

Unit Numerik 1

Gruppe PT2

Dipl.-Math. Marco Boßle

Universität Stuttgart

16. März 2011

Begriffe

Was ist Numerik?

„Die Numerische Mathematik, kurz Numerik genannt, beschäftigt sich als Teilgebiet der Mathematik mit der Konstruktion und Analyse von Algorithmen für kontinuierliche mathematische Probleme.

Hauptanwendung ist dabei die Berechnung von Lösungen mit Hilfe von Computern.“

www.wikipedia.de

Analytische Lösung gegeben durch Formel, z.B. Lösungen von

$$x^2 + 2px + q = 0$$

Nicht geschlossene Lösung gegeben durch unendliche Formel, z.B.

$$\frac{\pi^2}{6} = 1 + 1/4 + 1/9 + 1/16 + 1/25 + \dots$$

Algorithmus Festgelegte Vorgehensweise zur Lösung eines Problems (meist unter Beachtung von Sonderfällen) z.B. Bestimmung von Eigenwerten, Konstruktion des Schwerpunktes in einem Parallelogramm

Bit **B**inary **D**igit – interne Darstellung von Information, üblicherweise mit 0 und 1 dargestellt (Anwesenheit/Abwesenheit von Spannung)

Byte 8 Bit

Numerik

Ziele

- Lösung mathematischer Probleme
- Berechnung von Ergebnissen, die nicht geschlossen darstellbar sind
- Steigerung der Effizienz von Algorithmen
Laufzeit – Speicherbedarf

Anwendungen

- Simulation (Lösung von partiellen Differentialgleichungen)
- Visualisierung (Geometrische Datenverarbeitung)
- Steuerung (Echtzeitsysteme)
- Messung (Computertomographie)
- uvm.

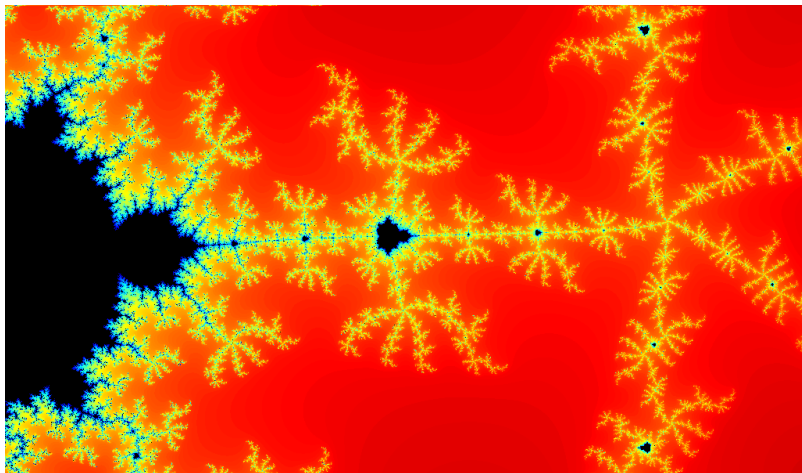
Gegebenheiten

endliche Zahlenmenge	⇒	Rundungsfehler
endlicher Speicher	⇒	Diskretisierungsfehler
endliche Rechenzeit	⇒	Verfahrensfehler

Experimentelle Mathematik

Rekursion $z_0 = 0$, $c \in \mathbb{C}$

$$z_{n+1} = z_n^2 + c$$



Ganze Zahlen - Integer

Ganze Zahlen können zu unterschiedlichen Basen β dargestellt werden. Üblicherweise wird für eine Zahl d eine festgelegte Anzahl n von Stellen reserviert:

$$d = \sum_{k=1}^n b_k \beta^{n-k}$$

Beispiele:

$$\begin{aligned} 1 &= 1 * 2^0 & &= (0 \ 0 \ 1)_2 & &= (0 \ 0 \ 1)_3 \\ 4 &= 1 * 2^2 & &= (1 \ 0 \ 0)_2 & &= (0 \ 1 \ 1)_3 \\ 7 &= 1 * 2^2 + 1 * 2^1 + 1 * 2^0 & &= (1 \ 1 \ 1)_2 & &= (0 \ 1 \ 2)_5 \\ 26 &= 1 * 2^4 + 1 * 2^3 + 1 * 2^1 & &= (1 \ 1 \ 0 \ 1 \ 0)_2 & &= (1 \ 10)_{16} \end{aligned}$$

⇒ es existiert stets eine größte darstellbare Zahl:

$$d_{\max} = \sum_{k=1}^n (\beta - 1) \beta^{n-k} = \beta^n - 1$$

Normalisierte Gleitpunktzahlen

Abgebrochene Dezimaldarstellung reeller Zahlen x

$$x = \sigma b_1 . b_2 \dots b_n \cdot \beta^p = \sigma \sum_{k=1}^n b_k \beta^{p-k+1}$$

Notation: $x = (\sigma b_1 . b_2 \dots b_n \text{ E } p)_\beta$

Dabei ist

- Vorzeichen σ
- Mantisse $[b_1, b_2 \dots b_n]$, $b_1 \neq 0, n > 0$
- Exponent $p = [\pm b_{n+1} \dots b_{n+m}]$, $m > 0$

Beispiele:

