 \in I$. Außerdem verlangen wir für die Lösung natürlich

$$(t, y(t)) \in U \qquad \text{für alle } t \in I.$$

Eine Differentialgleichung mit einem Anfangswert heißt auch Anfangswertproblem. Wir wollen solche Probleme numerisch lösen.

8.1. Beispiel: Wir betrachten die Differentialgleichung

$$y'(x) = f(x, y(x)) = 2xy(x), \qquad \text{für alle } x \in \mathbb{R}.$$

In diesem Fall ist $U = \mathbb{R} \times \mathbb{R}$ und $n = 1$. Man kann diese Gleichung durch Trennung der Variablen lösen. Aus dem folgenden Satz wird folgen, dass die Gleichung mit gegebenen Anfangsbedingungen $y(x_0) = y_0$ lokal eindeutig lösbar ist. Wenn $y_0 = 0$ ist, so ist diese Lösung die Nullfunktion $y = 0$. Wenn nicht, so kann man schreiben

$$\frac{y'(x)}{y(x)} = x$$

Wir integrieren auf beiden Seiten unbestimmt (das heißt, wir nehmen auf beiden Seiten die Stammfunktionen) und erhalten

$$\ln|y(x)| = \int \frac{y'(x)}{y(x)}\,dx = \int 2x\,dx = x^2 + c.$$

Also

$$y(x) = \pm e^{x^2+c} = Ce^{x^2}$$

mit einer beliebigen Konstanten $C \in \mathbb{R}$. Diese Lösungen umfassen auch den Fall $y = 0$. C ist so zu bestimmen, dass die Anfangswerte $y(x_0) = y_0$ erfüllt sind. Das ist immer eindeutig möglich.

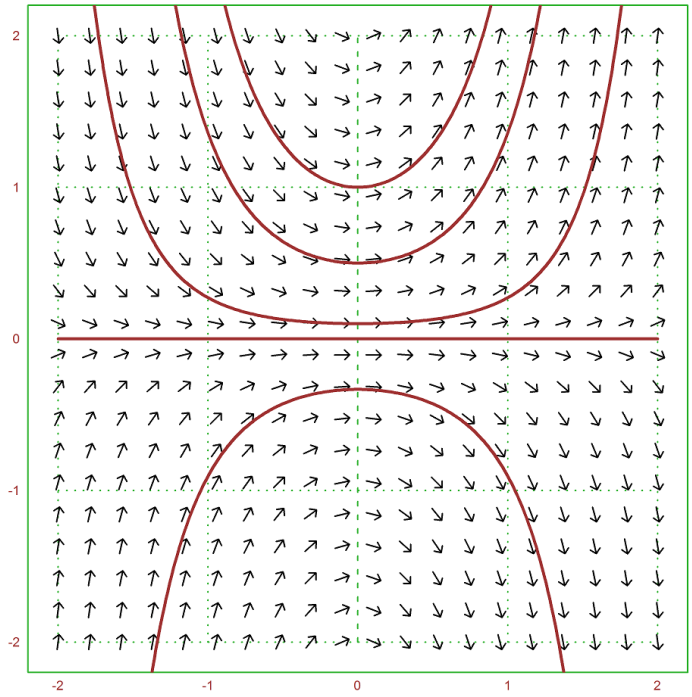

Abbildung 8.1: Differentialgleichung $y' = 2xy$

8.2. Beispiel: Man kann Differentialgleichungen so deuten, dass in jedem Punkt $(x, y) \in \mathbb{R}^2$ eine Steigung $f(x, y)$ vorgegeben ist. Zeichnet man diese Steigungen in der Ebene ein, so kann man die Lösungen der Differentialgleichung schon erkennen. EMT kann solche Vektorfelder zeichnen.

```
>vectorfield("2*x*y",-2,2,-2,2);
>plot2d("exp(x^2)*[1,0.5,0,-1/3,1/10]'",color=red,>add,thickness=2);
```

8.3 Aufgabe: Lösen Sie die Differentialgleichung

$$y'(x) = -y(x)^2$$

durch Trennung der Variablen. Beachten Sie den Sonderfall $y = 0$. Wie weit lässt sich die Lösung fortsetzen?

8.4 Aufgabe: Zeigen Sie, dass die Differentialgleichung

$$y'(x) = \sqrt{|2y(x)|}$$

Lösungen der Form

$$y(x) = \begin{cases} \frac{-(x-D)^2}{2}, & x < D, \\ 0, & D \le x \le C, \\ \frac{(x-C)^2}{2}, & x > C, \end{cases}$$

mit Konstanten $-\infty \le D \le C \le \infty$ hat. Sie ist also ausgehend von Anfangswerten $y(x_0) = 0$ nicht einmal lokal eindeutig lösbar.

8.5. Beispiel: Die Differentialgleichung zweiter Ordnung

$$u''(x) = x^2 u(x)$$

lässt sich in eine gewöhnliche Differentialgleichung im $\mathbb{R}^2$ umschreiben, indem wir $y_1(x) = u(x)$ und $y_2(x) = u'(x)$ setzen. Dann gilt nämlich

$$\begin{pmatrix} y_1'(x) \\ y_2'(x) \end{pmatrix} = \begin{pmatrix} y_2(x) \\ x^2 y_1(x) \end{pmatrix}$$

Dies entspricht dem Grundtypus

$$y'(x) = f(x, y(x))$$

wobei allerdings y eine Abbildung in den $\mathbb{R}^2$ ist. Aus diesem Grunde ist es nützlich, auch Differentialgleichungen mit $n > 1$ zu betrachten.

8.6. Beispiel: Die Schwingungsgleichung

$$u''(x) = -u(x)$$

hat die Lösungen

$$u(x) = \alpha \sin(x) + \beta \cos(x).$$

Sie ist eine homogene lineare Differentialgleichung mit konstanten Koeffizienten, also vom Typ

$$u^{(n+1)}(x) = a_n u^{(n)}(x) + \ldots + a_1 u'(x) + u(x).$$

Die komplexen Lösungen dieser Differentialgleichung können durch den Ansatz $u(x) = e^{\lambda x}$ ermittelt werden. Wir gehen auf diese Theorie hier nicht weiter ein. Äquivalent zur Schwingungsgleichung ist das System

$$\begin{pmatrix} y_1'(x) \\ y_2'(x) \end{pmatrix} = \begin{pmatrix} y_2(x) \\ -y_1(x) \end{pmatrix} = f(y)$$

mit der Lösung $u = y_1$. Wir können dafür ein Vektorfeld in der Ebene zeichnen, sowie typische Lösungen der Form

$$y(x) = r\,(\sin(x), \cos(x))\,.$$

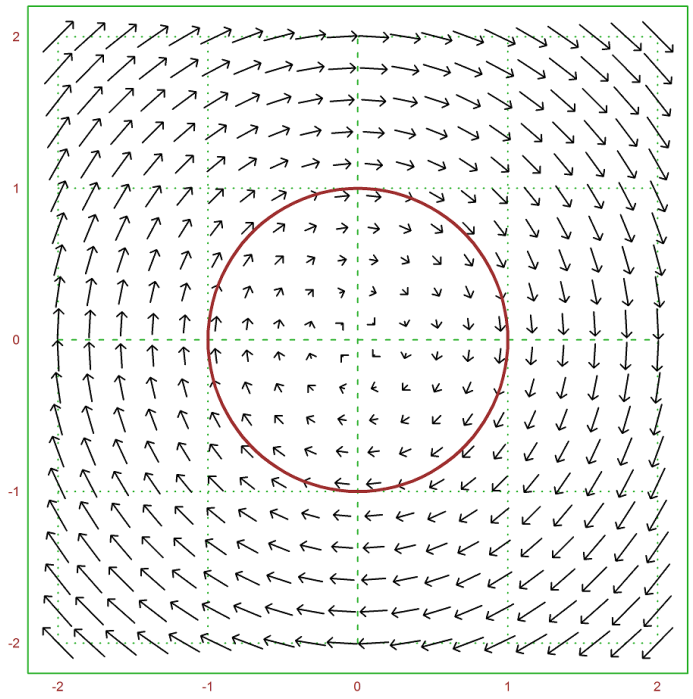

Abbildung 8.2: Vektorfeld für die Schwingungsgleichung

```
>vectorfield2("y","-x",-2,2,-2,2);
>plot2d("sin(x)","cos(x)",xmin=0,xmax=2pi,>add,color=red,thickness=2);
```

8.7 Aufgabe: Zeigen Sie, dass dann $y_1(x+c)$ für geeignet gewähltes c tatsächlich jede gewünschte Form

$$u(x) = y_1(x+c) = \alpha \sin(x) + \beta \cos(x)$$

annimmt, indem Sie $y_1(x) = \mathrm{Re}(re^{ix})$ schreiben.

Der wichtigste Existenzsatz für Lösungen von Differentialgleichungen ist der Satz von Picard-Lindelöff. Dieser Satz macht die Voraussetzung, dass $f(x,y)$ einer lokalen Lipschitzbedingung in y genügt. Das heißt, es gibt zu jedem Punkt $(x,y) \in U$ eine Umgebung V von (x,y) und ein $L > 0$, so dass

$$\|f(x,y_1) - f(x,y_2)\| \le L\|y_1 - y_2\|$$

für alle $(x,y_1) \in V$ und $(x,y_2) \in V$ gilt. f genügt einer globalen Lipschitzbedingung auf U, wenn diese Gleichung auf ganz U gilt mit dem gleichen $L > 0$ gilt.

8.8 Satz: *Falls eine gewöhnliche Differentialgleichung*

$$y'(x) = f(x, y(x))$$

einer lokalen Lipschitzbedingung in U genügt, so existiert zu jeder Anfangsbedingung $(x_0, y_0) \in U$ eine eindeutige Lösung. Der Weg $x \mapsto (x, y(x))$ lässt sich bis zum Rand von U fortsetzen.

Falls $U = I \times \mathbb{R}^n$ ist für ein offenes Intervall $I \subset \mathbb{R}$, und f einer globalen Lipschitzbedingung auf U genügt, so existiert eine eindeutige Lösung der Differentialgleichung $y : I \to \mathbb{R}$ zu den gegebenen Anfangswerten $y(x_0) = y_0$, $x_0 \in I$.

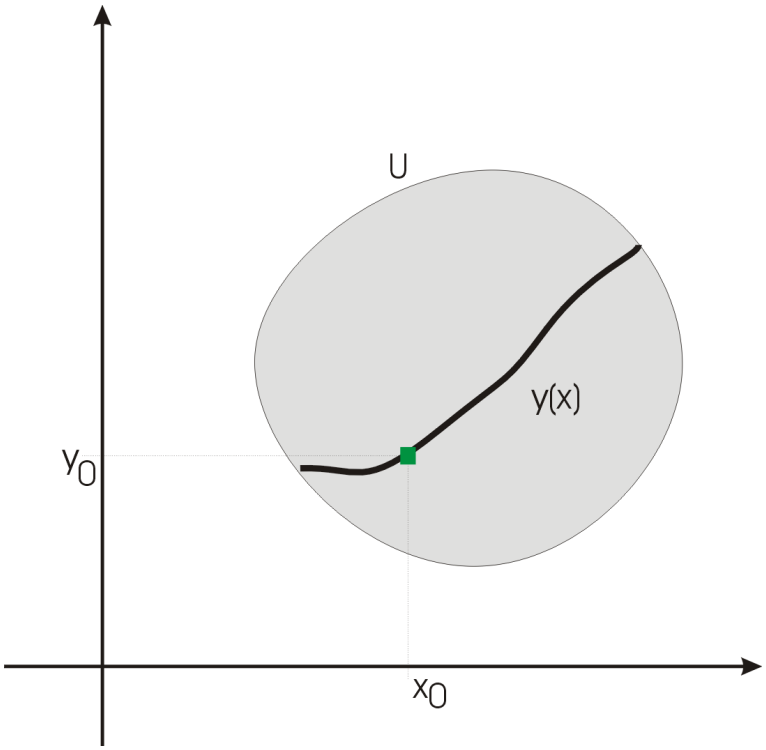

Abbildung 8.3: Fortsetzung der Lösung bis zum Rand von U

Wenn f nur für $t \geq x_0$ einer Lipschitzbedingung genügt, so folgt die Existenz einer Lösung nur für $t \geq x_0$. Man kann dazu den Existenzsatz anwenden, indem man

$$f(x, y) = f(x_0, y)$$

für $x \leq x_0$ setzt.

8.9. Beispiel: (1) Die Lösung $y(x) = e^{x^2}$ des Anfangswertproblems

$$y'(x) = 2xy(x), \quad y(0) = 1$$

ist eindeutig bestimmt und auf ganz $\mathbb{R}$ fortsetzbar, weil die Funktion $f(x, y) = 2xy$ offenbar auf jeder Menge $(-r, r) \times \mathbb{R}$ einer globalen Lipschitzbedingung genügt. Es gilt nämlich

$$|2xy_1 - 2xy_2| = 2|x||y_1 - y_2| \leq 2r|y_1 - y_2|.$$

(2) Wählen wir $U = \mathbb{R} \times (0, \infty)$, so lässt sich die Lösung $y(x) = 1/(x + 1)$ des Anfangswertproblems

$$y'(x) = -y(x)^2, \quad y(0) = 1$$

nicht für alle $x \in \mathbb{R}$ fortsetzen, aber die Kurve $(x, y(x))$ liegt ganz in U. Die Funktion $f(x, y) = -y^2$ erfüllt keine globale Lipschitzbedingung auf ganz U, denn

$$f(x, y_1) - f(x, y_2) = \frac{\partial}{\partial y} f(x, \xi)(y_2 - y_1) = -2\xi(y_2 - y_1)$$

mit $\xi \in (y_1, y_2)$. $|y|$ ist aber in U nicht beschränkt.

(3) Mit dem gleichen U lässt sich die Lösung $y(x) = x^2$ des Anfangswertproblems

$$y'(x) = 2\sqrt{y(x)}, \quad y(1) = 1$$

für $x \in (0, \infty)$ eindeutig fortsetzen, solange $y(x) > 0$ ist. Die Kurve (x, x^2) erreicht den Rand von U in $(0, 0)$.

8.10. Beispiel: Mit Hilfe von Maxima können exakte Lösungen von vielen Differentialgleichungen in EMT gefunden werden. Dabei wird eine Differentialgleichung mit Differentialoperatoren geschrieben, deren Auswertung durch einen Apostroph verhindert wird. Zur Lösung verwenden wir hier `ode2`.

```
>eq &= 'diff(y,x)=y/x^2
```

$$\frac{dy}{dx} = \frac{y}{x^2}$$

```
>sol &= ode2(eq,y,x)
```

$$y = \%c\, E^{-1/x}$$

Für die automatische Lösung von Anfangswertproblem steht `ic1` zur Verfügung. Bei bekannter allgemeiner Lösung wird dadurch die Konstante ermittelt.

```
>&ic1(sol,x=1,y=-1)
```

$$y = -E^{1-1/x}$$

```
>function ysol(x) &= y with ic1(sol,x=1,y=-1)
```

$$-E^{1-1/x}$$

```
>plot2d(ysol,0.5,2);
```

8.11. Beispiel: Als Beispiel für eine Differentialgleichung zweiter Ordnung lösen wir etwa das Anfangsproblem

$$y''(x) + y'(x) = x^3, \quad y(0) = 0, \quad y'(0) = 1$$

```
>sol &= ode2('diff(y,x,2)+y=x^3,y,x)
```

$$y = \%k1 \sin(x) + \%k2 \cos(x) + x^3 - 6\,x$$

```
>function ysol(x) &= y with ic2(sol,x=0,y=0,'diff(y,x)=1)
```

$$7 \sin(x) + x^3 - 6\,x$$

8.2 Streckenzugverfahren

Es ist naheliegend, einen Streckenzug zu konstruieren, der die Differentialgleichung auf jeder Teilstrecke in einem Punkt erfüllt. Dies führt zum Streckenzugverfahren, das nach Leonhard Euler benannt ist.

8.12. Definition: Um das Anfangswertproblem

$$y'(x) = f(x, y(x)), \quad y(x_0) = y_0$$

zu lösen, wählen wir Punkte

$$x_0 < x_1 < \ldots < x_m$$

und setzen

$$y_{k+1} = y_k + f(x_k, y_k)(x_{k+1} - x_k), \qquad k = 1, \ldots, m.$$

Verbindet man die Punkte (x_k, y_k) mit einem Streckenzug s, so hat dieser Streckenzug die Eigenschaft

$$s'(x_k) = \frac{s(x_{k+1}) - s(x_k)}{x_{k+1} - x_k} = \frac{y_{k+1} - y_k}{x_{k+1} - x_k} = f(x_k, y_k),$$

wobei die Ableitung rechtsseitig gemeint ist. Analog kann man die Lösung auch von x_0 aus nach links berechnen.

8.13 Satz: *Sei*

$$y : [a, b] \to \mathbb{R}^n$$

eine Lösung des Anfangswertproblems

$$y'(x) = f(x, y(x)), \quad y(a) = y_0.$$

f genüge einer globalen Lipschitzkonstante in

$$M_\eta = \{(t, s) : a \le t \le b \text{ und } \|y(t) - s\| < \eta\}$$

für ein $\eta > 0$. Dann liegen die Streckenzüge s_m des Eulerschen Streckenzugverfahrens mit den Stützpunkten

$$x_0 = a, \quad x_m = b$$

für m groß genug in M_η, und konvergieren gleichmäßig auf $[a, b]$ gegen y, wenn die Feinheit der Unterteilungen gegen 0 geht.

Der Beweis dieses Satzes wird später in einem allgemeineren Rahmen nachgeliefert.

8.14 Aufgabe: Zeigen Sie, dass die Voraussetzung, dass f in einem Streifen M_η eine Lipschitzkonstante erfüllt, ist für ein genügend kleines $\eta > 0$ immer erfüllt ist, wenn $f(x, y)$ nach y stetig partiell differenzierbar ist.

