

Rechnergestützte Analysis und Analytische Geometrie

Torsten Krüger - Max Kontak - Jörg Frohne

Department Mathematik, WiR AG, Universität Siegen

Stand: 11. April 2013



① Allgemeines

② Octave

③ Formelsammlung Lineare Algebra



Über dieses Skript

- Einblick in die Grundlagen von `octave`
- Basis für einen möglichst einfachen Einstieg in die Programmierung
- Die für die ersten Semester des Mathematik-Studiums benötigten Programmierkenntnisse werden bereitgestellt
- Pragmatischer Ansatz: Konzentration auf Grundlagen um Fundament für weiterführende Kurse zu legen



Über diesen Kurs

- Kurs insbesondere für Studienanfänger der Mathematik
- Für Programmieranfänger und Fortgeschrittene
- Kompakte Einführung in `octave`
- Vielfach Stoff aus der Linearen Algebra und Analysis der gymnasialen Oberstufe am Computer zu realisieren
- Für Leistungsnachweis: Regelmäßige, aktive Mitarbeit und abschließendes Kolloquium

[http://www.uni-siegen.de/fb6/wir/veranstaltungen/
softwarepraktikum_ss2013](http://www.uni-siegen.de/fb6/wir/veranstaltungen/softwarepraktikum_ss2013)



Konvention für Code-Beispiele

In diesem Skript werden Code-Beispiele wie folgt dargestellt:

- Nach einem > finden sich die zu machenden Eingaben
- Darunter werden dann – falls nötig – die entsprechenden Ausgaben aufgeführt



Kurzeinführung in die Linux-Shell

Standardmäßig befindet man sich in `/home/BENUTZERNAME` (analog zu „Eigene Dateien“ unter Windows).

<code>>ls</code>	"List", gibt kurz den Inhalt des aktuellen Verzeichnisses aus
<code>>cd Name</code>	"Change Directory", ins Verzeichnis Name wechseln
<code>>cd ..</code>	einen Ordner nach oben wechseln
<code>>mkdir Name</code>	"Make Directory", neues Verzeichnis erstellen
<code>>rm Name</code>	"Remove", Datei Name löschen
<code>>nameprog</code>	startet das (installierte) Programm nameprog
<code>>nameprog &</code>	startet das (installierte) Programm nameprog und gibt shell frei



Allgemeine Informationen zu octave

Was ist octave?

- Interaktive Skriptsprache z. B. für numerische Berechnungen, größtenteils kompatibel zu Matlab
- Freie Software für alle gängigen Systeme
- Kommandozeilenbasiert

Nützliche Links

- <http://www.gnu.org/software/octave>
- <http://www.gnu.org/software/octave/doc/interpreter>
- <http://math.jacobs-university.de/oliver/teaching/tuebingen/octave/octave/index.html>



Eingabekonventionen

- Kommentare werden in `octave` durch ein `%` eingeleitet
- Als Rechenoperatoren werden wie gewohnt `+`, `-`, `*`, `/` verwendet
- Als Zuweisungsoperator dient `=`
- Trennung von Befehlen in einer Zeile durch `,`
- Trennung von Befehlen in einer Zeile und Unterdrückung der Bildschirmausgabe durch `;`
- Für einen Zeilenumbruch innerhalb eines Befehls `...` am Zeilenende verwenden
- **Achtung:** Groß- und Kleinschreibung werden berücksichtigt!



Mathematische Operationen

Arithmetische Operationen

```
> x=1;y=2;           % Wertzuweisung. Wichtig: Der
                    % Name der Variablen steht
                    % jeweils links!

> x+y,x-y,x*y,x/y   % Eingabe der Befehle
ans = 3              % Ausgabe, die Befehle wurden
ans = -1             % durch ein Komma getrennt
ans = 2
ans = 0.50000
```

Quadratwurzel und Absolutbetrag

```
> abs(-1)+sqrt(x+y) % x und y von oben
ans = 2.7321
```



Mathematische Operationen

Trigonometrische und hyperbolische Funktionen

```
> sin(x*y), sinh(x*y),...    % Zeilenumbruch mit ...  
> z=asin(sin(x*y))          % Umkehrfunktionen
```

Exponentialfunktion, Logarithmus, Potenzen

```
> exp(x), x^y, log(y)        % Natürlicher Logarithmus  
> log2(x), log10(y)         % zur Basis 2 und 10
```



Zahlen in Octave

Reelle Zahlen werden durch Gleitkommazahlen repräsentiert.