- $(-1.234 \text{ E } 2)_{10} = -123.4$
- $(-1.01 \text{ E } -2)_{10} = -0.0101$
- $(+1.01 \text{ E } -2)_2 = 1 \cdot 2^{-2} + 0 \cdot 2^{-3} + 1 \cdot 2^{-4} = \frac{1}{4} + \frac{1}{16} = \frac{5}{16} = 0.3125$
- $\frac{23}{9} = 2 + \frac{5}{9} = 2 + \frac{1}{3} + \frac{2}{9} = (2.12 \text{ E } 0)_3$

Eigenschaften der normalisierten Gleitpunktzahlen:

- Umsetzung in Hardware
- Berechnung aufwändiger als Integer
- Alle Zahlen, deren Dezimaldarstellung nicht nach n Stellen abbricht, werden gerundet gespeichert
- Größte darstellbare Zahl: $x_{\max} = \beta^{p_{\max}+1}(1 - \beta^{-n})$
- Betragsmäßig kleinste darstellbare Zahl: $x_{\min} = \beta^{p_{\min}}$
- keine Darstellung für Null!
- eindeutige Gleitpunktzahl $R x$ für alle $x \in [-x_{\max}, x_{\max}]$

Fehler

Differenz zwischen exaktem x und fehlerbehaftetem \tilde{x}

Absoluter Fehler $\tilde{x} - x$

- Größe des Fehlers
- Unabhängig von der Größenordnung von x
- Unübersichtlich für große x

Relativer Fehler $\frac{\tilde{x} - x}{x}$

- Größenordnung des Fehlers
- Qualitative Aussage über den Fehler
- Evtl. wenig aussagekräftig für kleine x

Häufig interessieren nur die Beträge der Fehler

$$|\tilde{x} - x| \quad \frac{|\tilde{x} - x|}{|x|}$$

Rundungsfehler

Als Maschinengenauigkeit wird

$$\text{eps} = \beta^{1-n}/2$$

bezeichnet.

Reelle Zahlen werden zur Speicherung auf Gleitpunktzahlen gerundet

$$x \mapsto R(x) \quad x \in \mathbb{R}$$

Dabei entsteht ein relativer Fehler von

$$\frac{R x - x}{x} = \delta, \quad |\delta| < \text{eps}$$

Der gerundete Wert entspricht

$$R x - x = \delta x \quad \Rightarrow \quad R x = (1 + \delta)x$$

Beispiele

- Maschinengenauigkeit bei Mantissenlänge $n = 5$, Basis $\beta = 10$

$$\text{eps} = 10^{1-5}/2 = \frac{1}{20000} = 5 \cdot 10^{-5}$$

- π mit $n = 5$, $\beta = 10$: $(3.1416 \text{ E } 0)_{10}$

Rundungsfehler:

$$\frac{|R\pi - \pi|}{|\pi|} = \frac{3.1416 - 3.1415926535897 \dots}{3.1415926535897 \dots} \approx 2.34 \cdot 10^{-6} < \text{eps}$$

- Maschinengenauigkeit bei Mantissenlänge $n = 52$, Basis 2

$$\text{eps} = 2^{1-52}/2 \approx 0.000000000000000022204 = 2.2204 \cdot 10^{-16}$$

- $q = 100!$ mit $n = 52$, $\beta = 2$:

$$\begin{aligned} Rq &= R9332621544394415\mathbf{2681} \dots \\ &= (1.101100110000100101100100111 \\ &\quad 0110000111001010111011100 E 1000001011)_2 \\ &\approx 9.332621544394415\mathbf{0965} \cdot 10^{157} \end{aligned}$$

Absoluter Fehler

$$|Rq - q| \approx 1.72 \cdot 10^{141}$$

Relativer Fehler

$$\frac{|Rq - q|}{|q|} \approx \frac{1.72 \cdot 10^{141}}{9.33 \cdot 10^{157}} \approx 1.84 \cdot 10^{-17} < \text{eps}$$

Auslöschung

Addition

$$z = x + y$$

Berechnung in Gleitpunktzahlen $R(Rx + Ry)$

$$Rx = x + \delta_x x = (1 + \delta_x)x$$

$$Ry = y + \delta_y y = (1 + \delta_y)y$$

$$Rz = z + \delta_z z = (1 + \delta_z)z$$

Absoluter Fehler

$$\begin{aligned} |Rz - z| &= |R(Rx + Ry) - x - y| \\ &= |(1 + \delta_z)(x + \delta_x x + y + \delta_y y) - x - y| \\ &= |\delta_x x + \delta_y y + \delta_z(x + y) + \delta_x \delta_z x + \delta_y \delta_z y| \\ &\leq |\delta_x||x| + |\delta_y||y| + |\delta_z||x + y| + |\delta_x \delta_z||x| + |\delta_y \delta_z||y| \\ &\leq \text{eps}(|x| + |y| + |x + y|) + O(\text{eps}^2) \end{aligned}$$

Relativer Fehler

$$\begin{aligned} \frac{|Rz - z|}{|z|} &\leq \text{eps} \frac{|x| + |y| + |x + y|}{|x + y|} + O(\text{eps}^2) \\ &= \text{eps} \left(\frac{|x| + |y|}{|x + y|} + 1 \right) + O(\text{eps}^2) \end{aligned}$$

Für $x \approx -y$ ist

$$\frac{|x| + |y|}{|x + y|}$$

nicht kontrollierbar, der relative Fehler kann unverwertbar groß werden.