8.15. Beispiel: Das folgende EMT-Programm testet das Eulersche Streckenzug-Verfahren im Intervall $[0, 1]$ für die Differentialgleichung $y' = y$. Wie man in Abbildung 8.4 sieht, bleibt der Streckenzug immer unterhalb der korrekten Lösung.

```
>function euler (f,x,y0)...
$  y=zeros(size(x)); y[1]=y0;
$  for i=1 to cols(x)-1;
$    y[i+1]=y[i]+f(x[i],y[i])*(x[i+1]-x[i]);
$  end;
$  return y;
$endfunction
>function f(x,y) &= y
>plot2d("exp(x)",0,1);
>x=0:0.2:1; y=euler("f",x,1);
>plot2d(x,y,>points,>add,color=red);
>plot2d(x,y,>add,color=red);
>x=0:0.1:1; y=euler("f",x,1);
>plot2d(x,y,>points,>add,color=blue);
>plot2d(x,y,>add,color=blue);
```

Um die Ordnung p des Verfahrens zu ermitteln, nehmen wir an, dass für den Fehler

$$\epsilon_n = |y_n - y(b)| = c\, h^p$$

mit der Schrittweite $h = (b - a)/n$ gelte. Dann erhält man

$$p = \log_2 \left(\frac{\epsilon_n}{\epsilon_{2n}} \right).$$

In der Tat ergibt sich für das Eulersche Verfahren $p = 1$, indem wir in EMT den Fehler für ein Problem mit bekannter Lösung $y = e^{-x^2}$ vergleichen.

```
>function f(x,y) := -2*x*y
>err1000=euler("f",linspace(0,1,1000),1)[-1]-1/E
 0.00012276966149
>err2000=euler("f",linspace(0,1,2000),1)[-1]-1/E
 6.13490208505e-005
>log(err1000/err2000)/log(2)
 1.00084186649
```

8.16 Aufgabe: (a) Wenden Sie das Eulersche Streckenzugverfahren auf das Anfangswertproblem

$$y'(x) = y(x), \quad y(0) = 1$$

mit äquidistanten Stützstellen

$$0 = x_0 < x_1 < \ldots < x_n = 1$$

an, und berechnen Sie den Wert des Streckenzugs s_n in 1 exakt.

(b) Zeigen Sie

$$|s_n(1) - e| \leq \frac{C}{n}$$

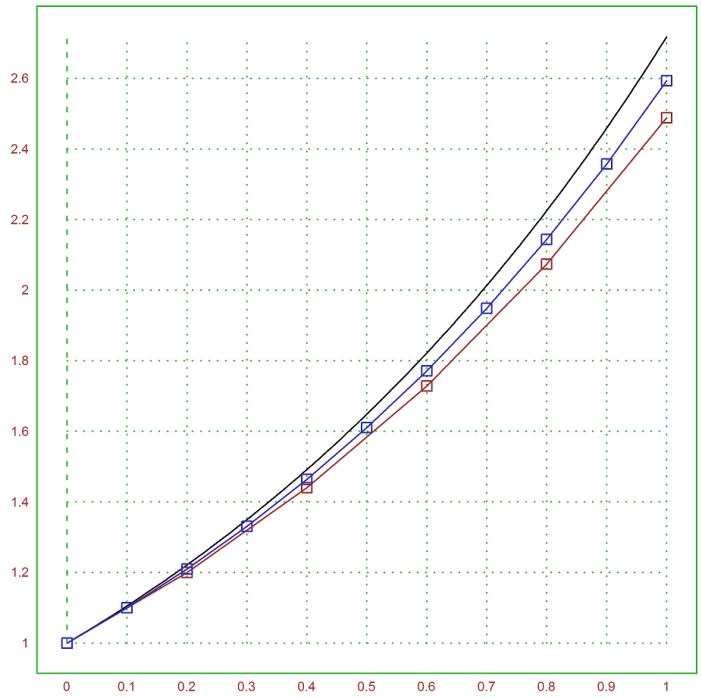

Abbildung 8.4: Eulersche Streckenzüge

für eine Konstante $C > 0$.

8.17 Aufgabe: (a) Schreiben Sie das Anfangswertproblem zweiter Ordnung

$$y''(x) = y(x), \quad y(0) = 1, \quad y'(0) = 0$$

ein gewöhnliches Anfangswertproblem um. Zeigen Sie, dass dieses System einer globalen Lipschitzbedingung genügt. Verwenden Sie das obige Programm, um den Wert $y(1)$ mit dem Eulerschen Streckenzugverfahren und äquidistanten Schrittweiten

$$0.1, \quad 0.01, \quad 0.001$$

zu berechnen. Berechnen Sie den Wert $s_n(1)$ des Eulerschen Streckenzugverfahrens bei Schrittweite $1/n$ exakt. Verallgemeinern Sie diese Ergebnisse auf Differentialgleichungen vom Typ $y' = Ay$. Berechnen Sie den speziellen Fall

$$y'' = -y, \quad y(0) = 1, \quad y'(0) = 0$$

exakt.

8.3 Verwendung höherer Ableitungen

Der Eulersche Streckenzug ist ein Spline 1. Ordnung, der die Differentialgleichung in diskreten Punkten löst. Man kann auch Splines höherer Ordnung verwenden. In jedem Fall soll gelten

$$s(x_k) = y_k, \quad s'(x_k) = f(x_k, y_k),$$

wobei die Ableitung wieder rechtsseitig gemeint ist, wenn $x_{k+1} > x_k$ ist. Um den Spline festzulegen, benötigt man höhere Ableitungen im Punkt x_k. Dazu stellen wir uns vor, $\tilde{y}$ sei die korrekte Lösung des Anfangswertproblems

$$\tilde{y}'(x) = f(x, \tilde{y}(x)), \quad \tilde{y}(x_k) = y_k.$$

Dann gilt natürlich

$$\tilde{y}'(x_k) = f(x_k, y_k).$$

Man berechnet

$$\tilde{y}''(x) = \frac{d}{dx} f(x, y(k)) = \frac{\partial}{\partial x} f(x, y(x)) + \frac{\partial}{\partial y} f(x, y(x)) y'(x).$$

Also

$$\tilde{y}''(x_k) = \frac{\partial}{\partial x} f(x_k, y_k) + \frac{\partial}{\partial y} f(x_k, y_k) f(x_k, y_k).$$

Auf die gleiche Art berechnet man höhere Ableitungen.

Man setzt nun

$$s(t) = y_k + \tilde{y}'(x_k)(t - x_k) + \tilde{y}''(x_k) \frac{(t - x_k)^2}{2},$$

für $x_k \le t \le x_{k+1}$, also

$$y_{k+1} = y_k + \tilde{y}'(x_k)(x_{k+1} - x_k) + \tilde{y}''(x_k) \frac{(x_{k+1} - x_k)^2}{2}.$$

8.18. Beispiel: Das folgende EMT-Programm demonstriert dieses Verfahren und das Eulersche Streckenzugverfahren im Vergleich. Numerische Experimente zeigen, dass das Eulersche Verfahren einen Fehler proportional der Schrittweite h hat, der unter Verwendung der zweiten Ableitung auf h^2 verbessert wird.

```
>function f(x,y) &= -2*x*y;
>function y2(x,y) &= diff(f(x,y),x)+diff(f(x,y),y)*f(x,y)

                                2
                             4 x  y - 2 y

>function fstep(x,y,h) &= y+f(x,y)*h+y2(x,y)*h^2/2

                    2     2
                   h  (4 x  y - 2 y)
                   ----------------- - 2 h x y + y
                           2
```

Wir verwenden nun solche Schritte, um ein Verfahren der Ordnung 2 zu erhalten.

```
>function euler2 (fstep,x,y0)...
$  y=x; y[1]=y0;
$  loop 1 to cols(x)-1;
$    y[#+1]=fstep(x[#],y[#],x[#+1]-x[#]);
$  end;
$  return y;
$endfunction
>x := 0:0.001:1;
>y := euler2("fstep",x,1);
>y[-1]-1/E
 -2.45412439981e-007
>euler("f",x,1)[-1]-1/E
 0.000122769661489
```

Wir Überprüfung noch wie im Beispiel 15 die global Ordnung.

```
>err1000=euler2("fstep",linspace(0,1,1000),1)[-1]-1/E
 -2.45412439537e-007
>err2000=euler2("fstep",linspace(0,1,2000),1)[-1]-1/E
 -6.13331719856e-008
>log(err1000/err2000)/log(2)
 2.00046890829
```

8.19. Beispiel: Es ist nicht schwer, für eine als Ausdruck gegebenen Funktion $f(x,y)$ höhere Ableitungen von $f(x,y(x))$ mit Maxima symbolisch zu berechnen.

```
>function f(x,y) &= -2*x*y;
>function y2(x,y) &= subst(f(x,y(x)),diff(y(x),x),diff(f(x,y(x)),x))

                              2
                         4 x  y(x) - 2 y(x)

>function y3(x,y) &= subst(f(x,y(x)),diff(y(x),x),diff(y2(x,y),x))

                                      3
                       12 x y(x) - 8 x  y(x)
```

Zur Verwendung dieses Audrucks muss man $y(x)$ wieder durch y ersetzen.

```
>&subst(y,y(x),y3(x,y))

                                  3
                          12 x y - 8 x  y
```

Man kann aber auch die Ableitungen in allgemeiner Form mit Maxima symbolisch in Abhängigkeit von $f = f(x,y)$ berechnen, wenn man die Abhängigkeiten von f und y bekannt gibt, und in jedem Schritt y' durch f ersetzt. Maxima berücksichtigt dann auch die Regel $f_{xy} = f_{yx}$.

```
>&depends(y,x); &depends(f,[x,y]),
>derivabbrev &= true;
>y1 &= f
```

$$f$$

```
>y2 &= subst(f,diff(y,x),diff(y1,x))
```

$$f_y f + f_x$$

```
>y3 &= subst(f,diff(y,x),diff(y2,x))|expand
```

$$f_{yy} f^2 + f_y^2 f + f_{xy} f + f_{xx} + 2 f_x f_y$$

8.4 Intervalleinschließung

Um eine gute Einschließung der Lösung zu erhalten, können wir die Taylorformel mit Restglied verwenden. Es gilt

$$y(b) = y(a) + \sum_{k=1}^{n} \frac{y^{(k)}(a)}{k!}(b-a)^k + \frac{y^{(n+1)}(\xi)}{(n+1)!}(b-a)^{n+1}.$$

Wir berechnen dazu die Ableitungen

$$y^{(k)}(t) = F_k(t, y(t))$$

durch fortgesetztes implizites Differenzieren von $f(t, y(t))$ wie im vorigen Abschnitt. Mit $\mathcal{X} = [a, b]$ benötigen wir zur Abschätzung des Restgliedes ein Intervall $\mathcal{Y}$, so dass mit

$$\mathcal{R} = \frac{F_{n+1}(\mathcal{X}, \mathcal{Y})}{(n+1)!}(b-a)^{n+1}$$

die Bedingung

$$y(a) + \sum_{k=1}^{n} \frac{R_k(a, y(a))}{k!}(b-a)^k + \mathcal{R} \subseteq \mathcal{Y}$$

gilt. Da für kleine $|b-a|$ das Intervall $\mathcal{R}$ sehr viel langsamer wächst als $\mathcal{Y}$ lässt sich dies durch vorsichtiges Ausweiten von $\mathcal{Y}$ erreichen.

8.20. Beispiel: EMT verwendet den folgenden Maxima-Code, um die Ableitungen für einen gegebenen Ausdruck zu berechnen. Man erhält die Taylorreihe, sowie das Restglied. Die Technik dazu haben wir schon im Beispiel 19 vorgeführt.

```
dgltaylor(n,expr,h) :=
  block ([d,i,s],
    d:expr, s:y(x)+h*expr,
    for i:2 thru n do
      (d:subst(expr,diff(y(x),x),diff(d,x)), s:s+d*h^i/i!),
    [s,subst(expr,diff(y(x),x),diff(d,x))*h^(n+1)/(n+1)!])$
```

Für das Beispiel $y' = -2 * x * y$ liefert das die folgende Entwicklung.

```
>function f(x,y) &= -2*x*y

                                - 2 x y

>&dgltaylor(1,f(x,y(x)),h)

                                  2     2
                                 h  (4 x  y(x) - 2 y(x))
            [y(x) - 2 h x y(x), ----------------------]
                                          2
```

Die Funktion `mxmidgl` liefert mit Hilfe dieses Algorithmus eine Intervalleinschließung der Differentialgleichung zurück. Man beachte, dass die Formeln für die Taylorreihe schnell recht groß werden können. Die Formeln für die Taylorreihe und den Rest werden im folgenden Beispiel implizit in der Funktion `mxmidgl` berechnet. Verwendet wird per Default der Grad 10.

```
>x := 0:0.01:1;
>y := mxmidgl(&f(x,y),x,1);
>y[-1]
 ~0.36787944117136,0.36787944117152~
>longestformat; 1/E
      0.3678794411714423
```

Ohne Kenntnisse über die Ableitungen von f gelingt die Abschätzung mit $n = 0$. Wir erhalten in diesem Fall den folgenden Satz.

8.21 Satz: *Sei $f : [x_k, x_{k+1}] \times \mathcal{Y} \to \mathbb{R}$ eine stetige Abbildung, $\mathcal{Y} \subseteq \mathbb{R}$ ein kompaktes Intervall, $y_k \in \mathcal{Y}$*

$$y_k + f([x_k, x_{k+1}], \mathcal{Y})(x_{k+1} - x_k) \subseteq \mathcal{Y}.$$

Dann existiert eine Lösung des Anfangswertproblems

$$y'(x) = f(x, y(x)), \quad y(x_k) = y_k,$$

die auf ganz $[x_k, x_{k+1}]$ definiert ist, mit

$$y(x_{k+1}) \in y_k + f([x_k, x_{k+1}], \mathcal{Y})(x_{k+1} - x_k)$$

für alle $t \in [x_k, x_{k+1}]$.

In Abbildung 8.5 ist im Bild dargestellt, wie die Einschließung funktioniert. Unter der Voraussetzung des Satzes liegen alle Lösungen des Anfangswertproblems im grünen Streifen. In der Abbildung ist $h = b - a$.

Der Beweis der Existenz einer Lösung benutzt Hilfsmittel aus der Funktionalanalysis, die wir hier nicht wiedergeben wollen. Im Prinzip wird gezeigt, dass jede Folge von immer feineren Eulerschen Streckenzügen eine Lösung als Häufungspunkt hat.

8.22 Aufgabe: Zeigen Sie, dass der Satz aus dem Mittelwertsatz folgt, wenn die Existenz der Lösung gesichert ist.

Der Satz gilt mit den entsprechenden Veränderungen auch für $n > 1$.

8.23. Beispiel: Um eine enge Einschließung zu erhalten, muss man sehr kleine Schrittweiten wählen. In EMT ist dies mit `idgl` implementiert. Die Ordnung ist nur 1, und daher ist das Verfahren nur in Notfällen brauchbar.

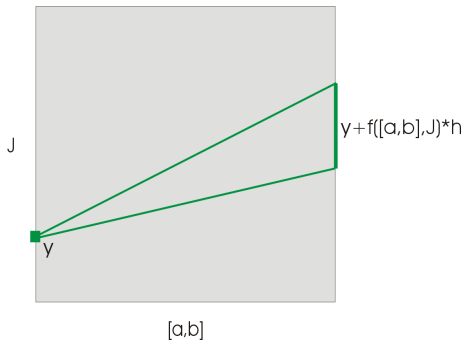

Abbildung 8.5: Intervalleinschließung eines Anfangswertproblems

```
>function f(x,y) := -2*x*y
>idgl("f",0:0.0001:1,1)[-1]
 ~0.36756,0.3682~
```

8.5 Konvergenzordnung

Wir betrachten in diesem Abschnitt allgemeine Verfahren der Form

$$y_{k+1} = y_k + \Phi(x_k, y_k, x_{k+1} - x_k)(x_{k+1} - x_k)$$

für Unterteilungen

$$x_0 < \ldots < x_m.$$

Dabei sei $\Phi(x, y, h)$ für $(x, y) \in U$ und $h > 0$ definiert und stetig.

8.24. Definition: Die Funktion Φ heißt konsistent mit der Differentialgleichung

$$y'(x) = f(x, y(x)),$$

wenn

$$\Phi(x, y, 0) = f(x, y)$$

gilt.

8.25. Definition: Sei $y : [a, b] \to \mathbb{R}$ eine Lösung des Anfangswertproblems

$$y'(x) = f(x, y(x)), \quad y(a) = y_0.$$

Dann heißt das durch Φ gegebene Verfahren konvergent, wenn für alle $x \in [a, b]$ und Unterteilungen

$$a = x_0 < x_1 < \ldots < x_m = x$$

gilt, dass $y_m \to y(x)$ konvergiert, wobei mit $m \to \infty$ auch die Feinheit der Unterteilung gegen 0 geht. Analog definiert man die Konvergenz, falls $x_0 = b$ ist.

Wir setzen in diesem Abschnitt außerdem voraus, dass sowohl f als auch Φ einer Lipschitzbedingung in y genügen. Also

$$\|f(x, y_1) - f(x, y_2)\| \leq L\|y_1 - y_2\|$$

und

$$\|\Phi(x, y_1, h) - \Phi(x, y_2, h)\| \leq L\|y_1 - y_2\|$$

für alle $(x, y_1), (x, y_2) \in U$ und $h > 0$, so dass $(x + h, y_1), (x + h, y_2) \in U$ ist.

8.26. Definition: Sei $y : [a, b] \to \mathbb{R}^n$ eine exakte Lösung unserer Differentialgleichung. Wir sagen, Φ habe einen lokalen Diskretisierungsfehler der Ordnung $p > 0$, wenn

$$\|y(x_{k+1}) - (y(x_k) + \Phi(x_k, y(x_k), x_{k+1} - x_k)(x_{k+1} - x_k))\| \leq C(x_{k+1} - x_k)^p$$

mit einer Konstanten $C > 0$ für alle möglichen Unterteilungen des Intervalls $[a, b]$ gilt.

Wir können in diesem Fall also schreiben

$$y(x_{k+1}) - y(x_k) = \Phi(x_k, y(x_k), x_{k+1} - x_k)(x_{k+1} - x_k) + \epsilon_k$$

mit

$$\|\epsilon_k\| \leq C(x_{k+1} - x_k)^p.$$

Der lokale Diskretisierungsfehler ist der Fehler in einem Schritt des Verfahrens, wenn man mit den exakten Werten $(x_k, y(x_k))$ starten würde.

8.27 Satz: *Wenn die Funktion Φ einen lokalen Diskretisierungsfehler der Ordnung $p > 1$ gemäß obiger Definition hat und einer Lipschitzbedingung mit Konstanten L genügt, so konvergiert das Verfahren und hat eine globale Konvergenzordnung $p - 1$. Genauer gilt*

$$\|y(x_m) - y_m\| \leq Ce^{L(b-a)}(b - a)h^{p-1},$$

wobei

$$h = \max_k (x_{k+1} - x_k)$$

die Feinheit der Unterteilung bezeichne, und $C > 0$ dieselbe Konstante, die bei der Abschätzung des lokalen Diskretisierungsfehlers auftaucht.