Beispiel

$1/\sqrt{2}$ in verschiedenen Formaten.

```
> format short           % Standardformat
0.70711
> format short e        % Exponentialdarstellung
7.0711e-01
> format long
0.707106781186547
> format long e
7.07106781186547e-01
```

Das Format `long e` entspricht in etwa der Gleitkommadarstellung, die im Computer hinterlegt ist.



Zahlen in Octave

Komplexe Zahlen können wie üblich als Kombination zweier reeller Zahlen bzw. Gleitkommazahlen x und y in der Form $x+i*y$ oder $x+j*y$ dargestellt werden.

Beispiele

```
>z=x+i*y
```

```
z = 1 + 2i
```

```
>w=x+j*y
```

```
w = 1 + 2i
```

```
> i*i
```

```
ans = -1
```

```
>log(x-y)
```

```
ans = 0.00000 + 3.14159i
```



Hinterlegte Konstanten

In `octave` sind einige wichtige mathematische Konstanten hinterlegt:

Beispiele

```
> e,pi,i,e^(i*pi)
ans = 2.7183
ans = 3.1416
ans = 0 + 1i
ans = -1.0000e+00 + 1.2246e-16i
```

An der letzten Ausgabe erkennt man, dass das Rechnen mit Gleitkommazahlen immer mit Ungenauigkeiten verbunden ist.

Eine spezielle Konstante ist `eps`. Sie gibt die Rechengenauigkeit des Systems an.



Erstellung von Matrizen

Kleine Matrizen können direkt eingegeben werden

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

```
A =
```

```
1    2    3
```

```
4    5    6
```

```
7    8    9
```

Die Eingabe erzeugt die Matrix A, darunter ist die Bildschirmausgabe zu sehen. Nach Verwendung des Zuweisungsoperators = ist für `octave` in dieser Variable die Matrix hinterlegt. Zeilen werden durch „;“, Spalten durch Kommas oder Leerzeichen voneinander getrennt.



Rechnen mit Matrizen

Wir erzeugen eine weitere 3×3 -Matrix durch

```
> B=[1 1 1;1 -1 1;1 0 1];
```

Die Bildschirmausgabe wird durch das Semikolon unterdrückt.
Jetzt können wir mit diesen Matrizen rechnen.

Beispiel

```
> S=A+B;
```

```
> D=A-B;
```

```
> P=A*B;
```

```
> P
```

```
P =
```

```
     6     -1     6
    15     -1    15
    24     -1    24
```



Zugriff auf Teile einer Matrix

Der Zugriff auf einzelne Bereiche einer Matrix A erfolgt allgemein über $A(\text{Zeile}, \text{Spalte})$.

Beispiele

```
> A(1,2)=A(1,3)+B(3,1)
```

```
A =
```

```
1 4 3
```

```
4 5 6
```

```
7 8 9
```

```
> A(3, [1 3])
```

```
ans =
```

```
7 9
```

```
> A(:,1)
```

```
ans =
```

```
1
```

```
4
```

```
7
```

Mit dem Doppelpunkt wird eine ganze Zeile bzw. Spalte ausgewählt.



Transponierte/Adjungierte Matrix

Das Hochkomma ' liefert für Matrizen die adjungierte Matrix.
Für Matrizen über \mathbb{R} entspricht dies der Transponierten.

Beispiele

```
> C = [1 2 3; 9 8 7]; > D = [i 5 3+i; 4*j 2+7*i 2];
> C' > D'
ans = ans =
    1    9    0 - 1i    0 - 4i
    2    8    5 - 0i    2 - 7i
    3    7    3 - 1i    2 - 0i
```

Wie man sieht, werden komplexe Einträge komplex konjugiert.
Ansonsten liefert `transpose(A)` die transponierte der Matrix A.



Elementweise Operationen

Soll eine Operation elementweise ausgeführt werden, so wird dies durch einen Punkt vor dem Operator erreicht.

Beispiel: Multiplikation

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

```
A =
```

```
1 2
```

```
3 4
```

```
5 6
```

```
> A.*A
```

```
ans =
```

```
1 4
```

```
9 16
```

```
25 36
```

Der Versuch $A*A$ zu berechnen liefert eine Fehlermeldung, da die Dimensionen für eine Matrixmultiplikation nicht zusammenpassen.