⇒ eine Addition von Gleitpunktzahlen ist nicht stabil

Beispiel:

- Mantissenlänge $n = 3$, Basis $\beta = 10$, $x = 2014$, $y = -2005$,
 $z = x + y = 9$

$$\frac{|R(Rx + Ry) - z|}{|z|} = \frac{|R((2.01 \text{ E } 3)_{10} + (-2.01 \text{ E } 3)_{10}) - 9|}{9} = 1$$

Algorithmischer Fehler

Beispiel: Fehlerbehaftete Berechnung von $f(x) = e^x$

$$x = 100, \quad \tilde{x} = 101 = x + \delta x \quad \text{mit } \delta = \frac{1}{100}$$

relativer Fehler im Eingangswert

$$\frac{|\tilde{x} - x|}{|x|} = \frac{|x + \delta x - x|}{|x|} = |\delta| = 0.01$$

Fehler der Berechnung

$$|f(\tilde{x}) - f(x)| = e^x e^{\delta x} - e^x = e^x (e^{\delta x} - 1) = e^{100} (e - 1) \approx 4.62 \cdot 10^{43}$$

$$\frac{|f(\tilde{x}) - f(x)|}{|f(x)|} = \frac{|e^x (e^{\delta x} - 1)|}{|e^x|} = e - 1 \approx 1.72$$

Kondition einer Berechnung

Definition

Für eine differenzierbare Funktion f wird die *Kondition* durch

$$c = \frac{|xf'(x)|}{|f(x)|}$$

definiert.

Theorem (Kondition)

Gegeben ist eine reelle differenzierbare Funktion $y = f(x)$, eine fehlerbehaftete Eingabe $\tilde{x} = x + \delta x$ mit dem rel. Eingabefehler δ .

Dann gilt:

$$\frac{|f(\tilde{x}) - f(x)|}{|f(x)|} \approx c|\delta|$$

Beweis:

$$\frac{|f(\tilde{x}) - f(x)|}{|f(x)|} = \frac{|f(x + \delta x) - f(x)|}{|\delta x|} \frac{|\delta x|}{|f(x)|} \approx |f'(x)| \frac{|\delta||x|}{|f(x)|} = \frac{|x||f'(x)|}{|f(x)|} |\delta|$$

□

Die Konditionszahl gibt also an, wie stark der Eingabefehler durch die Funktion f verstärkt wird.

Beispiele:

- $f(x) = e^x \Rightarrow c = \frac{|xe^x|}{|e^x|} = |x|$

Gut konditioniert für kleine x

- $f(x) = x + a \Rightarrow c = \frac{|x|}{|x + a|}$

Schlecht konditioniert für $x \approx -a$

Sonderfall:

- $f(x) = \cos(x) \Rightarrow c = \frac{|x \sin(x)|}{|\cos(x)|}$

Schlecht konditioniert für $x \approx \pi/2$

Allerdings wird hier nur der relative Fehler groß. Der absolute Fehler für $x = \pi/2$ ist

$$\begin{aligned} |f(\tilde{x}) - f(x)| &= |\cos(\pi/2 + \delta\pi/2) - 0| \\ &= |\cos(\pi/2)\cos(\delta\pi/2) - \sin(\pi/2)\sin(\delta\pi/2)| \\ &= |\sin(\delta\pi/2)| \end{aligned}$$

Stabilität

Ein Algorithmus ist eine Folge von Operationen. Eingabewerte sind

$$x_1, \dots, x_\ell \in D$$

$$x_{\ell+1} = f_{\ell+1}(x_1, \dots, x_\ell)$$

$$x_{\ell+2} = f_{\ell+1}(x_1, \dots, x_\ell, x_{\ell+1})$$

$$\vdots$$

$$x_n = f_n(x_1, \dots, x_{n-1})$$

Definition (Konditionen)

Für alle $x \in D$ mit $f(x) \neq 0$ ist die Kondition der Einzelschritte definiert durch

$$c_k = \left| \frac{\partial f_n}{\partial x_k} \right| \frac{|x_k|}{|x_n|}.$$

Theorem (Stabilität)

Der relative algorithmische Fehler ist

$$\frac{|\tilde{x}_n - x_n|}{|x_n|} \leq |\delta_n| + \sum_{k=1}^{n-1} c_k |\delta_k| + O(\text{eps}^2) \leq \left(1 + \sum_{k=1}^{n-1} c_k\right) \text{eps} + O(\text{eps}^2).$$

Ein Algorithmus heißt stabil, wenn eine Konstante $s \in \mathbb{R}^+$ existiert mit

$$1 + \sum_{k=\ell+1}^{n-1} c_k \leq s \sum_{k=1}^{\ell} c_k$$

Ein stabiler Algorithmus hat demnach die Eigenschaft, die Eingabefehler höchstens um einen festen Faktor zu verstärken. Ein für praktische Zwecke geeigneter Algorithmus muss zudem noch eine moderate Stabilitätskonstante s aufweisen.