Beweis: Bezeichne $E_k = \|y_k - y(x_k)\|$ den Fehler des Verfahrens im n-ten Schritt. Dann gilt mit der Abkürzung $h_k = x_{k+1} - x_k$

$$\begin{aligned}
E_{k+1} &= \|y_{k+1} - y(x_{k+1})\| \\
&= \|(y_{k+1} - y_k) + (y_k - y(x_k)) + (y(x_k) - y(x_{k+1}))\| \\
&= \|\Phi(x_k, y_k, h_k)h_k + (y_k - y(x_k)) - (\Phi(x_k, y(x_k), h_k)h_k + \epsilon_k))\| \\
&\leq \|\Phi(x_k, y_k, h_k) - \Phi(x_k, y(x_k), h_k)\| h_k + E_k + Ch_k^p \\
&\leq (1 + Lh_k)E_k + Ch_k^p
\end{aligned}$$

Rekursiv erhält man wegen $1 + x \leq e^x$

$$\begin{aligned} E_m &\leq C(e^{L(h_0+\ldots+h_{m-1})}h_0^p + \ldots + h_{m-1}^p) \\ &\leq Ce^{L(b-a)} \sum_k h_k^p \\ &\leq Ce^{L(b-a)} h^{p-1}(h_0 + \ldots + h_{m-1}) \\ &\leq Ce^{L(b-a)}(b-a)h^{p-1}. \end{aligned}$$

Es folgt die Behauptung. **q.e.d.**

8.28 Satz: *Ein konsistentes Φ, das einer Lipschitzbedingung genügt, führt zu einem konvergenten Verfahren.*

Beweis: Wir schätzen den lokalen Diskretisierungsfehler ab. Mit der Abkürzung $h_k = x_{k+1} - x_k$ gilt wegen $y'(x_k) = f(x_k, y(x_k))$

$$\begin{aligned} &\|y(x_{k+1}) - (y(x_k) + \Phi(x_k, y(x_k), h_k)h_k)\| \\ &\qquad \leq h_k\big(\|\frac{1}{h_k}(y(x_{k+1}) - y(x_k)) - y'(x_k)\| \\ &\qquad\qquad + \|f(x_k, y(x_k)) - \Phi(x_k, y(x_k), h_k)\|\big) \end{aligned}$$

Aufgrund der Konsistenzbedingung und der stetigen Differenzierbarkeit der Funktion y lässt sich für jedes $\epsilon > 0$ der lokale Diskretisierungsfehler mit ϵh abschätzen, wenn die Feinheit h der Unterteilung klein genug ist, wobei $\epsilon > 0$ global auf jeder kompakten Teilmenge von U gewählt werden kann. Die Behauptung folgt also aus der Abschätzung im Beweis des obigen Satzes. **q.e.d.**

8.29. Beispiel: Da das Eulersche Streckenzugverfahren konsistent ist, konvergiert es gegen die Lösung der Differentialgleichung, wenn f einer Lipschitzbedingung genügt.

Um die Konvergenzordnung des Eulerschen Streckenzugverfahrens realistisch abzuschätzen, nehmen wir an, dass die Lösung y zweimal stetig differenzierbar ist. Zunächst gilt

$$\frac{1}{h_k}(y(x_{k+1}) - y(x_k)) = \begin{pmatrix} y_1'(\xi_1) \\ \vdots \\ y_n'(\xi_n) \end{pmatrix}$$

an Zwischenstellen $x_k \leq \xi_i \leq x_{k+1}$. Wendet man den Zwischenwertsatz nochmals mit Hilfe der zweiten Ableitung $y''(x)$ an, so erhält man

$$\begin{aligned} &\|y(x_{k+1}) - (y(x_k) + \Phi(x_k, y(x_k), h_k)h_k)\| \\ &\qquad \leq h_k\|\frac{1}{h_k}(y(x_{k+1}) - y(x_k)) - y'(x_k)\| \leq Ch_k^2 \end{aligned}$$

Mit einem $C > 0$.

Das Eulersche Streckenzugverfahren hat also die lokale Diskretisierungsordnung $p = 2$, und damit globale Konvergenzordnung $p = 1$.

8.6 Einschrittverfahren höherer Ordnung

Wir stellen in diesem Abschnitt exemplarisch Einschrittverfahren vor. Dies sind Verfahren, die weder Werte aus vorherigen Schritten verwenden, noch partielle Ableitungen von f.

Ein einfaches Verfahren mit einer globalen Konvergenzordnung $p = 4$ ist das vierstufige Runge-Kutta-Verfahren. Man berechnet nacheinander

$$\begin{aligned} k_1 &= f(x_k, y_k), \\ k_2 &= f(x_k + h/2, y_k + hk_1/2), \\ k_3 &= f(x_k + h/2, y_k + hk_2/2), \\ k_4 &= f(x_k + h, y_k + hk_3), \end{aligned}$$

und setzt schließlich

$$y_{k+1} = y_k + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4).$$

Das Verfahren ist sehr einfach aufgebaut. Es verwendet in jedem Schritt viermal die Funktion $f(x, y)$.

Das folgende Java-Programm testet dieses Verfahren für das Anfangswertproblem

$$y'(x) = 2xy(x), \quad y(0) = 1$$

mit Lösung

$$y(x) = e^{x^2}$$

im Intervall $[0, 1]$. Das Programm gibt tatsächlich $p = 3.99$ als Ordnung aus. Die Ordnung wird wie im Beispiel 15 berechnet.

```
public class RungeKutta
    implements DGLFunction
{
    public double f (double x, double y)
    {
        return 2*x*y;
    }

    /**
     * Vierstufige Runge-Kutta-Methode (Globale Ordnung 4)
     * @param x0 Startwert
     * @param y0 Startwert in x[0]
     * @param x Zielwert
     * @param n Schritte
     * @param f Interface mit Funktion f(x,y)
     * @return Vektor der y-Werte
     */
    public static double compute (double x0, double y0,
        double x, int n, DGLFunction f)
    {
        double h=(x-x0)/n;
        for (int i=0; i<n; i++)
        {
            x=x0+i*h;
            double k1=f.f(x,y0);
```

```
            double k2=f.f(x+h/2,y0+h*k1/2);
            double k3=f.f(x+h/2,y0+h*k2/2);
            double k4=f.f(x+h,y0+h*k3);
            y0=y0+h/6*(k1+2*k2+2*k3+k4);
        }
        return y0;
    }

    /**
     * Testet die Runge-Kutta-Methode mit der DGL y'=2xy.
     * Schrittweiten sind 5,10,20,40,...,1280
     * Die Stützpunkte werden äquidistant gewählt.
     */
    public static void main (String[] args)
    {
        double olderror=0.0;
        for (int n=5; n<=1280; n*=2)
        {
            double y=compute(0.0,1.0,1.0,n,new RungeKutta());
            double error=Math.abs(y-Math.exp(1.0));
            System.out.println("n="+n+", Fehler="+error);
            if (olderror!=0.0)
            {
                double p=Math.log(olderror/error)/Math.log(2);
                System.out.println("Ordnung="+p);
            }
            olderror=error;
        }
    }
}
```

Benötigt wird noch das folgende Interface.

```
interface DGLFunction
{
    public double f (double x, double y);
}
```

8.30 Satz: *Das vierstufige Runge-Kutta-Verfahren hat die globale Konvergenzordnung 4, wenn die Funktion $f(x,y)$ viermal stetig partiell differenzierbar ist.*

Beweis: Der Beweis besteht aus langwierigen Rechnungen. Man berechnet zunächst

$$\begin{aligned} y' &= f(x,y) = f \\ y'' &= f_x + f_y y' = f_x + f f_y \\ y''' &= \ldots \\ y'''' &= \ldots \end{aligned}$$

Dies setzt man in die Taylor-Entwicklung von y ein.

$$y(x+h) = y(x) + \sum_{\nu=1}^{4} \frac{y^{\nu}(x)}{\nu!} h^{\nu} + O(h^5)$$

ein. Dabei steht $O(h^5)$ für eine Funktion $r(h)$ mit

$$|r(h)| \leq C h^5.$$

Bei gegebenen Werten x_k und $y(x_k)$ erhält man auf diese Weise eine Entwicklung

$$y(x_{k+1}) - y(x_k) = \sum_{i=1}^{4} S_i(f)h^i + O(h^5)$$

mit $h = x_{k+1} - x_k$. Dabei sind die Koeffizienten $S_i(f)$ Ausdrücke in den partiellen Ableitungen von f an der Stelle (x_k, y_k).

Damit ein Verfahren mit einem lokalen Diskretisierungsfehler der Ordnung 5 entsteht, muss schließlich $S_i(f) = T_i(f)$ für $i = 1, \ldots, 4$ sein.

Mit Maxima kann man dies symbolisch ausrechnen.

```
>derivabbrev &= true;
>&depends(f,[x,y]);
>&depends(y,x);
>y1 &= f;
>y2 &= subst(f,diff(y,x),diff(y1,x));
>y3 &= subst(f,diff(y,x),diff(y2,x))|expand;
>y4 &= subst(f,diff(y,x),diff(y3,x))|expand;
>L &= taylor(y1*h+y2*h^2/2+y3*h^3/6+y4*h^4/24,h,0,4)
```

```
          3             2                             3          2
       ((f  f      + (4 f  f  + 3 f f ) f    + f (f )  + f  (f )
             y y y          y        x   y y       y      x   y

       ... + f h
```

Analog entwickelt man die Ausdrücke k_i in eine Taylorreihe mit Variable h und berechnet

$$y_{k+1} - y_k = \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) = \sum_{i=1}^{4} T_i(f)h^i + O(h^5).$$

Dabei verwendet man die Taylorreihe von f bis zur Ordnung 3.

```
>&remove(y,dependency)
```

```
                                done
```

```
>f1 &= diff(f,x)*hx+diff(f,y)*hy
```

```
                             f  hy + f  hx
                              y       x
```

```
>f2 &= diff(f1,x)*hx+diff(f1,y)*hy|expand
```

```
                          2                          2
                 f    hy  + 2 f    hx hy + f    hx
                  y y          x y          x x
```

```
>f3 &= diff(f2,x)*hx+diff(f2,y)*hy|expand;
>function Tf(hx,hy) &= f+f1+f2/2+f3/6;
>k1 &= f;
>k2 &= Tf(h/2,k1*h/2);
>k3 &= taylor(Tf(h/2,k2*h/2),h,0,4);
>k4 &= taylor(Tf(h,k3*h),h,0,4);
```

```
>K &= taylor(expand(h/6*(k1+2*k2+2*k3+k4)),h,0,4);
>&L-K
```

$$0$$

Es stellt sich heraus, dass der Ausdruck K identisch mit dem obigen Ausdruck L ist. **q.e.d.**

8.31. Beispiel: Wir testen die Ordnung des in EMT eingebauten Runge-Verfahrens wie in Beispiel 15. Dabei müssen wir allerdings auf $n = 100, 200$ zurückgehen, da sich sonst fast 0 als Fehler ergibt.

```
>function f(x,y) := -2*x*y
>err100=runge("f",linspace(0,1,100),1)[-1]-1/E
 1.63759839023e-010
>err200=runge("f",linspace(0,1,200),1)[-1]-1/E
 1.0227430014e-011
>log(err100/err200)/log(2)
 4.00106601924
```

8.32. Beispiel: Per Hand können wir eine zweistufige Methode herleiten. Es gilt

$$\begin{aligned} y' &= f(x_k, y_k) = f, \\ y'' &= f_x + f_y y' = f_x + f f_y. \end{aligned}$$

Wir haben daher

$$y(x_k + h) = y(x_k) + fh + \frac{h^2}{2}(f_x + f f_y) + O(h^3).$$

Unser Verfahren verwendet

$$\begin{aligned} k_1 &= f(x_k, y_k) \\ k_2 &= f(x_k + h, y_k + k_1 h) \end{aligned}$$

und schließlich

$$y_{k+1} = y_k + \frac{h}{2}(k_1 + k_2).$$

Dies entwickeln wir nach h, also

$$\begin{aligned} k_1 &= f \\ k_2 &= f + f_x h + f_y k_1 h + O(h^2) \\ &= f + f_x h + f_y f h + O(h^2) \end{aligned}$$

Damit wird

$$y_{k+1} = y_k + fh + \frac{h^2}{2}(f_x + f f_y) + O(h^3).$$

Damit wird der lokale Diskretisierungsfehler

$$|y_{k+1} - y(x_k + h)| = O(h^3)$$

8.33 Aufgabe: Testen Sie am Beispiel $y' = -2xy$, ob dieses Verfahren tatsächlich die globale Ordnung $p = 2$ hat.

8.34. Beispiel: Man kann die hier vorgestellten Verfahren auch anwenden, um Anfangswertprobleme mit $n > 1$ zu lösen. Als Beispiel lösen wir die Gleichung von Lotke-Volterra zur Modellierung von Jäger-Beute-Populationen aus der Biologie. Die Gleichungen lauten

$$y_1'(t) = ay_1(t)(1 - y_2(t)),$$
$$y_2'(t) = y_2(t)(y_1(t) - 1).$$

Dabei sei $y_1(t)$ die Populationsgröße der Beute zur Zeit t, deren Zuwachs von der aktuellen Größe abhängt, aber auch von der Anzahl der Jäger $y_2(t)$. Analog hängt der Zuwachs der Jäger von deren Populationsgröße ab, aber auch vom verfügbaren Nahrungsangebot.

8.35. Beispiel: Wir verwenden die Runge-Kutta-Methode, die in EMT eingebaut ist, um das Jäger-Beute-Modell zu lösen (siehe Abbildung 8.6). Mit $a = 10$ ergibt sich ein periodisches Verhalten der Funktion y_1 und y_2. Benutzt wurde folgender einfacher Programmcode.

```
>function f(x,y) := [10*y[1]*(1-y[2]),y[2]*(y[1]-1)];
>x=0:0.01:5;
>y=runge("f",x,[3,1]);
>xplot(x,y);
```

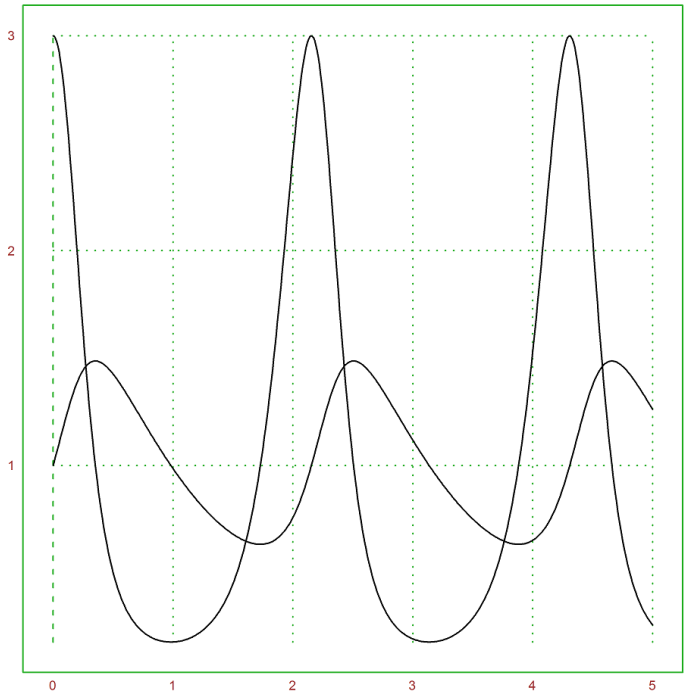

Abbildung 8.6: Jäger-Beute-Modell

8.36 Aufgabe: Betrachten Sie alle möglichen Verfahren des Typs

$$\begin{aligned} k_1 &= f(x_k, y_k) \\ k_2 &= f(x_k + ah, y_k + h(b + ck_1)) \\ y_{k+1} &= y_k + h(dk_1 + ek_2) \end{aligned}$$

mit der globalen Konvergenzordnung $p = 2$. Welche Bedingungen sind an die Koeffizienten a, b, c, d, e zu stellen?

8.37 Aufgabe: Zeigen Sie, dass das Verfahren

$$\begin{aligned} k_1 &= f(x_k, y_k) \\ k_2 &= f(x_k + \frac{h}{2}, y_k + \frac{h}{2}k_1) \\ k_3 &= f(x_k + h, y_k - hk_1 + 2hk_2) \\ y_{k+1} &= y_k + \frac{h}{6}(k_1 + 4k_2 + k_3) \end{aligned}$$

die globale Konvergenzordnung $p = 3$ hat. Benutzen Sie dazu die Reihenentwicklung

$$f(x + h_1, y + h_2) = f(x, y) + f_x h_1 + f_y h_2 + f_{xx}\frac{h_1^2}{2} + f_{xy} h_1 h_2 + f_{yy}\frac{h_2^2}{2} + O(h^3).$$

Für die Lösungsfunktion y benutzen Sie eine Reihenentwicklung

$$y(x + h) = y(x) + hy'(x) + \frac{h^2}{2}y''(x) + \frac{h^3}{3}y'''(x) + O(h^4).$$

8.7 Implizite Verfahren

8.38. Definition: Verfahren, bei denen die neuen Werte nicht explizit aus den alten berechnet werden, sondern bei denen die neuen Werte durch Lösen eines Systems von Gleichungen bestimmt werden müssen, nennt man implizite Verfahren.

8.39. Beispiel: Ein Beispiel ist folgendes zweistufige Verfahren der Ordnung $p = 4$.

$$\begin{aligned} k_1 &= f(x_k + \frac{3 - \sqrt{3}}{6}h, y_k + \frac{1}{4}hk_1 + \frac{3 - 2\sqrt{3}}{12}hk_2) \\ k_2 &= f(x_k + \frac{3 + \sqrt{3}}{6}h, y_k + \frac{3 + 2\sqrt{3}}{12}hk_1 + \frac{1}{4}hk_2) \end{aligned}$$

und schließlich

$$y_{k+1} = y_k + \frac{h}{2}(k_1 + k_2).$$

Das Problem ist, dass man die Gleichungen zur Berechnung der k_i nur iterativ lösen kann. Es erweist sich allerdings, dass ein wenige Iterationsschritte ausreichen, um an die Ordnung 4 heranzukommen. Die Beschaffung der Startwerte von k_1 und k_2 nennt man Prädiktor-Schritt, und die Iterationsschritte nennt man Korrektor-Schritte.

8.40. Beispiel: Um die optimale Anzahl der Iterationen zu testen, verwenden wir das folgende EMT-Programm. Es stellt sich in diesem Beispiel heraus, dass 3 Iterationen mit insgesamt 7 Funktionsauswertungen genügen, um die Ordnung 4 zu erreichen.

```
>function f(x,y) := -2*x*y
>function impl (f,x,y0,iter=1)...
$  a=(3-sqrt(3))/6; b=(3+sqrt(3))/6;
$  c=(3-2*sqrt(3))/12; d=(3+2*sqrt(3))/12;
$  k1=0; k2=0;
$  for i=1 to cols(x)-1
$  h=x[i+1]-x[i];
$  k1=f(x[i],y0); k2=k1;
$  loop 1 to iter
$    k1=f(x[i]+a*h,y0+h*(k1/4+c*k2));
$    k2=f(x[i]+b*h,y0+h*(d*k1+k2/4));
$  end;
$  y0=y0+h/2*(k1+k2);
$  end;
$  return y0
$endfunction
>x=0:0.01:1; impl("f",x,1)-1/E
 -3.7412423326e-008
>x=0:0.01:1; impl("f",x,1,2)-1/E
 -4.65449961906e-009
>x=0:0.01:1; impl("f",x,1,3)-1/E
 -5.05374631032e-011
```

8.41 Aufgabe: Testen Sie die Ordnung für 1, 2, 3, 4 Iterationsschritte.

```
>function f(x,y) := -2*x*y
>err100=impl("f",linspace(0,1,100),1,3)[-1]-1/E
 -5.05369079917e-011
>err200=impl("f",linspace(0,1,200),1,3)[-1]-1/E
 -3.16185966298e-012
>log(err100/err200)/log(2)
 3.99849215715
```

8.42. Beispiel: Oft reicht ein Korrektor-Schritt aus, um ein optimales Ergebnis zu erreichen. Weitere Schritte bringen keine Verbesserung. Als einfach zu rechnendes Beispiel nehmen wir die implizite Methode

$$k_1 = f(x_k + \frac{h}{2}, y_k + \frac{h}{2}k_1).$$

und

$$y_{k+1} = y_k + hk_1.$$

Wir erhalten

$$k_1 = f + \frac{h}{2}f_x + f_y\frac{h}{2}k_1 + O(h^2).$$

Setzt man dies einmal in sich selbst ein, so sieht man

$$k_1 = f + \frac{h}{2}(f_x + f_y f) + O(h^2).$$

Dies ist exakt das richtige Verhalten für ein Verfahren mit globaler Konvergenzordnung $p = 2$. Allerdings wird diese Ordnung auch schon erreicht, wenn man

$$k_1 = f(x_k + \frac{h}{2}, y_k + \frac{h}{2}f(x_k, y_k))$$

setzt. Man nennt diese explizite Variante des Verfahrens verbessertes Eulersches Streckenzug-verfahren.

8.8 Mehrschrittverfahren

8.43. Definition: Einschrittverfahren gehen immer nur vom Punkt (x_k, y_k) aus. explizite Mehrschrittverfahren verwenden dagegen zusätzliche $u+1$ frühere Werte

$$y_{k-u}, \ldots, y_k.$$

Eine einfache Idee ist, die Differentialgleichung als Intergralgleichung umzuschreiben. Also

$$y(x_{k+1}) = y(x_k) + \int_{x_k}^{x_{k+1}} y'(x)\,dx = y(x_k) + \int_{x_k}^{x_{k+1}} f(x, y(x))\,dx.$$

Nun wird als Näherung für $f(x, y(x))$ das Interpolationspolynom

$$p(x_{k-u}) = f_{k-u} = f(x_{k-u}, y_{k-u}), \quad \ldots, \quad p(x_k) = f_k = f(x_k, y_k)$$

vom Grad u verwendet. Man erhält auf diese Weise die Mehrschrittverfahren von Adams-Bashforth.

Mehrschrittverfahren müssen natürlich gestartet werden, indem man sich die Werte

$$y_1, \ldots, y_u$$

mit anderen Verfahren beschafft. Klar ist, dass der lokale Fehler dieser Startverfahren gut genug sein muss.

8.44 Satz: *Die globale Ordnung der Methode von Adams-Bashforth mit äquidistanter Schrittweite ist $u+1$.*

Beweis: Wir nehmen an, dass die Startschritte genau bestimmt sind, so dass

$$E_u = c_1\, h^{u+1}$$

ist, wobei E_k diesmal den maximalen Fehler der ersten k Schritte bezeichne. Aufgrund der Lipschitz-Bedingung lässt sich der Fehler für die ersten f_k dann durch LE_k abschätzen. Für das Interpolationspolynom $\tilde{p}$ vom Grad u an die exakten Werte für y' folgt aus der Restgliedabschätzung des Interpolationsfehlers eine Fehlerabschätzung

$$|\tilde{p}(x) - y'(x)| = c_2\, h^{u+1} \qquad \text{für } x_k \le x \le x_{k+1},$$

wobei die Konstante c_2 für alle Schritte gleich gewählt werden kann. Bezeichne p das Interpolationspolynom gemäß der Beschreibung des Verfahrens im k-ten Schritt. Da die Differenz $p - \tilde{p}$ auf den Interpolationspunkten maximal LE_k auseinander liegt, bekommt man

$$|p(x) - \tilde{p}(x)| = c_3\, LE_k \qquad \text{für } x_k \le x \le x_{k+1}$$

Insgesamt erhält man wegen der Definition des Verfahrens

$$E_{k+1} = E_k + c_3 h_k L E_k + c_2 O(h^{u+2}).$$

Daraus folgt die Behauptung wie im Beweis von Satz 27. **q.e.d.**

Es ist für diesen Beweis nicht wirklich nötig, dass die Schrittweite äquidistant ist. Jedoch muss eine untere Schranke für h_k existieren, da sonst die Schranke für $p - \tilde{p}$ fehl schlägt.

8.45. Beispiel: Mit $u = 3$ und äquidistanten Stützstellen ergibt sich

$$y_{k+1} = y_k + \frac{h}{24}(55f_k - 59f_{k-1} + 37f_{k-2} - 9f_{k-3}).$$

mit

$$f_i = f(x_i, y_i).$$

Dies ist die 4-schrittige Methode von Adams-Bashforth. Es braucht in jedem Schritt nur ein Funktionswert ausgewertet zu werden. Aufgrund der negativen Koeffizienten könnten allerdings Auslöschungseffekte auftreten.

Wir können diese Gleichungen in Maxima folgendermaßen herleiten.

```
>function p(x) &= a0 + a1 * x + a2 * x^2 + a3 * x^3;
>sol &= solve([p(-3*h)=d,p(-2*h)=c,p(-h)=b,p(0)=a],[a0,a1,a2,a3]);
>&integrate(p(x) with sol[1],x,0,h)

                    (- 9 d + 37 c - 59 b + 55 a) h
                    ------------------------------
                                  24
```

Das folgende EMT-Programm verwendet die vierstufige explizite Methode von Runge-Kutta, um die ersten vier Startwerte zu berechnen. Es stellt sich heraus, dass diese Methode bei aufwändig zu berechnenden Funktionen f schneller zum Ziel kommt als Runge-Kutta. Wir vergleichen im Beispiel zwei Rechnungen mit etwa gleich vielen Funktionsauswertungen.

```
>function p(x) &= a0 + a1 * x + a2 * x^2 + a3 * x^3;
>sol &= solve([p(-3*h)=d,p(-2*h)=c,p(-h)=b,p(0)=a],[a0,a1,a2,a3]);
>function step([d,c,b,a],h) &= integrate(p(x) with sol[1],x,0,h);
>function adams (f,a,b,n,y0)...
$  x=linspace(a,b,n);
$  y=zeros(size(x));
$  y[1:4]=runge(f,x[1:4],y0);
$  d=f(x[1:4],y[1:4]);
$  h=(b-a)/n;
$  for i=5 to n+1
$  y[i] = y[i-1]+step(d,h);
$  d = rotleft(d); d[4]=f(x[i],y[i]);
$  end;
$  return y;
$endfunction
>function f(x,y) := -2*x*y;
>y=adams("f",0,1,400,1);
>y[-1]-1/E
 1.27548138718e-010
>runge("f",0:0.01:1,1)[-1]-1/E
 1.63759394933e-010
```

8.46 Aufgabe: Testen Sie das Verfahren, und bestimmen Sie numerisch die Ordnung für $y' = -2xy$ wie im Beispiel 15 mit $n = 100, 200$.

Das Verfahren wurde im Folgenden noch in Java implementiert. Es verwendet die Klasse `RungeKutta` für die Startwerte.

```
/**
 * Adams-Bashforth-Methode (Globale Ordnung 4)
 * @param x0 Startwert
 * @param y0 Startwert in x[0]
 * @param x Zielwert
 * @param n Schritte
 * @param f Interface mit Funktion f(x,y)
 * @return Vektor der y-Werte
 */
public static double compute (double x0, double y0,
    double x1, int n, DGLFunction F)
{
    double h=(x1-x0)/n;
    // Berechne die ersten 4 Werte zum Starten:
    double x[]=new double[4];
    for (int i=0; i<4; i++) x[i]=x0+i*h;
    double f[]=RungeKutta.compute(x,y0,F);
    y0=f[3];
    // Speichere f_0,...,f_3 ab:
    for (int i=0; i<4; i++)
        f[i]=F.f(x[i],f[i]);
    // Extrapolationsschritte:
    y0=y0+h/24*(55*f[3]-59*f[2]+37*f[1]-9*f[0]);
    for (int i=4; i<n; i++)
    {
        for (int j=0; j<3; j++) f[j]=f[j+1];
        f[3]=F.f(x0+i*h,y0);
        y0=y0+h/24*(55*f[3]-59*f[2]+37*f[1]-9*f[0]);
    }
    return y0;
}
```

8.47 Aufgabe: Leiten Sie die explizite 2-schrittige Adams-Bashforth-Methode her.

8.48. Definition: Als weiteres Beispiel für ein implizites Mehrschrittverfahren betrachten wir ein Prädiktor-Korrektor-Verfahren von Adams-Bashforth-Moulton (A-B-M-Methode). Der Prädiktor-Schritt ist

$$y_{k+1}^P = y_k + \frac{h}{12}(23f_k - 16f_{k-1} + 5f_{k-2}).$$

Dies ist die explizite 3-schrittige Methode von Adams-Bashforth. Nun verbessert man diesen Prädiktor-Wert mit dem Korrektor

$$y_{k+1} = y_k + \frac{h}{24}(9f(x_{k+1}, y_{k+1}^P) + 19f_k - 5f_{k-1} + f_{k-2}).$$

Das entspricht dem Integral eines Interpolationspolynoms durch die nun bekannten Werte. Es stellt sich heraus, dass die globale Konvergenzordnung dieser Methode ebenfalls $p = 4$ ist. Der Vorteil ist, dass man nur 3 Startwerte benötigt.

8.49. Beispiel: Es zeigt sich, dass man sogar eine implizite, 2-Schritt-Methode mit globaler Konvergenzordnung $p = 4$ konstruieren kann. Die Methode ist sogar recht einfach.

$$y_{k+1} = y_{k-1} + \frac{h}{3}(f(x_{k+1}, y_{k+1}) + 4f_k + f_{k-1}). \tag{8.1}$$

Dies ist die Simson-Regel, angewandt auf

$$y(x_{k+1}) = y(x_{k-1}) + \int_{x_{k-1}}^{x_{k+1}} f(x, y(x))\, dx.$$

Es stellt sich heraus, dass man mit zwei Korrektor-Schritten ein Verfahren der Ordnung 4 bekommt. Als Prädiktor verwendet man einfach die Adams-Bashforth-Methode mit zwei Schritten

$$y_{k+1} = y_k + \frac{h}{2}(3f_k - f_{k-1}),$$

und führt zweimal die Korrektur aus. In jedem Schritt fallen dann drei Funktionsauswertungen an. Es werden zwei alte Funktionswerte verwendet.

8.50 Aufgabe: Implementieren Sie dieses Verfahren in EMT, und testen Sie seine Konvergenzordnung.

Man beachte, dass eine einfache Interpolation mit den Werten von $y_{k-u}, \ldots, y_k$ nicht stabil ist. Dies liegt daran, dass der Interpolationsfehler bei der Extrapolation mit einer festen Konstanten M, die unabhängig von der Schrittweite ist, multipliziert wird, was zu einer Explosion der Fehlerfortpflanzung führt. Beim oben vorgestellten Verfahren wird der Fehler mit einer Konstanten $1 + Mh_k$ multipliziert.

8.51. Definition: Eine Klasse von impliziten Mehrschrittverfahren verwendet Interpolation in

$$x_{k-u}, \quad \ldots, \quad x_k, \quad x_{k+1}$$

mit den bekannten Werten und einem Wert y_{k+1}, der von einem Prädiktor bestimmt wird, und versucht $y_{k+1} = p(x_{k+1})$ so zu bestimmen, dass

$$p'(x_{k+1}) = f\left(x_{k+1}, y_{k+1}\right))$$

gilt. Diese Verfahren heißen BDF-Verfahren (backward differential formulas). Das nicht-lineare Problem kann durch Iteration gelöst werden. Man bestimmt ein Interpolationspolynom mit

$$p'(x_{k+1}) = f(x_{k+1}, y_{k+1}).$$

und setzt im nächsten Schritt

$$y_{k+1} = p(x_{k+1}).$$

Alternativ wäre es möglich, Quasi-Newton-Verfahren anzuwenden, um die Gleichung zu lösen. Für Differentialgleichungen im $\mathbb{R}^m$ bietet sich das Broyden-Verfahren an. Die Interpolation wird dabei in jeder Komponenten durchgeführt.

BDF-Verfahren werden trotz der Probleme bei der Berechnung bevorzugt, weil sie eine größere Stabilität aufweisen, wie man etwa im Beispiel der folgenden Aufgabe sieht.

8.52 Aufgabe: Zeigen Sie, dass das Verfahren

$$y_{k+1} = y_k + h_k f(x_{k+1}, y_{k+1})$$

von diesem Typ ist (implizites Streckenzugverfahren). Berechnen Sie die Iterationsschritte für die Differentialgleichung

$$y' = -cy, \quad y(0) = 1$$

mit äquidistanten $h = 1/n$. Zeigen Sie

$$\lim_{k\to\infty} y_k = 0$$

unabhängig von n und c. Zeigen Sie, dass das für das normale Streckenzugverfahren nicht der Fall ist.

8.53. Beispiel: Mit Maxima kann man die Koeffizienten des Verfahrens berechnen. Wir berechnen dazu $p(h)$ für $p \in \mathcal{P}_2$ mit

$$p(-h) = y_0, \quad p(0) = y_1, \quad p'(h) = f_2.$$

Das Ergebnis ist eine Funktion `ystep` mit

$$y_s(y_0, y_1, f_2, h) = p(h).$$

Mit dieser Funktion wird dann y_2 berechnet.

```
>function p(x) &= a+b*x+c*x^2;
>sol &= solve([p(-h)=y0,p(0)=y1,diffat(p(x),x=h)=yd2],[a,b,c]);
>function ps(x) &= p(x) with sol[1];
>function ystep(y0,y1,yd2,h) &= ps(h)|ratsimp
                                2 h yd2 + 4 y1 - y0
                                -------------------
                                         3
```

Die Implementation in EMT ergibt schließlich ein Verfahren der Ordnung 2. Wir verwenden hier Maxima, um den Ausdruck `ystep` direkt in den Code von `bdf` einzufügen.

```
>function bdf (f,a,b,n,y0) ...
$  x=linspace(a,b,n);
$  y=zeros(size(x));
$  y[1:2]=runge(f,x[1:2],y0);
$  h=(b-a)/n;
$  for i=2 to n;
$  y[i+1]=y[i]+f(x[i],y[i])*h;
$  repeat
$    yold=y[i+1];
$    yd2=f(x[i+1],yold);
$    y[i+1]=&:ystep(y[i-1],y[i],yd2,h);
$    until yold~=y[i+1];
$  end;
$  end
$  return y;
$endfunction
>error1000=bdf("-2*x*y",0,1,1000,1)[-1]-1/E
 4.9057824697e-007
>error2000=bdf("-2*x*y",0,1,2000,1)[-1]-1/E
 1.22635597177e-007
>log(error1000/error2000)/log(2)
 2.00010545601
```

8.9 Schrittweitensteuerung

Zur Schrittweitensteuerung benötigt man in jedem Schritt eine Abschätzung des lokalen Fehlers E_k. Man wählt dann $h_k = (x_{k+1} - x_k)$ so, dass

$$E_k \leq \epsilon h_k$$

wird, wobei $\epsilon > 0$ eine vorgewählte Genauigkeit ist. Für den globalen Fehler gilt damit

$$E = \sum_k E_k \leq \epsilon \sum_k h_k = \epsilon(b - a).$$

Wird dann E_k zu groß, so wird h_k halbiert, bis E_k klein genug ist, wird es zu klein, so kann man h_k verdoppeln. Als Startpunkt wählt man $h_k = h_{k+1}$. Man nennt solche Verfahren auch adaptive Verfahren.

8.54. Beispiel: Zur Abschätzung von E_k liegt es nahe, Verfahren verschiedener Ordnung miteinander zu verbinden, um Rechenaufwand zu sparen. Beispielsweise kann man die verbesserte Streckenzugmethode mit lokalem Diskretisierungsfehler $O(h^3)$

$$\begin{aligned} k_1 &= f(x_k, y_k) \\ k_2 &= f(x_k + \frac{h}{2}, y_k + \frac{h}{2}k_1) \\ \tilde{y}_{k+1} &= y_k + hk_2 \end{aligned}$$

zu einer Runge-Kutta-Methode mit lokalen Fehler $O(h^4)$ ausbauen, indem man

$$\begin{aligned} k_3 &= f(x_k + h, y_k - hk_1 + 2hk_2) \\ y_{k+1} &= y_k + \frac{h}{6}(k_1 + 4k_2 + k_3) \end{aligned}$$

setzt. Den Wert $\|\tilde{y}_{k+1} - y_{k+1}\|$ kann man als Abschätzung für ϵ_k verwenden und damit die Schrittweite steuern.

Auf diese Art ist es auch möglich, eine Runge-Kutte-Methode der globalen Konvergenzordnung 4 mit Schrittweitensteuerung zu bilden.