Beispiel: Nullstellenbestimmung

Gesucht wird die kleinste Nullstelle des Polynoms $x^2 - 2ax + b = 0$ mit $0 < b \ll a$:

$$x = a - \sqrt{a^2 - b}$$

Der Berechnungsalgorithmus setzt sich somit folgendermaßen zusammen:

$$x_1 = a, \quad x_2 = b, \quad x_3 = x_1^2, \quad x_4 = x_3 - x_2, \quad x_5 = \sqrt{x_4},$$

$$x_6 = x_1 - x_5 = x_1 - \sqrt{x_4} = x_1 - \sqrt{x_3 - x_2} = x_1 - \sqrt{x_1^2 - x_2}$$

Die Konditionen der Berechnungsschritte sind somit

$$\begin{aligned} c_1 &= \left| \frac{\partial f_n}{\partial x_1} \right| \frac{|x_1|}{|x_n|} = \left| 1 - \frac{x_1}{\sqrt{x_1^2 - x_2}} \right| \frac{|x_1|}{\left| x_1 - \sqrt{x_1^2 - x_2} \right|} \\ &= \frac{a \left| \sqrt{a^2 - b} - a \right|}{\sqrt{a^2 - b} (a - \sqrt{a^2 - b})} = \frac{a}{\sqrt{a^2 - b}} \end{aligned}$$

$$\begin{aligned}
 c_2 &= \left| \frac{1}{2\sqrt{x_1^2 - x_2}} \right| \frac{|x_2|}{\left| x_1 - \sqrt{x_1^2 - x_2} \right|} = \frac{b}{2\sqrt{a^2 - b}(a - \sqrt{a^2 - b})} \\
 &= \frac{b(a + \sqrt{a^2 - b})}{2\sqrt{a^2 - b}(a^2 - (a^2 - b))} = \frac{a + \sqrt{a^2 - b}}{2\sqrt{a^2 - b}} \\
 c_3 &= \frac{a^2(a + \sqrt{a^2 - b})}{2b\sqrt{a^2 - b}} & c_4 &= \frac{\sqrt{a^2 - b}(a + \sqrt{a^2 - b})}{2b} \\
 c_5 &= \frac{\sqrt{a^2 - b}(a + \sqrt{a^2 - b})}{b}
 \end{aligned}$$

Mit $0 < b \ll a$ gilt nun $\sqrt{a^2 - b} \approx a$ und somit

$$c_1 \approx 1, \quad c_2 \approx 1, \quad c_3 \approx \frac{a^2}{b}, \quad c_4 \approx \frac{a^2}{b}, \quad c_5 \approx \frac{2a^2}{b},$$

Für die Stabilität sind somit die Größen

$$1 + \sum_{k=\ell+1}^{n-1} c_k = 1 + 4 \frac{a^2}{b} \quad \sum_{k=1}^{\ell} c_k = 2$$

zu untersuchen. Die erste Größe wird für kleine $b > 0$ aber beliebig groß, die Berechnung der kleinsten Nullstelle von

$$x^2 - 2ax + b = 0 \quad \text{durch} \quad x = a - \sqrt{a^2 - b}$$

ist in diesem Bereich nicht stabil.

Ursache hierfür ist (für $\sqrt{a^2 - b} \approx a$) die Auslöschung relevanter Stellen in der Differenz $a - \sqrt{a^2 - b}$.

Einen stabilen Algorithmus erhält man (für diesen Bereich) durch

$$x = \frac{b}{a + \sqrt{a^2 - b}}.$$

Programmiersprachen

Numerische Berechnungen können mit jeder Programmiersprache durchgeführt werden.

ATS Ada Basic C C# C++ D Erlang F# Fortran Go Haskell Java Lisp
MATLAB Modula OCaml PHP Pascal Perl Processing Python Ruby Scala

Alle Sprachen haben eine spezielle Ausrichtung und Fähigkeiten und eine entsprechende Verbreitung. Für die Numerik ist meist die Laufzeit ein kritischer Faktor, es können aber auch Verbreitung (Plattformen), Ästhetik und Erlernbarkeit in Betracht gezogen werden.

Eine Sonderstellung nehmen sogenannte Computeralgebrasysteme ein (Maple, Mathematica). Sie bieten symbolische Berechnungen an, mit einem mitunter immensen Rechenaufwand.

Beispiel in Maple:

```
[> sum(1/k^2, k=1..infinity);
```

$$\frac{1}{6}\pi^2$$

Matrizenmultiplikation in C++ und Matlab

```

#include<iostream>
#include<sys/time.h>
using namespace std;

int main(void) {
    // Initialisierung
    timeval start, end;
    gettimeofday(&start, 0);
    int A[2][3] = {{1, 3, 4},{2, 0, 1}};
    int B[3][4] = {{1, 2, 3, 1},{2, 2, 2, 2},{3, 2, 1, 4}};
    int C[2][4] = {{0, 0, 0, 0},{0, 0, 0, 0}};
    int i = 0, j = 0, k = 0;

    // Berechnung
    for(i = 0; i < 2; i++) {
        for( j = 0; j < 4; j++) {
            for( k = 0; k < 3; k++) {
                C[i][j] += A[i][k] * B[k][j];
            }
            // Ausgabe
            cout << " " << C[i][j];
        }
        cout << endl;
    }

    // Laufzeit
    gettimeofday(&end, 0);
    cout << "Rechenzeit: "
         << end.tv_usec - start.tv_usec
         << " mikrosec\n";
}