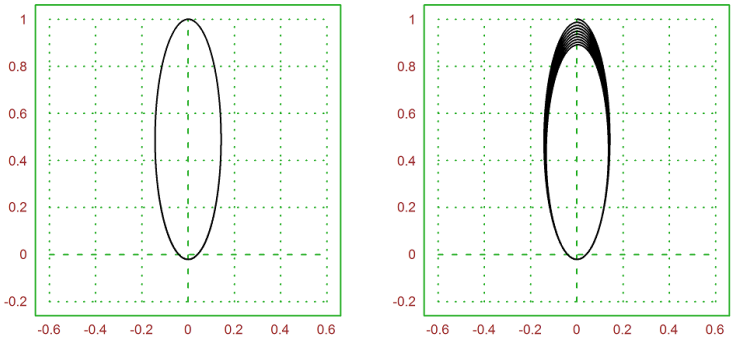

Abbildung 8.7: Rungeverfahren im Vergleich

8.55. Beispiel: Mit den in EMT eingebautem Runge- und adaptiven Runge-Verfahren lösen wir die Differentialgleichung des Zweikörperproblems, bei dem sich ein Planet um ein fixiertes Zentralgestirn bewegt. Die Newton-Gleichung für die Beschleunigung lautet bekanntlich

$$\|y''\| = \frac{-c}{\|y\|^2},$$

wenn das gesamte Massezentrum, also das Zentrum der Gravitation, in 0 angenommen wird, und $c > 0$ eine Konstante ist, die von den Massen und der Gravitationskonstanten abhängt. Dies führt zur Differentialgleichung zweiter Ordnung

$$y''(t) = \frac{-1}{\|y\|^3} y(t).$$

Übersetzt in ein System von Differentialgleichungen erster Ordnung mit den Funktionen (y_1, y_1', y_2, y_2') erhält man

$$\begin{aligned} u_1'(t) &= u_2(t) \\ u_2'(t) &= \frac{-cu_1(t)}{(u_1(t)^2+u_3(t))^{2/3}} \\ u_3'(t) &= u_4(t) \\ u_4'(t) &= \frac{-cu_3(t)}{(u_1(t)^2+u_3(t))^{2/3}} \end{aligned}$$

Abbildung 8.7 zeigt das normale Rungeverfahren. Es wurde $c = 1$ verwendet, sowie die Anfangsbedingungen

$$y(0) = (0,1), \quad y'(0) = (v_0, 0), \quad v_0 = 1, 0.8, 0.6, 0.4, 0.2.$$

Die Bahnen des Planeten werden immer exzentrischer. In der Theorie sind diese Bahnen bekanntlich Ellipsen mit einem Brennpunkt in 0. Wenn der Planet dem Massenschwerpunkt nahe kommt, wird die Lösung allerdings so schlecht, dass sie unbrauchbar wird. Das adaptive Runge-Verfahren vermeidet dieses Problem.

Das folgende EMT-Programm zeigt, dass das adaptive Verfahren sogar mehrere Umläufe des Planeten stabil übersteht.

```
>function f(x,y) ...
$  r=sqrt(y[1]^2+y[3]^2);
$  return [y[2],-y[1]/r^3,y[4],-y[3]/r^3];
$endfunction
>figure(2,2);
>t=0:0.001:20;
>y=adaptiverunge("f",t,[0,0.2,1,0]);
>figure(1); plot2d(y[1],y[3],a=-0.6,b=0.6,c=-0.2,d=1);
>y=runge("f",t,[0,0.2,1,0]);
>figure(2); plot2d(y[1],y[3],a=-0.6,b=0.6,c=-0.2,d=1);
```

8.10 Stabilität

Stabilität bei der numerischen Lösung von Differentialgleichungen hat viele Facetten. Zum einen kann das Problem heikel sein, zum anderen aber auch das Verfahren, obwohl das Problem eine stabile Lösung besitzt.

Man muss sich darüber im Klaren sein, dass manche Differentialgleichungen instabil sind. Sie verhalten sich wie schlecht gestellte Probleme und hängen sensibel von den Anfangswerten ab.

8.56. Beispiel: Das Anfangswertproblem der Form

$$y'(x) = \lambda(y(x) - f(x)) + f'(x), \quad y(x_0) = y_0$$

hat die allgemeine Lösung

$$y(x) = (y_0 - f(x_0))e^{\lambda(x-x_0)} + f(x).$$

Für den speziellen Anfangswert $y(x_0) = y_0 = f(x_0)$ ergibt sich die Lösung $f(x)$. Für nur leicht geänderte Anfangswerte weicht die Lösung allerdings exponentiell davon ab.

Mit Hilfe des folgenden EMT-Programms wurde die Lösung des Anfangswertproblems

$$y'(x) = 10\left(y(x) - \frac{x^2}{1+x^2}\right) + \frac{2x}{(1+x^2)^2}, \quad y(0) = 0$$

berechnet und mit der richtigen Lösung verglichen.

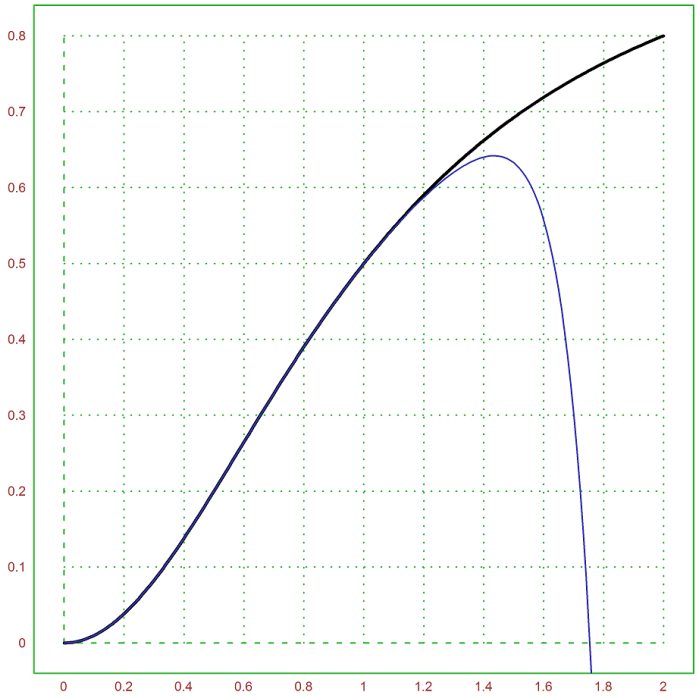

Abbildung 8.8: Instabiles Anfangswertproblem

```
>function f(x,y) ...
$  r=1+x^2;
$  return 10*(y-x^2/r)+2*x/r^2;
$endfunction
>t=0:0.01:2;
>y=runge("f",t,0);
>plot2d(t,t^2/(1+t^2),thickness=2);
>plot2d(t,y,color=blue,>add);
```

In Abbildung 8.8 kann man erkennen, wie sehr die Lösung von der tatsächlichen Lösung abweicht. Man beachte, dass nicht nur der ungenau im Rechner repräsentierte Anfangswert, sondern auch die Fehler in jedem Schritt zu diesem Verhalten beitragen.

8.57 Aufgabe: Finden Sie alle Lösungen der Differentialgleichung

$$y'' = 100y$$

zum Beispiel durch den Ansatz

$$y(x) = e^{\lambda x}.$$

Lösen Sie das Anfangswertproblem $y(0) = 0, y'(0) = -10$ mit EMT auf dem Intervall $[0, 4]$. Verwenden Sie zum Beispiel `runge`, `lsoda` oder auch `adaptiverunge`. Erklären Sie den Fehler.

Es ist aber auch möglich, dass das Problem eigentlich gut gestellt ist, aber die Lösung in Regionen kommt, in denen der weitere Verlauf instabil ist, oder nicht stabil mit dem gewählten Verfahren gelöst werden kann. Es wären zur Lösung dann sehr kleine Schrittweiten notwendig.

8.58. Beispiel: Bei Differentialgleichungen wie

$$y' = \frac{-y}{y+c}, \quad y(0) = -1$$

für sehr kleine c verhält sich die Lösung zunächst wie $y = 1 - x$. In der Nähe von $y = 0$ gibt es aber eine qualitative Änderung der Differentialgleichung. Solche Gleichungen nennt man in diesen Regionen steife Differentialgleichungen. Nicht-adaptive Verfahren überspringen einfach die Region der Steifheit. Adaptive Verfahren müssen in solchen Regionen sehr kleine Schrittweiten wählen.

```
>x=1:0.01:4;
>c=1e-5; y=runge("-y/(c+y)",x,1);
>plot2d(x,y);
```

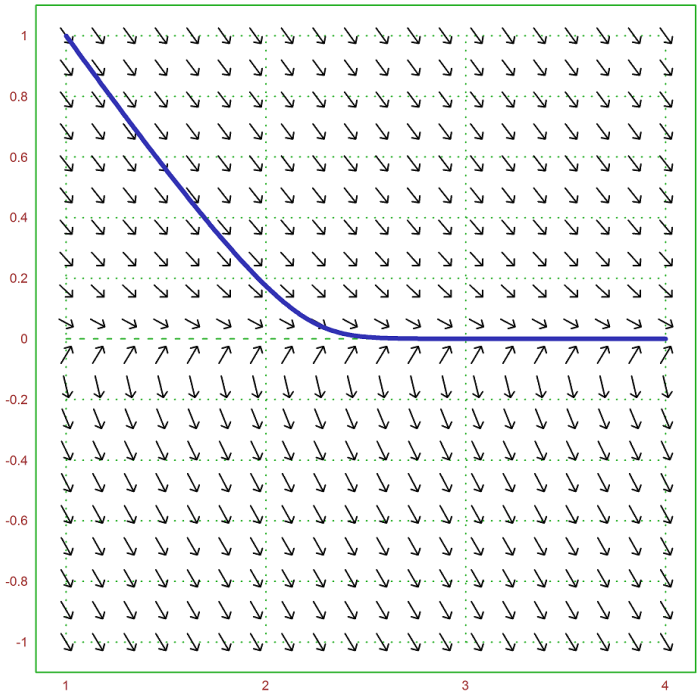

Abbildung 8.9: Steife Differentialgleichung

Die Lösung, die dieses EMT-Programm auswirft ist praktisch $y = 1 - x$. Wir können die Probleme für $c = 0.1$ sichtbar machen (siehe Abbildung 8.9). Es ist klar, dass mit Schrittweiten $h > 4c$ der Bereich, in dem die Änderung stattfindet, nicht gefunden wird.

```
>c=0.1; vectorfield("-y/(c+y)",1,4,-1,1);
>plot2d(x,runge("-y/(c+y)",x,1),add=1,color=blue,thickness=3);
```

Die korrekte Lösung wird mit Maxima nur implizit angegeben. Sie lässt sich aber als Funktion $x = y(x)$ zeichnen.

```
>&ode2('diff(y,x)=-y/(c+y),y,x)
```

$$- c \log(y) - y = x + \%c$$

```
>y=epsilon:0.01:1; x=-c*log(y)-y;
>plot2d(x+2,y,a=1,b=4,c=-1,d=1,color=blue,thickness=3);
```

8.59. Definition: Für eine exaktere Diskussion des Problems betrachten wir das Problem

$$y' = \lambda y$$

mit $\lambda \in \mathbb{C}$. Wir sagen, dass ein Verfahren für λ und h A-stabil ist, wenn es dieses Problem mit konstanter Schrittweite h so löst, das

$$\lim_{n\to\infty} y_n = 0$$

ist.

Man beachte, dass das für $\mathrm{Re}(\lambda) < 0$ das korrekte Verhalten ist. Die Idee hinter der Definition ist, dass λ der Lipschitzkonstante entspricht, und exponentielle Anteile eine bestimmte Lösung auch dann nicht stören sollen, wenn die Schrittweite h groß ist. Für Systeme entspricht λ einem Eigenwert der lokal linearisierten Form der Differentialgleichung. Wir können darauf hier nicht weiter eingehen.

8.60 Aufgabe: Zeigen Sie, dass das explizite Eulersche Streckenzugverfahren nur stabil ist, wenn $|1 + \lambda h| < 1$ gilt, und dass das implizite Streckenzugverfahren für alle λ mit $\mathrm{Re}(\lambda) < 0$ stabil ist.

8.61 Satz: *Das zweistufige BDF-Verfahren aus Beispiel 53 ist A-stabil.*

Beweis: Man kann im Fall $y' = \lambda y$ die Rekursion exakt ausrechnen. Dabei setzen wir die dortigen Berechnungen in EMT fort.

```
>&solve(ystep(y0,y1,lambda*y2,h)=y2,y2)
```

$$[y2 = \frac{y0 - 4\ y1}{2\ h\ lambda - 3}]$$

In der Tat führt diese Differenzengleichung zum Ziel. Wir probieren $\lambda = -5$ und 1000 Schritte aus.

```
>lambda=-5; n=1000; h=1/n;
>sequence("(x[n-2]-4*x[n-1])/(2*h*lambda-3)",[1,exp(-lambda/n)],n+1)[-1]
 0.00683957874862
>exp(-5)
 0.00673794699909
```

Es stellt sich heraus, dass die Rekursion auch bei anderen Startwerten, etwa $(0,1)$ oder $(1,0)$ stabil gegen 0 geht. Zur Untersuchung dieser Stabilität verwenden wir, dass sich Differenzengleichung mit dem allgemeinen Ansatz $y_n = t^n$ lösen lassen. Man erhält eine polynomiale Gleichung der Form

$$t^2 = \frac{1-4t}{2h\lambda - 3}.$$

Wenn t_1 und t_2 die Lösungen dieser Gleichung sind, so ist

$$y_n = \alpha_1 t_1^n + \alpha_2 t_2^n$$

(außer im Sonderfall $t_1 = t_2$). In jedem Fall ist $|t| < 1$ für die Stabilität notwendig für alle Nullstellen der Polynomgleichung. Mit $z = 1/t$ erhalten wir die Gleichung

$$2h\lambda = 3 - 4z + z^2 = q(z)$$

Da wir $\mathrm{Re}(h\lambda) < 0$ überprüfen wollen, muss $\mathrm{Re}(q(z)) \geq 0$ für alle $|z| \leq 1$ gelten. Genau dann nämlich hat unsere Originalgleichung keine Nullstelle $|t| \geq 1$. Dies überprüfen wir durch einen Plot in EMT, indem wir das Bild des Einheitskreises plotten. Es befindet sich ganz in der rechten Halbebene.

```
>function q(z) &= 3-4*z+z^2

                                2
                               z  - 4 z + 3

>z=exp(I*linspace(0,2pi,1000)); plot2d(q(z),r=10);
```

Für einen exakten Beweis muss man lediglich

$$p(x) = \mathrm{Re}(q(\cos(x) + i\,\sin(x))) = 2\cos(x)^2 - 4\cos(x) + 2$$

untersuchen. Das Polynom $2y^2 - 4y + 2$ ist aber auf $[-1,1]$ in der Tat nicht negativ. **q.e.d.**

8.62. Beispiel: Das BDF-Verfahren mit drei Schritten ist nicht mehr A-stabil. Allerdings sind die Bereiche, in denen es stabil ist, sehr groß. Der wichtige Fall $y' = iy$ führt jedoch erstaunlicherweise zu Instabilitäten für zu kleine h (siehe Abbilding 8.10).

```
>function p(x) &= a+b*x+c*x^2+d*x^3;
>sol &= solve([p(-2*h)=y0,p(-h)=y1,p(0)=y2,diffat(p(x),x=h)=yd3],[a,b,c,d]);
>function ps(x) &= p(x) with sol[1];
>function ystep(y0,y1,y2,yd3,h) &= ps(h)|ratsimp;
>sol &= solve(ystep(y0,y1,y2,lambda*y3,h)=y3,y3)

                          - 18 y2 + 9 y1 - 2 y0
                  [y3 = ----------------------]
                           6 h lambda - 11

>function q(z) &= 11+(num(rhs(sol[1])) with [y0=z^3,y1=z^2,y2=z])
```

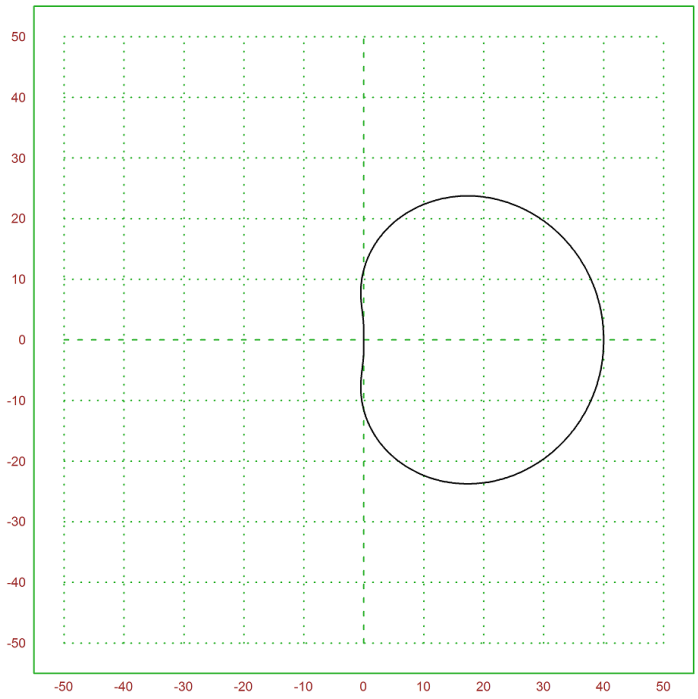

Abbildung 8.10: Bild der $q(z)$ mit $|z| = 1$

```
                3       2
  - 2 z  + 9 z  - 18 z + 11
```

```
>plot2d(q(z),r=50);
```

8.63. Beispiel: Als weiteres Beispiel zeigen wir, dass nahe liegende Algorithmen hinsichtlich der Stabilität völlig unbrauchbar sein können. Wir verwenden die Hermite-Interpolation

$$p(x_{n-1}) = y_{n-1}, \quad p'(x_{n-1}) = f(x_{n-1}, y_{n-1})$$
$$p(x_n) = y_n, \quad p'(x_n) = f(x_n, y_n)$$

und setzen

$$y_{n+1} = p(x_{n+1}).$$

Die Herleitung der Rekursionsformel ergibt eine recht einfache Formel.

```
>function p(x) &= a+b*x+c*x^2+d*x^3
>sol &= solve( ...
>  [p(-h)=y0,diffat(p(x),x=-h)=yd0,p(0)=y1,diffat(p(x),x=0)=yd1],[a,b,c,d])
>function ystep(y0,yd0,y1,yd1,h) &= p(h) with sol[1]
```

```
        h (2 yd1 + yd0) + h (yd1 + yd0) + h yd1 - 4 y1 + 5 y0
```

Jedoch zeigt ein einfaches Experiment, dass die Formel völlig unbrauchbar ist. Um das genauer zu untersuchen, lösen wir $y' = \lambda y$. Die Rekursion vereinfacht sich dann, insbesondere wenn wir $c = \lambda h$ setzen.

```
>y2 &= ystep(y0,lambda*y0,y1,lambda*y1,h)

         h (2 y1 lambda + y0 lambda) + h (y1 lambda + y0 lambda)
                                            + h y1 lambda - 4 y1 + 5 y0

 Set h*lambda=c.
>y2c &= ratsimp(y2 with lambda=c/h)

                      (4 c - 4) y1 + (2 c + 5) y0
```

Zur Untersuchung dieser Rekursion setzen wir wieder $y_n = t^n$.

```
>&c with solve(y2n-y2c,c),
>function h(t) &= % with [y2n=t^2,y1=t,y0=1]
                               2
                              t  + 4 t - 5
                              ------------
                                4 t + 2

>z=exp(I*linspace(0,2pi,1000)); ...
>plot2d(h(z),r=6); insimg;
```

Wir erhalten die Gleichung

$$c = \frac{t^2 + 4t - 5}{4t + 2},$$

die keine Lösung $|t| > 0$ haben sollte. Allerdings zeigt der Plot, dass dies für kein $c < 0$ der Fall ist.

8.64 Aufgabe: Versuchen Sie, $y' = -2xy$, $y(0) = 1$ mit diesem Verfahren auf $[0, 1]$ zu lösen.

8.11 LSODA-Algorithmus

Numerische Probleme werden in durch sorgfältig ausgewogene Schrittweitensteuerungen unter Verwendung von impliziten Verfahren, die für steife Differentialgleichungen geeigneter sind gut gelöst. EMT verwendet dazu `lsoda`, das auf einem Verfahren LSODA von Petzold und Hindmarsh mit einer C-Implementation von Heng Li beruht.

LSODA verwendet zur Lösung von impliziten Methoden zudem Quasi-Newton-Verfahren. Dadurch wird das Verfahren sehr effektiv.

8.65. Beispiel: Wir testen das Verfahren am Beispiel 58.

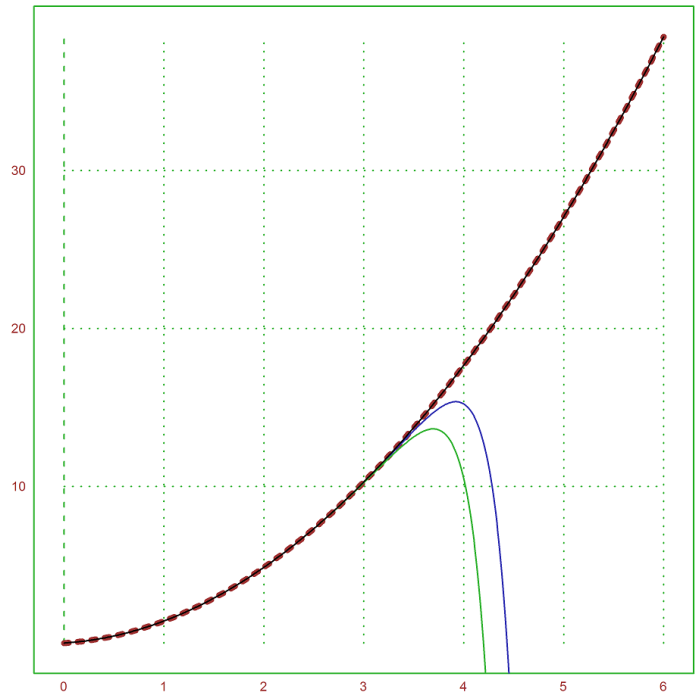

Abbildung 8.11: Vergleich verschiedener Verfahren

```
>c=1e-5; x=1:0.1:4; y=lsoda("-y/(c+y)",x,1,epsilon);
>plot2d(x,y);
```

8.66. Beispiel: Die allgemeine Lösung der Differentialgleichung

$$y' = 5(y - x^2)$$

hat einen exponentiellen Anteil, der beim Anfangswert $y(0) = 2/25$ nicht zum Tragen kommt.

```
>function f(x,y) := 5*(y-x^2)
>sol &= ode2('diff(y,x)=5*(y-x^2),y,x);
>function fsol(x,%c) &= expand(y with sol)

                          5 x    2   2 x   2
                       %c E    + x  + --- + --
                                      5    25
>vectorfield("f",0,5,0,5);
>c=-20; plot2d("fsol(x,0)",add=1,color=red,thickness=2);
```

In der Tat ist `lsoda` in der Lage, hier wesentlich länger der korrekten Lösung zu folgen, als dies bei anderen Verfahren der Fall ist.

```
>plot2d("fsol(x,0)",0,6,color=red,thickness=2,style="--");
>x=0:0.01:6; y=runge("f",x,2/25);
>plot2d(x,y,>add,color=blue);
>plot2d(x,adaptiverunge("f",x,2/25),>add,color=green);
>plot2d(x,lsoda("f",x,2/25),>add,color=black);
```

8.67 Aufgabe: Verwenden Sie den BDF-Algorithmus aus Beispiel 53 für dieses Problem, und zeigen Sie, dass dieser Algorithmus der Lösung länger folgt als das Runge-Verfahren, obwohl er nur die Ordnung 2 hat.

8.68 Aufgabe: Testen Sie als weiteres extremes Beispiel die Differentialgleichung

$$y' = y^2(1-y), \quad y(0) = \delta$$

auf dem Intervall $[0, 2/\delta]$ mit 1000 Zwischenschritten. Zählen Sie die Anzahl der Funktionsausrufe für `adaptiverunge` und für `lsoda`.

8.69. Beispiel: Es sollte jedoch bemerkt werden, dass schlechte Differentialgleichungen bisweilen nur durch schlechte Modellierung zustande kommen. In der Biologie simuliert die Schwelle $y = -c$ einfach nur das Absterben eines Zweiges von y, zum Beispiel durch Aussterben einer Art. Man sollte statt dessen lieber $f(x, y) = 0$ für $y < 0$ setzen.

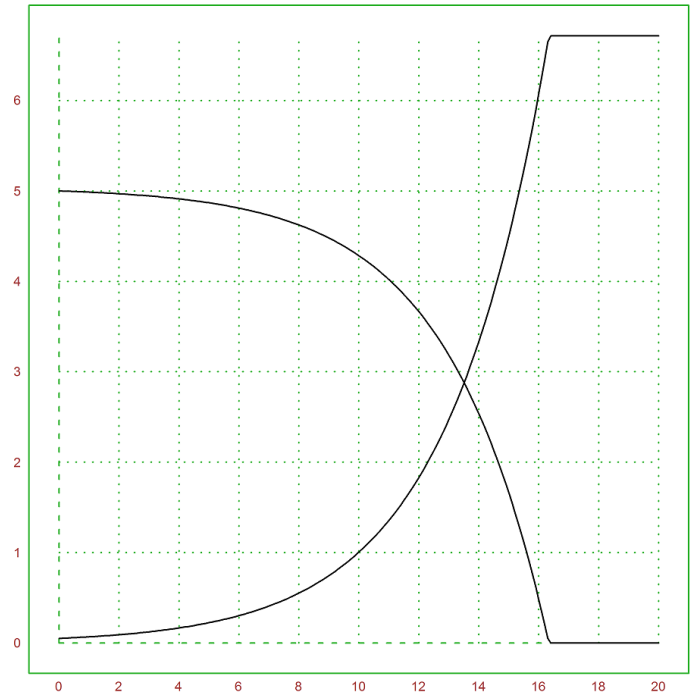

Abbildung 8.12: Biochemische Reaktion

Das folgende Beispiel stammt aus dem Buch von Cutlip und Shacham über Chemical and

Biochemical Engineering.

$$\frac{dB}{dt} = \frac{kBS}{K+S},$$
$$\frac{dS}{dt} = \frac{-0.75\,kBS}{K+S}.$$

Es tritt im Prinzip dasselbe Problem wie oben auf. Das Programm `lsoda` löst es, im Gegensatz zu anderen Verfahren. Aber es ist viel einfacher, den Extremfall $S < 0$ durch 0 abzufangen (siehe Abbildung 8.12). Sobald er eintritt, ändert sich S nicht mehr.

```
>k=0.3; K=1e-6;
>B0=0.05; S0=5; y0=[B0,S0];
>t0=0; tf=20; t=t0:0.1:tf;
>function dY (t,y) ...
$  global k,K
$  dBdt=k*y[1]*y[2]/(K+y[2]);
$  dSdt=min(-0.75*k*y[1]*y[2]/(K+y[2]),0);
$  return [dBdt,dSdt];
$endfunction
>y=adaptiverunge("dY",t,y0);
>plot2d(t,y);
```

8.12 Randwertprobleme

Zweipunkt-Randwertprobleme sind Differentialgleichungen der Form

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

bei denen Randwerte

$$y(a) = y_a, \quad y(b) = y_b$$

vorgegeben sind.

8.70. Beispiel: Das Randwertproblem

$$y''(t) = -y(t), \quad y(0) = 0, \quad y(\pi) = 0$$

hat keine eindeutige Lösung. Es sind nämlich alle Funktionen der Form

$$y(t) = c\sin(t)$$

Lösungen. Ein Fundamentalsystem für die Differentialgleichung sind zunächst die Funktionen

$$y(t) = c\sin(t) + d\cos(t).$$

Aus $y(0) = 0$ folgt $d = 0$. Das Randwertproblem

$$y''(t) = -y(t), \quad y(0) = 0, \quad y(\pi) = 1$$

hat daher gar keine Lösung. Das Randwertproblem

$$y''(t) = -y(t), \quad y(0) = 0, \quad y(\pi/2) = 1$$

die eindeutige Lösung $y(t) = \sin(t)$.

Als Rechenverfahren bietet sich an, das Anfangswertproblem

$$y_u''(t) = f(t, y_u(t), y_u'(t)), \quad y_u(a) = y_a, \quad y_u'(a) = u$$

mit Hilfe der Verfahren dieses Kapitels zu lösen, und dann die Lösung der Gleichung

$$\phi(u) := y_u(b) = y_b$$

zu suchen. Dieses Verfahren nennt man Schießverfahren.

8.71. Beispiel: Wir behandeln das Randwertproblem

$$y'' + y = \sin(x), \quad y(0) = 0, \quad y\left(\frac{\pi}{2}\right) = -1$$

mit EMT. In der Tat kann für dieses Problem in Maxima eine exakte Lösung berechnet werden.

```
>ysol &= ode2('diff(y,x,2)+y=sin(x),y,x)

                                  x cos(x)
                y = %k1 sin(x) - -------- + %k2 cos(x)
                                    2

>function f(x) &= y with bc2(ysol,x=0,y=0,x=%pi/2,y=-1)

                                    x cos(x)
                          - sin(x) - --------
                                      2

>plot2d(f,0,pi/2);
```

Wir lösen das Problem numerisch. Dazu verwenden wir `runge`, um ein Schießverfahren aufzusetzen und dann zu lösen. Die Differentialgleichung muss zunächst in ein System von Differentialgleichungen umgeschrieben werden. Die Funktion `runge` liefert dann eine Matrix mit den Zeilen y und y' zurück. Wir sind nur am letzten Wert der ersten Zeile interessiert, den wir gleich -1 machen wollen.

```
>function f(x,y) := [y[2],sin(x)-y[1]]
>t=linspace(0,pi/2,1000);
>function h(yd) := runge("f",t,[0,yd])[1,-1]
>ydsol := solve("h",-1,y=-1)
 -1.5
>plot2d(t,runge("f",t,[0,ydsol])[1]);
```

Kapitel 9

Eigenwerte

9.1 Das charakteristische Polynom

Zur Berechnung der Eigenwerte werden wir ehen nicht das charakteristische Polynom verwenden, weil die Nullstellen dieses Polynoms nicht stabil von den Koeffizienten abhängen. Zur Vereinfachung kann man aber die Matrix zunächst auf Tridiagonalgestalt bringen.

Sei A dazu symmetrisch. Wir verwenden Givens-Rotationen oder Householder-Matrizen, um

$$H_1^T A H_1 = \left(\begin{array}{c|ccccc} a_{1,1} & a_{1,2} & 0 & \dots & 0 \\ \hline a_{2,1} & & & & \\ 0 & & & & \\ \vdots & & & * & \\ 0 & & & & \end{array}\right)$$

(mit $a_{1,2} = a_{2,1}$) zu erreichen. Dies kann man mit orthogonalen Matrizen

$$H_i = \begin{pmatrix} I_i & 0 \\ 0 & \tilde{H}_i \end{pmatrix}$$

fortsetzen, so dass mit

$$H = H_1 \cdot \ldots \cdot H_{n-2}$$

die Matrix A in Tridiagonalgestalt gebracht wird. Also

$$H^T A H = \begin{pmatrix} * & * & & & 0 \\ * & * & * & & \\ & \ddots & \ddots & \ddots & \\ & & * & * & * \\ 0 & & & * & * \end{pmatrix} := T.$$

Weil H orthogonal ist, ist A ähnlich zu T und hat dasselbe charakteristische Polynom und dieselben Eigenwerte.

Wenn nötig, kann man nun das charakteristische Polynom von T durch Rekursion berechnen. Sei dazu

$$T_n = \left(\begin{array}{c|ccc} a_{1,1} & a_{1,2} & 0 & \dots & 0 \\ \hline a_{2,1} & & & & \\ 0 & & & & \\ \vdots & & & T_{n-1} & \\ 0 & & & & \end{array}\right)$$

eine Tridiagonalmatrix. Durch Entwickeln nach der ersten Spalte folgt

$$\det(T_n - \lambda I_n) = (a_{1,1} - \lambda)\det(T_{n-1} - \lambda I_{n-1}) - a_{2,1}a_{1,2}\det(T_{n-2} - \lambda I_{n-2}).$$

9.1. Beispiel: Man beachte, dass wir hier die Givensrotation $G_{j+1,i}$ benötigen, die $a_{i,j}$ annulliert. Die EMT-Funktion `givensrot` leistet genau das, wobei die Spalte und die beiden Zeilen, die verändert werden dürfen, angegeben werden muss.

```
>A=random(5,2)
      0.918802182236      0.997492478596
       0.47321194607      0.185709276137
      0.421585074083       0.72084285704
      0.247803741164      0.834083494682
      0.443719830232      0.371865996909
>B,Q=givensrot(1,2,3,A,id(5));
>B
      0.918802182236      0.997492478596
     -0.633769296032     -0.618168850565
                   0     -0.414692213552
      0.247803741164      0.834083494682
      0.443719830232      0.371865996909
```

Die Rotationsmatrix wird in Q gespeichert. Wir berechnen daher in jedem Schritt noch

$$\tilde{A} = B \cdot Q' = Q \cdot A \cdot Q'.$$

Die einfachste Implementation sieht daher fogendermaßen aus.

```
>function tridiag (A) ...
$  n=cols(A); M=id(n);
$  for j=1 to n-2;
$    for i=j+2 to n
$      if not A[i,j]~=0 then
$        A,Q = givensrot(j,j+1,i,A,M);
$        A = A.Q';
$      endif;
$    end
$  end
$  return A
$endfunction
```

Test an einem Beispiel.

```
>shortestformat; A=random(5,5); A=A+A'
     0.892      1.01      1.27      1.19      1.17
      1.01     0.837     0.544     0.927      1.35
      1.27     0.544      1.71     0.731      1.05
      1.19     0.927     0.731      0.22     0.945
      1.17      1.35      1.05     0.945      1.12
>T := tridiag(A)
     0.892     -2.33         0         0         0
     -2.33      3.73    -0.654         0         0
         0    -0.654    -0.284    -0.186         0
         0         0    -0.186     0.657    -0.518
         0         0         0    -0.518    -0.215
```

Die Implementation der Rekursion zur Auswertung des charakteristischen Polynoms ist ebenfalls nicht sehr schwer.

```
>function evalcharpoly (x,T) ...
$  n=cols(T);
$  a=T[n,n]-x;
$  b=a*(T[n-1,n-1]-x)-T[n-1,n]*T[n,n-1];
$  for i=n-2 to 1 step -1
$    c=(T[i,i]-x)*b-T[i,i+1]*T[i+1,i]*a;
$    a=b;
$    b=c;
$  end
$  return b;
$endfunction
>longestformat;
>evalcharpoly(1.8,T)
       30.73162114893318
>det(A-1.8*id(5))
       30.73162114893318
>plot2d("evalcharpoly(x,T)",-1,1);
```

Bei nicht symmetrischen Matrizen A entsteht natürlich durch die Rotationen keine Tridiagonalmatrix, aber immer noch eine Hessenberg-Matrix, bei der die zweite und weitere Diagonalen unterhalb der Hauptdiagonalen verschwinden.

9.2 Aufgabe: Sei $A \in \mathbb{K}^{n\times n}$ und $\lambda \in \mathbb{K}$ ein Eigenvektor von A. Zeigen Sie, dass es ein $i \in \{1, \ldots, n\}$ gibt mit

$$|a_{i,i} - \lambda| \leq \sum_{j\neq i} |a_{i,j}|.$$

Geben Sie damit für eine beliebige Matrix $A \in \mathbb{K}^n$ eine obere Schranke für die Beträge der Eigenwerte an (Gerschgorin-Kreise). Zeigen Sie dass alle stark diagonaldominanten, Hermiteschen Matrizen $A \in \mathbb{K}^{n\times n}$ positiv definit sind.

9.3 Aufgabe: Geben Sie eine Rekursionsformel für die Ableitung des charakteristischen Polynoms $\chi_T'(\lambda)$ an, wenn T eine Tridiagonalmatrix ist.

9.4 Aufgabe: Zeigen Sie

$$\det \begin{pmatrix} 2x & 1 & & & & \\ 1 & 2x & 1 & & & \\ & \ddots & \ddots & \ddots & & \\ & & 1 & 2x & 1 & \\ & & & 1 & 2x & \sqrt{2} \\ & & & & \sqrt{2} & 2x \end{pmatrix} = 2T_n(x),$$

wobei T_n das n-te Chebyshevpolynom ist, und die Matrix n Zeilen und Spalten hat.