```

```

% Initialisierung
tic;
A = [1 3 4; 2 0 1];
B = [1 2 3 1; 2 2 2 2; 3 2 1 4];

% Berechnung
C=A*B;

% Ausgabe
C

% Laufzeit
toc;

```

Laufzeit:

MATLAB	0.26 ms
C++	0.03 ms

Hilfesystem in MATLAB

Aufruf mit `help <Funktionsname>` oder `doc <Funktionsname>`

Beispiel:

```
>> help det
```

```
DET    Determinant.
```

```
DET(X) is the determinant of the square matrix X.
```

```
Use COND instead of DET to test for matrix singularity.
```

```
See also cond.
```

```
Overloaded methods:
```

```
gf/det
```

```
sym/det
```

```
laurmat/det
```

```
Reference page in Help browser
```

```
doc det
```

Zahlen

```

>> 123           >> 1.23           >> .000123
ans =            ans =            ans =
    123          1.2300          1.2300e-04

>> 1.23e-4      >> 12i           >> 12+3i
ans =            ans =            ans =
 1.2300e-04     0 +12.0000i         12.0000 + 3.0000i

```

Sonderfälle: inf (infinity) und nan (Not a Number)

```

>> 1e400      >> 1-inf      >> -1+inf      >> inf+inf
ans =         ans =         ans =         ans =
    Inf       -Inf         Inf         Inf

>> inf-inf    >> 1/0           >> 0/0
ans =         Warning: Divide by zero.  Warning: Divide by zero.
    NaN       ans =         ans =
              Inf         NaN

```

Elementare Funktionen

- +, -, *, /, ^ Verwendung wie gewohnt
- Malpunkt * muss immer geschrieben werden
- Klammerung mit (), Punkt vor Strich!
- Ein Semikolon am Ende der Zeile unterbindet die Ausgabe des Ergebnisses
- Mathematische Funktionen
 - ▶ cos, sin, tan, acos, asin, atan...
 - ▶ exp, pow2 Exponentialfunktion zur Basis e bzw. 2
 - ▶ log, log10, log2 Logarithmus zur Basis e, 10 bzw. 2
 - ▶ sqrt Quadratwurzel
 - ▶ abs, angle, conj, real, imag komplexe Zahlen
 - ▶ round, floor, ceil Runden (kaufmännisch, ab bzw. auf)
 - ▶ mod, rem, sign Modulo, Divisionsrest, Vorzeichen

Variablen

Variablen haben einen Bezeichner, der mit einem Buchstaben beginnt und aus max. 63 Buchstaben, Zahlen und `_` besteht.

- Zuweisung:

```
>> a=3  
a = 3
```

- Verwendung wie andere Zahlen

```
>> sqrt(3*a)-3  
ans = 0
```

- Groß- und Kleinschreibung beachten

```
>> A=3; a=2; A*a  
ans = 6
```

- Informationen über alle Variablen: `who` oder `whos`
- Löschen von Variablen: `clear A` oder `clear`

Beispiel

```
>> x=sqrt(2)
x =
    1.4142
>> y=asin(1/x)
y =
    0.7854
>> y-pi/4
ans =
   -1.1102e-16
>> who
Your variables are:
ans  x    y
>> clear y
>> whos
  Name      Size      Bytes  Class      Attributes
  ans       1x1         8      double
  x         1x1         8      double
>> clear
>> whos
>>
```

Matrizen

Intern behandelt Matlab (fast) alle Variablen als Matrizen. Skalare und Vektoren sind spezielle Matrizen der Größe 1×1 , $1 \times n$ bzw. $n \times 1$.
Kommata oder Leerzeichen trennen Spalten, Semikola oder Zeilenumbrüche trennen Zeilen.

```
>> 1                >> [1 1]                >> [1,1]
ans =              ans =              ans =
     1             1     1             1     1
>> [1,2;3,4]       >> [1 2;3 4]
ans =              ans =
     1     2             1     2
     3     4             3     4
>> [1 2;           >> [[1;3] [2;4]]
     3 4]
ans =              ans =
     1     2             1     2
     3     4             3     4
```

Vektoren und Matrizen

- `[1:3:14]=[1 4 7 10 13]` `[1:4]=[1 2 3 4]`
Vektor mit Einträgen von 1 bis ≤ 14 mit Differenzen 3
- `linspace(1,14,3)=[1.0000 7.5000 14.0000]`
Vektor mit 3 gleichverteilten Einträgen von 1 bis 14
- `zeros(n,m)`
Null-Matrix der Größe $n \times m$
- `ones(n,m)`
Eins-Matrix der Größe $n \times m$
- `eye(n,m)`
Einheits-Matrix der Größe $n \times m$
- `diag([1 2 3])`
Matrix mit dem angegebenen Vektor auf der Diagonalen
- `rand(n,m)`
Zufalls-Matrix der Größe $n \times m$ mit Elementen zwischen 0 und 1

Indizierung

Matrizen können teilweise ausgelesen werden

```
A =
    11    12    13    14    15    16    17    18    19
    21    22    23    24    25    26    27    28    29

>> A(2,3)      >> A(1:2,3)      >> A(:,3:end-2)
ans =           ans =           ans =
    23           13           13    14    15    16    17
           23           23    24    25    26    27

>> A(:, [2 8])  >> A(:,3:end-2)
ans =           ans =
    12    18           13    14    15    16    17
    22    28           23    24    25    26    27
```

aber auch gesetzt oder gelöscht werden.

```
>> A(2,:)=[]
A =
    11    12    13    14    15    16    17    18    19

>> A(1,3)=2
A =
    11    12     2    14    15    16    17    18    19
```

Scripte

In Matlab können Anweisungen in eine Datei geschrieben werden, die nach einem Aufruf wie die sukzessive Ausführung der einzelnen Anweisungen wirkt.

Datei `test.m`:

```
x=100;  
y=-3*x+301;  
z=sin(pi*y)
```

Aufruf in Matlab:

```
>> test  
z = 1.2246e-16
```

Scripte können in Matlab über [Datei] – [Öffnen] im Matlab-eigenen Editor geöffnet oder über [Datei] – [Neu] in diesem erstellt werden.

Sehr wertvoll ist die Verwendung von Kommentaren zur Dokumentation des Quellcodes.

```
% Initialisierung
```

```
a=3; b=134; c=pi;
```

```
% Berechnung des geometrischen Mittels
```

```
x=sqrt(a^2+b^2+c^2);
```

Funktionen

Anders als Skripte haben Funktionen Ein- und Ausgabewerte. Die berechneten Zwischenergebnisse sind nach außen nicht sichtbar (keine Kollisionen), nur die Rückgabewerte werden „veröffentlicht“.

```
function [ output_args ] = untitled1( input_args )
%UNTITLED1 Summary of this function goes here
%   Detailed explanation goes here

end
```

- Sichere Abkapselung der Variablen
- Verwendbar wie eingebaute Matlab-Funktionen, z.B. $y=\sin(x)$
- Verbesserung der Lesbarkeit von Quellcode
- In einer Funktion sind auch Unterfunktionen verwendbar

Verwendung einer Funktion mit Unterfunktionen

Datei `mfunc_test.m`

```
function [A,B,C]
    =mfunc_test(n,m,p)
```

```
A=(n+m)/2;
```

```
B=sqrt(n^2+m^2);
```

```
C=unterfunktion(p);
```

```
function y=unterfunktion(x)
    y=3*x^2-2*x+5;
```

Aufruf:

```
>> [A,B,C]=mfunc_test(2,3,4)
```

```
A =
    2.5000
```

```
B =
    3.6056
```

```
C =
    45
```

- Beliebig viele Unterfunktionen möglich
- `unterfunktion` ist von außen nicht aufrufbar

Schleifen

Eine for-Schleife hat einen bestimmten Umfang

```
for index = [vector]
    Schleifenkörper
end
```

In jedem Durchlauf wird `index` auf das nächste Element von `vector` gesetzt.

Eine while-Schleife hat eine Laufbedingung

```
while Bedingung
    Schleifenkörper
end
```

Vor jedem Durchlauf wird geprüft, ob die Bedingung erfüllt ist.

Beispiel 1:

```
function K=matlab_multkette(k1,k2,n)
% Berechnet einen Vektor, dessen Elemente
% das Produkt seiner beiden Vorgänger ist
K = [k1 k2 zeros(1,n-2)];
for k=3:n
    K(k) = K(k-2)*K(k-1);
end
```

Beispiel 2:

```
function P=potenzen(beta,n)
P=1;
while P<n
    P=P*beta;
end
```

Operatoren mit Matrizen

Manipulation von Matrizen

- Transponierung `'`
- Spiegelung `fliplr`
- Spiegelung `flipud`

Beispiel:

```
>> A=[1 2; 3 4]
```

```
A = 1 2  
    3 4
```

```
>> A=[1 2; 3 4]
```

```
A = 1 2  
    3 4
```

```
>> A'
```

```
ans = 1 3  
      2 4
```

```
>> [1 2]'
```

```
ans = 1  
      2
```

```
>> fliplr(A)
```

```
ans = 2 1  
      4 3
```

```
>> flipud(A)
```

```
ans = 3 4  
      1 2
```

Matrizen können nur miteinander verrechnet werden, wenn sie passende Größen aufweisen:

- Multiplikation: $(k \times n) \cdot (n \times m) \rightarrow (k \times m)$
- Addition: $(n \times m) + (n \times m) \rightarrow (n \times m)$

Ausnahmen

- Skalare Zuweisung/Erweiterung
- Skalare Multiplikation
- Skalare Addition

Beispiele:

```
>> A=[1 2 3; 4 5 6]
```

```
A = 1 2 3
     4 5 6
```

```
>> 2*A
```

```
ans = 2 4 10 10
      8 10 10 10
```

```
>> A(:, [3,4])=5
```

```
A = 1 2 5 5
     4 5 5 5
```

```
>> A-1
```

```
ans = 0 1 4 4
      3 4 4 4
```

Spezialfälle

- Skalarprodukt $\text{dot}(v, w) = v' * w$
- Kreuzprodukt $\text{cross}(v, w)$

Weitere Manipulationen

- min, max Kleinstes/Größtes Element
- sort Sortierung einer Matrix/eines Vektors
- sum, prod Summe/Produkt der Elemente eines Vektors