9.2 Jacobi-Verfahren

Die Tridiagonalisierung von A aus dem vorigen Abschnitt lässt sich nicht mit endlich vielen Schritten fortsetzen, so dass A zu einer ähnlichen Diagonalmatrix wird. Deswegen muss man Iterationsverfahren anwenden, wobei eine Folge von ähnlichen Matrizen erzeugt wird, die gegen eine Diagonalmatrix konvergieren. Eines der verwendeten Verfahren, das sehr allgemein und recht stabil konvergiert, ist das Jacobi-Verfahren.

9.5 Satz: *Sei $A \in \mathbb{R}^{n\times n}$ symmetrisch, und $(A_m)_{m\in\mathbb{N}}$ die Folge von Matrizen, die dadurch entsteht, dass man in jedem Schritt das betragsgrößte Element außerhalb der Diagonalen von A_m mit Givens-Rotationen von rechts und links zu 0 macht, also*

$$A_{m+1} = G_m A_m G_m^T$$

mit Givens-Rotationen G_m, $A_0 = A$. Dann konvergiert die Folge (A_m) gegen eine Diagonalmatrix.

Beweis: Wir bezeichnen mit

$$T_m = \sum_{i\neq j} a_{i,j,m}^2$$

die Summe der quadrierten Nebendiagonalelemente von A_m. Es ist $T_m \to 0$ zu zeigen. Sei $G_m = G_{i,j}(\phi)$. Die Rotation

$$G_{i,j}(\phi)A_m$$

ändert nur die i-te und j-te Spalte von A, und zwar so,

$$a_{i,k,m}^2 + a_{j,k,m}^2$$

dabei für $k = 1,\ldots,n$ unverändert bleibt. Für die nachfolgende Multiplikation mit G_m^T gilt in den Spalten dasselbe. Nach Konstruktionsvorschrift gilt aber im $m+1$-ten Schritt

$$a_{i,j,m+1} = a_{j,i,m+1} = 0.$$

Wir schließen daraus

$$a_{i,i,m+1}^2 + a_{j,j,m+1}^2 = a_{i,i,m}^2 + 2a_{i,j,m}^2 + a_{j,j,m}^2.$$

Also

$$T_{m+1} = T_m - 2a_{i,j,m}^2.$$

Weil $|a_{i,j,n}|$ maximal gewählt wird, ist

$$T_m \leq (n^2 - n)a_{i,j,m}^2,$$

also

$$a_{i,j,m}^2 \geq \frac{1}{n^2-n} T_m.$$

Man erhält

$$T_{m+1} \leq \left(1 - \frac{2}{n^2-n}\right) T_m.$$

Die Konvergenz folgt also aus

$$\left(1 - \frac{2}{n^2 - n}\right) < 1.$$

q.e.d.

Die Konvergenz ist in Wirklichkeit besser, als man hier den Eindruck hat. Es stellt sich in der letzten Phase des Algorithmus eine quadratische Konvergenz ein.

Die Frage ist, wie man Givens-Rotationen berechnet, so dass $a_{i,j,m+1} = 0$ wird. Es muss gelten

$$\begin{pmatrix} a_{i,i,m+1} & 0 \\ 0 & a_{j,j,m+1} \end{pmatrix} = G_m \begin{pmatrix} a_{i,i,m+1} & a_{i,j,m+1} \\ a_{i,j,m+1} & a_{j,j,m+1} \end{pmatrix} G_m^T.$$

Man hat also das gewöhnliche Diagonalisierungsproblem einer symmetrischen 2×2-Matrix vorliegen. Bekanntlich ist diese Diagonalisierung bei symmetrischen Matrizen immer mit einer orthogonalen Matrix möglich. Dies beweist die Existenz der Matrix G_m.

Die Elemente $a_{i,i,m+1}$ und $a_{j,j,m+1}$ sind die Eigenvektoren der Matrix

$$H = \begin{pmatrix} a_{i,i,m} & a_{i,j,m} \\ a_{i,j,m} & a_{j,j,m} \end{pmatrix}.$$

Offenbar ist dann

$$v_1 = \begin{pmatrix} -a_{i,j,m} \\ a_{i,j,m} - a_{i,j,m+1} \end{pmatrix}$$

Ein Eigenvektor zu $a_{i,j,m+1}$. Die erste Spalte von G_m^T ist daher $v_1/\|v_1\|$. Die zweite Spalte ist ein normierter, auf v_1 senkrecht stehender Vektor.

Es erweist sich allerdings, das das folgende Verfahren etwas stabiler und effizienter ist. Berechnet man $a_{i,j,M+1}$ bei einem Drehwinkel ϕ, so sieht man, dass

$$(a_{i,i,m} - a_{j,j,m}) \cos\phi \sin\phi + a_{i,j,m}(\cos^2\phi - \sin^2\phi) = 0$$

gelten muss. Wegen

$$\cos(2\phi) = \cos^2\phi - \sin^2\phi, \quad \sin(2\phi) = 2\cos\phi\sin\phi$$

hat man

$$\frac{\cos^2\phi - \sin^2\phi}{2\cos\phi\sin\phi} = \cot(2\phi) = \frac{a_{i,i,m} - a_{j,j,m}}{2a_{i,j,m}} := \theta.$$

Setzt man $t = \tan\phi$, so erhält man

$$\frac{1 - t^2}{2t} = \theta$$

oder

$$t^2 - 2\theta t - 1 = 0.$$

mit den Lösungen

$$t_{1,2} = -\theta \pm \sqrt{\theta^2 + 1} = \frac{1}{\theta \pm \sqrt{\theta^2 + 1}}.$$

Wir setzen

$$t = \begin{cases} \dfrac{1}{\theta + \operatorname{sign}\theta\sqrt{\theta^2+1}} & \theta \neq 0. \\ 1 & \theta = 0. \end{cases}$$

Es ergibt sich schließlich

$$\cos\phi = \frac{1}{\sqrt{1+t^2}}, \quad \sin\phi = t\cos\phi.$$

Damit berechnet man

$$a_{i,i,m+1} = a_{i,i,m} - a_{i,j,m}\tan\phi, \quad a_{j,j,m+1} = a_{j,j,m} + a_{i,j,m}\tan\phi.$$

Auch die beiden Rotationen lassen sich mit

$$r := \frac{\sin\phi}{1+\cos\phi}$$

vereinfachen zu

$$\begin{aligned} a_{i,k,m+1} &= a_{i,k,m} - \sin\phi(a_{j,k,m} + r a_{i,k,m}) \\ a_{j,k,m+1} &= a_{j,k,m} + \sin\phi(a_{i,k,m} - r a_{j,k,m}) \end{aligned}$$

und

$$\begin{aligned} a_{k,i,m+1} &= a_{k,i,m} - \sin\phi(a_{k,j,m} + r a_{k,i,m}) \\ a_{k,j,m+1} &= a_{k,i,m} + \sin\phi(a_{k,i,m} - r a_{k,j,m}) \end{aligned}$$

für $k \neq i, j$.

9.6. Beispiel: Das Verfahren ist in EMT in der Funktion `jacobi` implementiert. Wir testen es am Beispiel der Matrix

$$A = \det \begin{pmatrix} 2 & 1 & & & & & \\ 1 & 2 & 1 & & & & \\ & \ddots & \ddots & \ddots & & & \\ & & 1 & 2 & 1 & & \\ & & & 1 & 2 & \sqrt{2} & \\ & & & & \sqrt{2} & 2 & \end{pmatrix}$$

deren Eigenwerte wir mit Aufgabe 4 berechnen können.

```
>n=20;
>A=setdiag(setdiag(2*id(n),1,1),-1,1);
>A[n-1,n]=sqrt(2); A[n,n-1]=sqrt(2);
>max((sort(jacobi(A)))-chebzeros(0,4,20))
 0
```

Man beachte, dass `sort` eine zusätzliche Klammer benötigt, da es in EMT zwei Werte zurück gibt. Alternativ kann man das Ergebnis zunächst einer Variablen zuweisen.

9.7 Aufgabe: Beweisen Sie, dass die Eigenwerte von A tatsächlich die Nullstellen des Chebyshev-Polynoms auf $[0, 4]$ sind.

Diese Gleichungen wurden im folgenden Java-Programm umgesetzt. Man kann allerdings darüber hinaus noch berücksichtigen, dass die Matrix symmetrisch ist, und daher immer nur der untere Teil neu berechnet werden muss. Um die Übersichtlichkeit nicht zu gefährden, wurde dies hier nicht implementiert.

```
public static double[] computeJacobi (double A[][])
   int n=A.length;
    int count=0;
    while (true)
       double max=0;
        int p=0,q=0;
        for (int i=1; i<n; i++)
            for (int j=0; j<i; j++)
               double h=Math.abs(A[i][j]);
                if (h>max)
                   max=h;
                    p=i; q=j;


        if (max<epsilon) break;
        double theta=(A[q][q]-A[p][p])/(2*A[p][q]);
        double t=1;
        if (theta>0) t=1/(theta+Math.sqrt(theta*theta+1));
        else t=1/(theta-Math.sqrt(theta*theta+1));
        double c=1/Math.sqrt(1+t*t);
        double s=t*c;
        A[p][p]=A[p][p]-A[q][p]*t;
        A[q][q]=A[q][q]+A[q][p]*t;
        A[p][q]=A[q][p]=0;
        double r=s/(1+c);
        for (int j=0; j<n; j++)
           if (j==p || j==q) continue;
            double h=A[p][j]-s*(A[q][j]+r*A[p][j]);
            A[q][j]=A[q][j]+s*(A[p][j]-r*A[q][j]);
            A[p][j]=h;

        for (int i=0; i<n; i++)
           if (i==p || i==q) continue;
            double h=A[i][p]-s*(A[i][q]+r*A[i][p]);
            A[i][q]=A[i][q]+s*(A[i][p]-r*A[i][q]);
            A[i][p]=h;

        count++;
        if (count>100*n*n*n)
            throw new RuntimeException("Iteration failed!");

    double d[]=new double[n];
    for (int i=0; i<n; i++) d[i]=A[i][i];
    return d;
```

Es stellt sich heraus, dass die Suche nach dem maximalen Eintrag in A_m nicht notwendig ist. Ein Verfahren, dass einfach alle Einträge unterhalb der Diagonalen zyklisch durchläuft, ist genauso gut. Wir können hier nur die Konvergenz beweisen.

9.8 Satz: *Das Jacobi-Verfahren konvergiert auch, wenn die Einträge unterhalb der Diagonalen zyklisch zu* 0 *gemacht werden.*

Beweis: Sei $B_0 = A$ die Ausgangsmatrix und B_{m+1} die Matrix, die nach Durchlaufen eines vollständigen Zyklus aus B_m entsteht. Sei $s_m = \Sigma(B_m)$ die Summe der quadrierten Nebendiagonalelemente von B_m. Die Folge $(s_m)_{m\in\mathbb{N}}$ ist streng monoton fallend. Angenommen, sie konvergiert gegen $s > 0$.

Aufgrund der obigen Formeln ist die Abbildung

$$B_m \mapsto B_{m+1}$$

stetig. Wir bezeichnen diese Abbildung mit ϕ. Da die Quadratsummen der Elemente der Matrizen B_m beschränkt sind, hat die Folge einen Häufungspunkt

$$B = \lim_{\nu\to\infty} B_{m_\nu}.$$

Es muss gelten $\Sigma(B) = s$, da Σ stetig ist und $s > 0$. Außerdem

$$\Sigma(\phi(B)) < \Sigma(B) = s.$$

Dies widerspricht aber

$$\Sigma(\phi(B)) = \lim_m \Sigma(\phi(B_{m_\nu})) = \lim_m s_{m_\nu+1} = s.$$

Es folgt die Behauptung. **q.e.d.**

Die Eigenvektoren der Matrix erhält man beim Jacobi-Verfahren als die Spalten des Produktes der G_m bzw. jedes Häufungspunkts dieses Produktes.

Der Fehler nimmt geometrisch mit einer Rate $1 - 1/n^2$ ab. Man braucht daher cn^2 Schritte, um eine Verbesserung um den Faktor

$$e^{-c} \approx \left(1 - \frac{1}{n^2}\right)^{cn^2}$$

zu erreichen. In jedem Schritt ist der Aufwand $O(n^2)$ wegen der Suche nach dem Maximum. Der Aufwand ist also $O(n^4)$, um eine vorgegebene Genauigkeit zu erreichen. Bei zyklischen Verfahren ist diese Abschätzung so nicht zu beweisen. Es fällt allerdings die aufwändige Maximumssuche weg.

Das Jacobi-Verfahren erweist sich in der Praxis als ein sehr gutes Verfahren für kleine bis mittlere Matrizen.

Für nicht symmetrische Matrizen A scheitert das Verfahren daran, dass die ausgewählten 2×2 Untermatrizen im Allgemeinen nur komplexe Eigenwerte haben oder gar nicht diagonalisierbar sind. Es gibt aber eine komplexe Variante des Jacobi-Verfahrens, die in der Praxis für alle Matrizen konvergiert.

Dabei wird das Maximum der Elemente unterhalb der Diagonalen $|a_{ij}|$ zu 0 gemacht, ohne dass, jedoch das Element $|\tilde{a}_{j,i}|$ oberhalb der Diagonalen ebenfalls 0 wird. Also

$$\begin{pmatrix} \tilde{a}_{i,i} & \tilde{a}_{i,j} \\ 0 & \tilde{a}_{j,j} \end{pmatrix} = Q^* \begin{pmatrix} a_{i,i} & a_{ij} \\ a_{j,i} & a_{j,j} \end{pmatrix} Q,$$

wobei Q unitär ist. Die erste Spalte von Q enthält daher einen Eigenwert der 2×2-Matrix.

Das folgende EMT-Programm implementiert dieses Verfahren auf eine Weise, die nicht sehr effizient ist. Es ist lediglich als Experiment anzusehen.

```
>function rotation2 (A) ...
$    l=eigenvalues(A);
$    v=[-A[1,2];A[1,1]-l[1]];
$    l=sqrt(conj(v').v);
$    v=v/l;
$    Q=[v[1],conj(v[2]);v[2],-conj(v[1])];
$    return Q;
$endfunction
>function cjacobi (A,eps=sqrt(epsilon)) ...
$    A0=A;
$    n=cols(A0);
$    repeat;
$        H=band(A0,-n,-1);
$        E=extrema(abs(H));
$        J=extrema(E[:,3]');
$        i=J[4];
$        j=E[i,4];
$        if J[3]<eps; break; endif;
$        A2=A0[[j,i],[j,i]];
$        Q2=rotation2(A2);
$        Q=id(n);
$        Q[[j,i],[j,i]]=Q2;
$        A0=conj(Q').A0.Q;
$    end;
$    return diag(A0,0);
$endfunction
>A=random(5,5);
>cjacobi(A)
 [ -0.302481174095+2.53488124406e-012i
 0.365732672026+5.08569624297e-007i  -0.214374695732+0.302175039756i
 -0.214374969489-0.302175548315i  2.30356014677-1.27255983533e-011i ]
>eigenvalues(A)
 [ -0.302481174098+0i   -0.214374637233-0.302174409838i
 -0.214374637233+0.302174409838i  0.365732281273+0i   2.30356014677+0i  ]
```

9.3 Vektoriteration

Bei der einfachen Vektoriteration startet man mit einem Vektor $x_0 \in \mathbb{K}^n$ und iteriert

$$x_{m+1} = \frac{1}{\|Ax_m\|} Ax_m.$$

9.9 Satz: *Für die Eigenwerte der Matrix $A \in \mathbb{K}^{n\times n}$ gelte*

$$|\lambda_1| = \ldots = |\lambda_r| > |\lambda_{r+1}| \geq \ldots |\lambda_n|.$$

und

$$\lambda_1 = \ldots = \lambda_r.$$

Dann ist jeder Häufungspunkt der einfachen Vektoriteration für fast alle Startpunkte ein Eigenvektor der Matrix A, und der Eigenwert lässt sich durch

$$\lim_{m\to\infty} x_m^* A x_m = \lambda_1$$

berechnen.

Fast alle Startpunkte bedeutet hier, dass Startpunkte alle bis auf eine Nullmenge (im Lebesgueschen Sinn) funktionieren. Die exakte Bedingung an den Startvektor wird sich im Beweis ergeben.

Zum Verständnis des Satzes beachte man auch, dass x_m eine beschränkte Folge ist, und daher jede Teilfolge einen Häufungspunkt hat.

Beweis: Wir nehmen der Einfachheit halber zunächst an, dass A diagonalisierbar ist. Sei $v_1, \ldots, v_n$ eine zugehörigen Basis aus Eigenvektoren und

$$x_0 = \sum_{k=1}^{n} \mu_k v_k \neq 0.$$

Dann ist für fast alle x_0 der Koeffizient μ_1 verschieden von 0. Da offensichtlich jedes x_m ein mit einer positiven Konstante normiertes Vielfaches von $A^m x_0$ ist, folgt sofort

$$x_m = \frac{1}{\|A^m x_0\|} A^m x_0.$$

Man hat

$$A^m x_0 = \sum_{k=1}^{n} \lambda_k^m \mu_k v_k.$$

Also

$$x_m = \frac{\lambda_1^m}{|\lambda_1|^m} \frac{1}{\|\sum_{k=1}^{n} (\lambda_k/\lambda_1)^m \mu_k v_k\|} \sum_{k=1}^{n} (\lambda_k/\lambda_1)^m \mu_k v_k.$$

Sei nun x ein Häufungspunkt der Folge x_m. Es gilt wegen der Voraussetzungen des Satzes

$$\lim_{m\to\infty} \frac{1}{\|\sum_{k=1}^{n} (\lambda_k/\lambda_1)^m \mu_k v_k\|} \sum_{k=1}^{n} (\lambda_k/\lambda_1)^m \mu_k v_k = \frac{1}{\|\sum_{k=1}^{r} \mu_k v_k\|} \sum_{k=1}^{r} \mu_k v_k.$$

Also folgt

$$x = \frac{\sigma}{\|\sum_{k=1}^{r} \mu_k v_k\|} \sum_{k=1}^{r} \mu_k v_k.$$

wobei σ ein Häufungspunkt der Folge $(\lambda_1/|\lambda_1|)^m$ ist. Also ist x in der Tat im Eigenraum von λ_1.

Wenn A nicht diagonalisierbar ist, so kann man den Satz beweisen, indem man benutzt, dass A ähnlich zu einer Jordanmatrix ist.

Sei x irgendein Häufungspunkt der x_m, also ein Eigenvektor zum Eigenwert λ_1. Dann gilt für m_k aus der entsprechenden Teilfolge

$$\lim_{k\to\infty} x_{m_k}^* A x_{m_k} = x^* A x = \lambda_1 x^* x = \lambda_1$$

wegen $\|x\| = 1$. **q.e.d.**

9.10 Aufgabe: Sei $\lambda \in \mathbb{K}$, $\lambda \neq 0$. $A \in \mathbb{K}^{n\times n}$ sei die Jordanmatrix

$$A = \begin{pmatrix} \lambda & 1 & & & 0 \\ & \lambda & 1 & & \\ & & \ddots & \ddots & \\ & & & \lambda & 1 \\ 0 & & & & \lambda \end{pmatrix}.$$

Berechnen Sie für $m > n$ die Matrix A^m, in dem Sie $A = \lambda I_n + H$ zerlegen. Zeigen Sie, dass für alle Startwerte $x_0 \in \mathbb{K}^n$, $x_0 \neq 0$, jeder Häufungspunkt der Vektoriteration ein Vielfaches des Einheitsvektors e_1 ist.

9.11. Beispiel: Die Matrix

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

ist nicht diagonalisierbar. Sie ist eine Jordanmatrix. Mit dem Startpunkt

$$x_0 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

hat man

$$x_m = \frac{1}{\sqrt{1+m^2}} \begin{pmatrix} m \\ 1 \end{pmatrix}.$$

Es tritt Konvergenz gegen den Eigenvektor

$$x = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

ein. Die Konvergenzgeschwindigkeit ist aber schlecht, denn

$$\|x_m - x\| \approx \frac{1}{m}.$$

Ohne die Voraussetzung

$$\lambda_1 = \ldots = \lambda_r.$$

ist der Satz nicht gültig. Beispiele sind alle reelle Matrizen, deren betragsgrößter Eigenwert λ nicht reell ist. In diesem Fall ist ja immer auch $\overline{\lambda} \neq \lambda$ Eigenwert und $|\lambda| = |\overline{\lambda}|$. Ein anderes einfaches Beispiel ist die Matrix

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

für die zum Beispiel mit dem Startvektor $x = (1,1)^T$

$$x_m = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ (-1)^m \end{pmatrix}$$

wird. Kein Häufungspunkt dieser Folge ist Eigenvektor.

Die Iterationsfolge konvergiert im Allgemeinen nicht gegen einen Eigenvektor, wie das Beispiel

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

zeigt. Dort wird

$$x_m = \frac{1}{\sqrt{2}} \begin{pmatrix} (-1)^m \\ (-1)^m \end{pmatrix}$$

Zwar sind alle x_m Eigenvektoren, aber die Folge konvergiert nicht. Wenn der höchste Eigenwert allerdings positiv ist, tritt Konvergenz ein, wie man im Beweis des Satzes sieht.

Für positiv definite Matrizen ist die Voraussetzung des Satzes über die Eigenvektoren immer erfüllt, so dass mit fast allen Startvektoren Konvergenz eintritt. Aus numerischen Gründen tritt die Konvergenz in der Tat für alle Startwerte ein.

Die Iteration ist natürlich am effektivsten, wenn die Matrix schwach besetzt ist. Aus diesem Grund sollte eine symmetrische Matrix vorher orthogonal in eine ähnliche Tridiagonalmatrix übergeführt werden. Der Aufwand hängt allerdings immer noch stark vom Verhältnis der Eigenwerte ab.

9.12. Beispiel: In EMT kann man die Funktion `iterate` verwenden, die selbständig auf Konvergenz testet. Allerdings ferlässt man sich dann auf die Konvergenz der Eigenvektoren. Besser ist es wie im folgenden Beispiel die Eigenwerte zu testen.

```
>function eigiter (M,x) ...
$  lambda=0;
$  repeat
$    xnew=M.x;
$    lambdanew=x'.xnew;
$    until lambdanew~=lambda;
$    x=xnew/norm(xnew);
$    lambda=lambdanew;
$  end;
$  return lambdanew,x;
$endfunction
>n=5; M=random(n,n); A=M'.M;
>lambda,x=eigiter(A,random(n,1)); lambda,
 5.82888660854
>x'.A.x
 [ 5.82888660854 ]
>max(abs(eigenvalues(A)))
 5.82888660854
```

Die eingebaute Funktion `eigenvalues` berechnet die Eigenwerte einfach über die Nullstellen des charakteristischen Polynoms.

Die Iteration ist natürlich am einfachsten auszuführen, wenn A schwach besetzt ist. Es empfiehlt sich daher, A zunächst in eine ähnliche Tridiagonalmatrix umzuwandeln, wenn A nicht ohnehin schwach besetzt ist.

9.13. Beispiel: Im folgenden Beispiel erzeugen wir wieder die Matrix aus Aufgabe 4, und führen die Iteration mit einer komprimierten Matrix aus. Dazu verwenden wir `cpx` und `cpxmult`. Obwohl die beiden hächsten Eigenwerte hier nur schwach getrennt sind, lässt sich das Verfahren mit EMT für $n = 100$ durchführen.

```
>n=100;
>A=2*id(n); A=setdiag(setdiag(A,1,1),-1,1);
>A[n-1,n]=sqrt(2); A[n,n-1]=sqrt(2);
>M=cpx(A);
>function eigiter (M,x) ...
$  lambda=0;
$  repeat
$    xnew=cpxmult(M,x);
$    lambdanew=x'.xnew;
$    until lambdanew~=lambda;
$    x=xnew/norm(xnew);
$    lambda=lambdanew;
$  end;
$  return lambdanew,x;
$endfunction
>x0=random(n,1);
>eigiter(M,x0)
 3.99975325597
>max(chebzeros(0,4,n))
 3.99975326496
```

9.14 Aufgabe: Wenn B positiv definit ist, so wird das Minimum von

$$f(x) = x^T Bx$$

unter der Nebenbedingung

$$g(x) = x^T x = \|x\|^2 = 1$$

im jedem Eigenvektor zum kleinsten Eigenwert von B angenommen, das Maximum in jedem Eigenvektor zum größten Eigenwert von B.

Aufgrund dieser Aufgabe kann die Vektoriteration als Versuch gedeutet werden, die Funktion

$$f(x) = x^T A^{-1} x$$

unter der Nebenbedingung

$$g(x) = x^T x = \|x\|^2 = 1$$

zu minimieren. Das Minimum wird in jedem Eigenvektor zum kleinsten Eigenwert von A^{-1} angenommen, also zum größten Eigenwert von A. Dazu minimiert man, ausgehend von x_n mit $\|x_n\| = 1$ die Funktion f auf der Tangentialebene zur Nebenbedingung in x, also auf

$$E = \{x : x^T x_n = 1\}.$$

Das heißt, man verwendet die Nebenbedingung

$$\tilde{g}(x) = x^T x_n = 1.$$

Nach Lagrange wird dieses Minimum in einem Punkt $\tilde{x}_{n+1}$ angenommen, in dem

$$2\tilde{x}_{n+1}^T A^{-1} = \text{grad } f(\tilde{x}_{n+1}) = \lambda \text{ grad } \tilde{g}(\tilde{x}_{n+1}) = \lambda x_n^T$$

angenommen. Das bedeutet in einem Punkt

$$\tilde{x}_{n+1} = \frac{\lambda}{2} A x_n.$$

Normiert man dieses Minimum, so dass $g(x_{n+1}) = 1$ gilt, so folgt

$$x_{n+1} = \frac{1}{\|Ax_n\|} Ax_n.$$

Dies ist exakt das Vektoritationsverfahren.

Die Konvergenzgeschwindigkeit hängt im diagonaliserbaren Fall offenbar von dem Verhältnis

$$|\lambda_{r+1}/\lambda_1|$$

ab. Aus diesem Grunde scheitert die Iteration oft in der Praxis, obwohl die Voraussetzungen erfüllt sind. Um die Konvergenz zu verbessern, kann man die inverse Iteration nach Wielandt anwenden.

Sei dazu λ eine Näherung für den Eigenwert λ_k von A. Dann hat die Matrix $A - \lambda I$ den Eigenwert $\lambda_k - \lambda$ und folglich die Matrix $(A - \lambda I)^{-1}$ den Eigenwert

$$\frac{1}{\lambda_k - \lambda}.$$

Wenn nun

$$|\lambda_k - \lambda| \ll |\lambda_i - \lambda|$$

für alle Eigenwerte λ_i, $i \neq k$ ist, so hat man

$$\frac{1}{|\lambda_k - \lambda|} \gg \frac{1}{|\lambda_i - \lambda|}.$$

Dies ist günstig für die Vektoriteration

$$x_{m+1} = \frac{1}{\|(A - \lambda I)^{-1} x_m\|} (A - \lambda I)^{-1} x_m.$$

Um die Iteration zu berechnen, lösen wir das Gleichungssystem

$$(A - \lambda I)\tilde{x}_{m+1} = x_m$$

und setzen

$$x_{m+1} = \frac{1}{\|\tilde{x}_{m+1}\|} \tilde{x}_{m+1}.$$

Weil aber der Aufwand relativ hoch ist, wird dieses Verfahren nur zur schnellen Verbesserung von Näherungswerten mit einem einzigen Iterationsschritt angewendet.

Die inverse Iteration ist am leichtesten auszuführen, wenn A eine Tridiagonalmatrix ist. In diesem Fall lässt sich das Gleichungssystem mit einem Aufwand von $O(n)$ auflösen.

9.15. Beispiel: Wir verbessern das Ergebnis der Iteration im vorigen Beispiel durch einen Schritt der inversen Iteration. Zur Läsung des Gleichungssystems verwenden wir das CG-Verfahren, das in EMT für komprimierte Matrizen mit `cpxfit` enthalten ist. Das Ergebnis ist in der Tat deutlich besser.

```
>x0=random(n,1);
>lambda,x=eigiter(M,x0); lambda,
 3.99975325597
>max(chebzeros(0,4,n))
 3.99975326496
>Alambda=cpxset(M,(1:n)'|(1:n)'|(2-lambda));
>xv=cpxfit(Alambda,x);
>xv=xv/norm(xv);
>xv'.cpxmult(M,xv)
 [ 3.99975326496 ]
```

Anstatt eine komprimierte Matrix $A - \lambda I_n$ neu zu erzeugen, setzen wir hier einfach die Diagonalelemente von A auf $2 - \lambda$.

9.4 LR- und QR-Verfahren

Das LR-Verfahren von Rutishauser und das QR-Verfahren von Francis sind ziemlich ähnlich aufgebaut. Die Idee beider Verfahren ist, eine Folge von ähnlichen Matrizen A_m aus $A = A_0$ zu konstruieren, die unter gewissen Bedingungen gegen eine rechte obere Dreiecksmatrix konvergieren.

Im LR-Verfahren wird A_m durch Frobeniusmatrizen in eine rechte obere Dreiecksmatrix umgewandelt, solange das ohne Permutationen möglich ist. Also etwa

$$F_m A_m = R_m$$

F_m enthält dabei die nötigen Zeilenoperationen. Anschließend werden inversen Operationen von rechts als Spaltenoperationen angewendet, also

$$F_m A_m F_m^{-1} = R_m F_m^{-1} =: A_{m+1}.$$

A_m und A_{m+1} sind dabei ähnlich und haben dieselben Eigenwerte. Mit $L_m = F_m^{-1}$ kann man das als

$$A_m = L_m R_m, \quad A_{m+1} = R_m L_m$$

schreiben. Das Verfahren bricht zusammen, wenn in einem Schritt keine LR-Zerlegung möglich ist.

Analog verwendet das QR-Verfahren statt Frobeniusmatrizen orthogonale Matrizen

$$A_m = Q_m R_m, \quad A_{m+1} = R_m Q_m.$$

Eine QR-Zerlegung ist ja immer möglich. Wir beschränken uns daher auf dieses Verfahren.

9.16 Satz: *Setzt man*

$$P_m = Q_0 \cdot \ldots \cdot Q_{m-1}, \quad H_m = R_{m-1} \cdot \ldots \cdot R_0,$$

so ist P_m orthogonal und H_m eine rechte obere Dreiecksmatrix, und es gilt für das QR-Verfahren

$$A_{m+1} = P_m^T A P_m,$$

sowie

$$A^m = P_m H_m$$

für alle $m \in \mathbb{N}$.

Beweis: Die erste Gleichung ist eine unmittelbare Folgerung aus der Ähnlichkeitsbeziehung

$$A_{m+1} = R_m Q_m = Q_m^T A_m Q_m,$$

die für alle $m \geq 0$ gilt. Außerdem

$$Q_m A_{m+1} = A_m Q_m.$$

Es folgt.

$$\begin{aligned} P_m H_m &= Q_0 \cdot \ldots \cdot Q_{m-1} R_{m-1} \cdot \ldots \cdot R_0 \\ &= Q_0 \cdot \ldots \cdot Q_{m-2} A_{m-1} R_{m-2} \cdot \ldots \cdot R_0 \\ &= A_0 P_{m-1} H_{m-1} \\ &= A P_{m-1} H_{m-1}. \end{aligned}$$

Per Induktion folgt die Behauptung. **q.e.d.**

9.17 Satz: *Die Matrix* $A \in \mathbb{R}^{n \times n}$ *habe n Eigenwerte, die alle vom Betrag verschieden sind, und die Matrix der Eigenvektoren, der Größe nach sortiert, habe eine LR-Zerlegung. Dann konvergiert das QR-Verfahren gegen eine obere Dreiecksmatrix.*

Den Beweis dieses Satzes, der von Wilkinson stammt, findet man im Buch von Stoer und Bulirsch. Dort findet man auch Shift-Strategien, die die Konvergenz verbessern und ähnlich wie die inverse Iteration von Wielandt funktionieren. Es gibt auch eine Erweiterung des Verfahrens für reelle Matrizen mit konjugiert komplexen Eigenwerten.

Der Aufwand des Verfahrens wird reduziert, wenn die Iteration auf Hessenberg-Matrizen A angewendet wird. Dann sind nämlich alle A_m wieder von dieser Form. Denn es genügt dann, $n-1$ Givens-Rotationen auf eine Hessenberg-Matrix A_m anzuwenden, um eine obere Dreiecksmatrix R_m zu erhalten. Also

$$G_{n-1,n} \cdot \ldots \cdot G_{1,2} A_m = R_m.$$

Es gilt dann

$$A_{m+1} = R_m G_{1,2}^T \cdot \ldots \cdot G_{n-1,n}^T.$$

Die Rotationen $G_{i,i+1}$ wirken nur auf die i-te und $i+1$-te Spalte. Man sieht auf diese Weise, dass auch A_{m+1} eine Hessenbergmatrix ist. Wenn A_m symmetrisch ist, so ist es auch A_{m+1}, so dass also auch Tridiagonalmatrizen erhalten bleiben.

9.18. Beispiel: Wir testen das Verfahren am Beispiel aus Aufgabe 4. Zur Demonstration zeigen wir zunächst, dass Tridiagonal-Matrizen in der Tat in dieser Form bleiben.

```
>n=4;
>A=2*id(n); A=setdiag(setdiag(A,1,1),-1,1);
>A[n-1,n]=sqrt(2); A[n,n-1]=sqrt(2);
>shortestformat; A,
         2         1         0         0
         1         2         1         0
         0         1         2      1.41
         0         0      1.41         2
>B,Q=givensqr(A,id(n));
>B
     -2.24     -1.79    -0.447         0
         0     -1.67     -1.91    -0.845
         0         0     -1.77     -2.28
         0         0         0    -0.302
>B.Q'
       2.8     0.748         0         0
     0.748      2.34      1.06         0
         0      1.06      2.68     0.241
         0         0     0.241     0.182
```

Die Implementation der Iteration geht dann wie folgt.

```
>function qriter (A) ...
$  M=id(cols(A));
$  H=M;
$  repeat
$    B,Q=givensqr(A,M);
$    Anew=B.Q';
$    H=Q.H;
$    until all(Anew~=A);
$    A=Anew;
$  end;
$  return Anew,H;
$endfunction
>D,H=qriter(A);
>longformat; sort(diag(D,0))
 [ 0.152240934977  1.23463313527  2.76536686473  3.84775906502 ]
>sort(real(eigenvalues(A)))
 [ 0.152240934977  1.23463313527  2.76536686473  3.84775906502 ]
>totalmax(abs(H.A.H'-D))
 0
```

Der Aufwand in jedem Schritt ist damit $O(n^2)$, bei Tridiagonalmatrizen sogar nur $O(n)$.

9.19. Beispiel: Dabei ist zu beachten, dass man für komprimierte Tridigonalmatrizen besser spezielle Verfahren in einer Programmiersprache verwenden sollte. Der folgende Code zeigt eine einfache Implementation in Java.

```
/**
 * Diese Funktion iteriert eine Tridiagonalmatrix
 * mit dem QR-Verfahren, bis die Elemente unterhalb
 * der Diagonalen verschwinden.
 * Die Konvergenz ist nur gesichert, wenn die Eigenwerte
 * vom Betrag her verschieden und reell sind.
 * Die Implementation ist nicht optimal.
 * @param A Tridiagonalmatrix
 * @return Vektor mit Eigenwerten
 */
```

```
public static double[] qriterate (double A[][], double epsilon)
   int n=A.length;
    double cc[]=new double[n-1];
    double ss[]=new double[n-1];
    int count=0;
    while (true)
       double sum=0;
        for (int j=0; j<n-1; j++) // Loop über Spalten
           // Berechne c=cos(phi) und s=sin(phi):
            sum+=Math.abs(A[j+1][j]);
            double r=Math.sqrt(A[j+1][j]*A[j+1][j]+A[j][j]*A[j][j]);
            if (r<1e-16)  // Nichts zu tun!
               cc[j]=2; continue;

            double c=A[j][j]/r,s=-A[j+1][j]/r;
            // Rotiere j-te und i-te Zeile von A
            for (int k=0; k<n; k++)
               double h=A[j][k]*c-A[j+1][k]*s;
                A[j+1][k]=A[j][k]*s+A[j+1][k]*c;
                A[j][k]=h;

            cc[j]=c; ss[j]=s;

        for (int j=0; j<n-1; j++) // Loop über Spalten
           // Berechne c=cos(phi) und s=sin(phi):
            double c=cc[j],s=ss[j];
            if (c>1) continue;
            // Rotiere j-te und i-te Spalte von A
            for (int k=0; k<n; k++)
               double h=A[k][j]*c-A[k][j+1]*s;
                A[k][j+1]=A[k][j]*s+A[k][j+1]*c;
                A[k][j]=h;


        if (sum<epsilon) break;
        count++;
        if (count>n*n*100)
            throw new RuntimeException("Iteration failed!");

    double x[]=new double[n];
    for (int i=0; i<n; i++)
        x[i]=A[i][i];
    return x;
```

Index