Beispiele:

```
>> A=[3 2 4 1; 2 2 3 4]
```

```
A =
```

```
     3     2     4     1
     2     2     3     4
```

```
>> sort(A)
```

```
ans =
```

```
     2     2     3     1
     3     2     4     4
```

```
>> sum(A)
```

```
ans =
```

```
     5     4     7     5
```

```
>> sort(A,2)
```

```
ans =
```

```
     1     2     3     4
     2     2     3     4
```

```
>> prod(A)
```

```
ans =
```

```
     6     4    12     4
```

Soll statt einer Matrixmultiplikation alle Elemente miteinander verrechnet werden, so können die Punkt-Operatoren verwendet werden:

- `.*` elementweise Multiplikation zweier Matrizen gleicher Größe
- `.^` elementweises Potenzieren einer Matrix
- `./` elementweise Division zweier Matrizen gleicher Größe

Beispiele:

```
>> A=[1 2 3; 4 5 6], B=[7 8; 9 10]
```

```
A =           B =
  1  2  3       7  8
  4  5  6       9 10
```

```
>> B*A           >> A.*A           >> A./A           >> A.^2
ans =           ans =           ans =           ans =
  39  54  69       1  4  9       1  1  1       1  4  9
  49  68  87      16 25 36       1  1  1      16 25 36
```

```
>> A*A
??? Error using ==> mtimes
Inner matrix dimensions must agree.
```

Verzweigungen

Eine bedingte Verzweigung wird in Matlab mit der `if`-Konstruktion realisiert.

```
if <Bedingung>
    <Anweisungen>
elseif <Bedingung>
    <Anweisungen>
else
    <Anweisungen>
end
```

Beispiel:

```
if zeit<7
    disp('Ich schlafe noch');
elseif zeit<8
    disp('Ich frühstücke gerade');
elseif zeit>22
    disp('Ich schlafe schon wieder');
else
    disp('Ich bin außer Haus');
end
```

Function Handles

Mit der Konstruktion

`name=@(Argumente) Funktion`

lässt sich eine (mathematische) Funktion definieren.

`f=@(x,y) 2*x.*y + y.^2` ist verwendbar wie `function z=f(x,y)`
`z = 2*x.*y + y.^2;`

Der große Vorteil von Function Handles ist, dass Sie wiederum als Argument einer Funktion übergeben werden können.

Beispiel:

```
function m=maxest(f,a,b)
    %Schätzer für das Maximum
    % einer Funktion
    m = max(f(linspace(a,b)));
```

Aufruf mit

```
>> f = @(x) x.^2+2*x-5;
>> maxest(f,-2,5)
ans = 30
```

Lineare Gleichungssysteme

Ein lineares Gleichungssystem

$$Ax = b$$

kann in Matlab mit dem Backslash-Operator gelöst werden.

```
>> A=[1 2 5; 4 5 6; 7 8 9], b=[0;8;14]
```

```
A =          b =  
    1    2    5          0  
    4    5    6          8  
    7    8    9         14
```

```
>> A\b
```

```
ans =  
    1  
    2  
   -1
```

Für $\det(A) = 0$ hat das Gleichungssystem keine Lösung:

```
>> A=[1 2 3; 4 5 6; 7 8 9], b=[0;8;14], A\b
```

```
A =           b =
     1     2     3         0
     4     5     6         8
     7     8     9        14
```

```
Warning: Matrix is close to singular or badly scaled.
Results may be inaccurate. RCOND = 1.541976e-18.
```

```
ans =
 1.0e+16 *
 0.9007
-1.8014
 0.9007
```

Der Backslash-Operator gibt auch für unterbestimmte oder uneindeutige Systeme genau eine Lösung aus.

Polynome

Für ein Polynom

$$p(x) = a_1x^{n-1} + \dots + a_{n-1}x + a_n$$

wird in Matlab nur der Vektor mit den Koeffizienten gespeichert.

$$p = [a_1 \dots a_n]$$

Für die weitere Bearbeitung stehen folgende Befehle zur Verfügung:

- `polyval(p,x)` Auswertung von p an x
- `roots(p)` Nullstellen des Polynoms p
- `polyfit(x,y,n)` Approximation gegebener Daten x, y
- `conv(p,q)` Koeffizienten der Produkts aus p und q

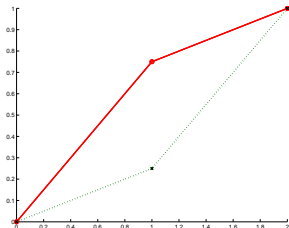
2D-Grafiken

Matlab hat einen reichhaltigen Fundus von Hilfsmitteln zur Visualisierung.

- `plot(x,y)` Zeichnet Polygonzug mit Ecken $P_k = (x_k, y_k)$
Als drittes Argument können zusätzliche Optionen angegeben werden.

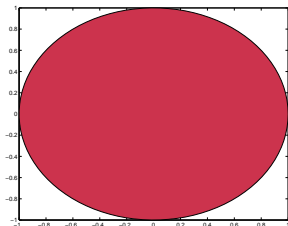
Farbe	Ecken	Linien
b blau	. Punkte	- durchgängig
k schwarz	o Kreise	: gepunktet
r rot	x Kreuze	-- gestrichelt

- `hold on` Verhindert das Löschen des Zeichenfensters vor dem nächsten Plot



```
hold on
plot([0 1 2],[0 3/4 1], 'r-o')
plot([0 1 2],[0 1/4 1], 'k:x')
```

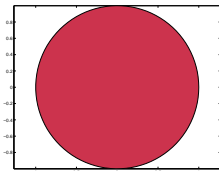
- `fill(x,y,c)` Füllt das Innere des Polygonzuges mit der Farbe `c`



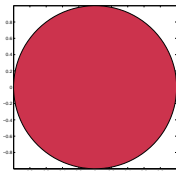
```
t=linspace(0,2*pi);  
fill(cos(t),sin(t),[0.2 0.5 0.25])
```

- `axis` Reguliert mit `equal`, `tight`, `off` die Darstellung der Achsen

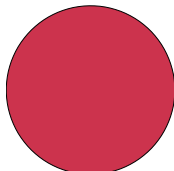
axis equal



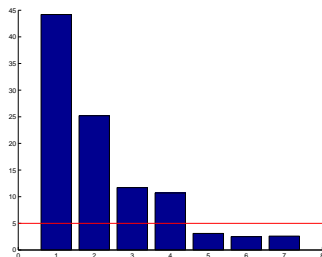
axis equal tight



axis equal off



- `bar(y)` Balkengraphik der Daten `y`



```
hold on
```

```
bar([44.2 25.2 11.7 10.7 3.1 2.5 2.6])
```

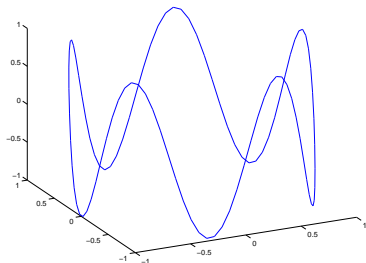
```
plot([0 8],[5 5], 'r-')
```

3D-Grafiken

Ein dreidimensionaler Plot ist entweder eine Kurve (eindimensional), eine Oberfläche (Mesh, zweidimensional) oder ein Körper (dreidimensional).

Plot von Kurven

- `plot3(x,y,z)` Polygonpunkte in Vektoren x,y,z



```
t=linspace(0,2*pi);  
plot3(cos(t),sin(t),cos(5*t))
```

Gitter

Matlab interpretiert x-, y- und z-Werte, die jeweils als Matrizen angegeben werden, als Gitter. Dieses kann relativ einfach mit meshgrid erzeugt werden:

```
>> x=1:5, y=2:4
```

```
x = 1 2 3 4 5      y = 2 3 4
```

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

```
X =
```

```
1 2 3 4 5
```

```
1 2 3 4 5
```

```
1 2 3 4 5
```

```
Y =
```

```
2 2 2 2 2
```

```
3 3 3 3 3
```

```
4 4 4 4 4
```

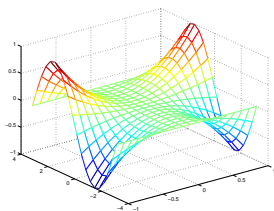
Plot von Flächen

Mit

```
x=linspace(-1,1);
y=linspace(-pi,pi);
[X,Y]=meshgrid(x,y);
Z=X.^2.*sin(Y);
```

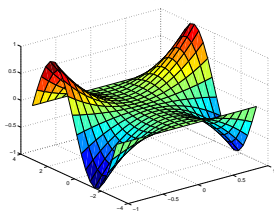
ergeben folgende Anweisungen die gezeichneten Plots.

`mesh(X,Y,Z)`



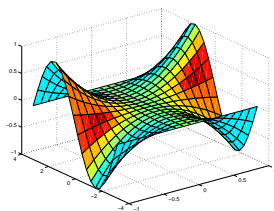
Drahtgitter

`surf(X,Y,Z)`



Kacheln

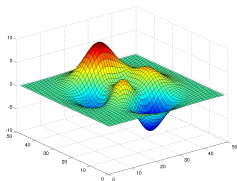
`surf1(X,Y,Z)`



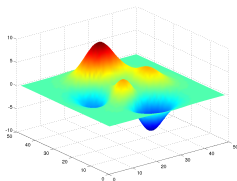
Beleuchtung

Optionen

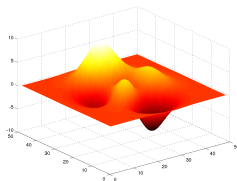
Plot
surf(peaks)



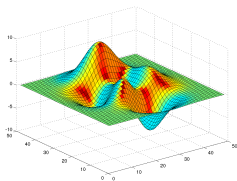
Gitternetz
shading interp



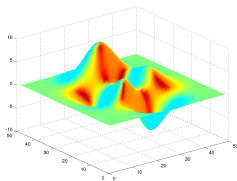
Farbe
colormap hot



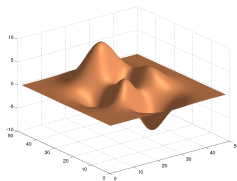
surf1(peaks)



shading interp



colormap copper



Ausblick

Viele Zeichenbefehle liefern ein Handle auf das gezeichnete Objekt zurück. Mit `get(handle)` bekommt man eine Auflistung der zugehörigen Eigenschaften, die mit `set(handle,Eigenschaft,Einstellung)` gesetzt werden können.

Matlab kann noch viel mehr. Eine kurze Einführung bietet `demodemos`. Für einen Überblick müssen aber auch die zahlreichen Toolboxen beachtet werden.