Lineare Gleichungssysteme

Lineare Gleichungssysteme der Form $Ax = b$ können über den Befehl `A\b` gelöst werden.

Beispiel

```
> A=[1 1 1;1 -1 1];b=[1 0]';  
> x=A\b  
x =  
  
0.25  
0.5  
0.25
```

Intern greift `octave` dabei auf angepasste Löser für lineare Gleichungssysteme zurück. Ist A eine reguläre quadratische Matrix, kann alternativ auch der Befehl `x=inv(A)*b` genutzt werden.



Sonstige Matrixoperationen

Hier eine Übersicht über einige weitere allgemeine Matrixoperationen:

```
A(:,k)=[]      % Löscht k-te Matrixspalte
A(k,:)=[]     % Löscht k-te Matrixzeile
diag(A,k)     % Extrahiert k-te Nebendiagonale von A;
              % im Falle eines Vektors ergibt sich
              % eine (Neben-)Diagonalmatrix
inv(A)        % Gibt, falls existent,
              % die Inverse von A an
rank(A)       % Rang der Matrix A
size(A)       % Größe der Matrix
det(A)        % Determinante einer quadr. Matrix
rref(A)       % Stellt Zeilenstufenform von A her
```



Eingabe spezieller Matrizen

Große Matrizen lassen sich nicht mehr gut von Hand eingeben.
In solchen Fällen sind etwa folgende Funktionen nützlich:

```
ones(n,m)      % (n,m)-Matrix aus lauter Einsen  
eye(n,m)       % (n,m)-Matrix mit Einsen auf der  
               % Diagonalen  
zeros(n,m)     % (n,m)-Matrix aus lauter Nullen
```

Auch $n \times m$ -Matrizen aus Zufallszahlen können erzeugt werden:

```
rand(n,m)      % Gleichverteilte Zufallszahlen  
randn(n,m)     % Normalverteilte Zufallszahlen
```



Vektoren

Vektoren sind Matrizen mit nur einer Zeile bzw. Spalte. Mit vorgegebenen Zahlen a , b und Schrittweite s erzeugt man über $v=a:s:b$ einen Vektor mit Elementen a , $a + s$, $a + 2s$, \dots , $a + ks$. Dabei gilt $a + ks \leq b$, falls $s > 0$ (analog falls $s < 0$).

Beispiel

Dieser Vektor kann dann z. B. so weiterverwendet werden:

```
> w=1:5:50
```

```
w =
```

```
1    6   11   16   21   26   31   36   41   46
```

```
> v=1:3:11
```

```
v =
```

```
1    4    7   10
```

```
> w(v)
```

```
ans =
```

```
1   16   31   46
```



Matrizen und Vektoren

Weitere Beispiele

```
> v=1:4           % Voreingestellte Schrittweite ist 1
v =
    1    2    3    4

> M=diag(v)
M =
    1    0    0    0
    0    2    0    0
    0    0    3    0
    0    0    0    4

> N=M+ones(size(M))
N =
    2    1    1    1
    1    3    1    1
    1    1    4    1
    1    1    1    5
```

`diag` erzeugt hier – wie oben angekündigt – eine Diagonalmatrix mit v auf der Diagonalen.



Matrizen und Vektoren

Weitere Beispiele

```
diag(N,1)           % Gibt die Elemente auf der ersten
ans =              % (oberen) Nebendiagonalen aus
    1
    1
    1
diag(v(1:3),-1)    % Bildet eine Matrix mit
ans =              % den Elementen von v(1:3) auf
    0    0    0    0 % der ersten unteren Nebendiagonalen
    1    0    0    0
    0    2    0    0
    0    0    3    0
```



Vektoren

Weiterhin ist der Befehl `linspace` überaus nützlich. So erzeugt `linspace(a,b,n)` einen Vektor mit erstem Element a , $n-2$ äquidistanten Zwischenelementen und letztem Element b .

Beispiel

```
> v=linspace (0,1,4)
```

```
v =
```

```
0.00000    0.33333    0.66667    1.00000
```



Matrizen

Größere Matrizen können durch die üblichen Operationen, aber auch durch die Eingabe ganzer Teilmatrizen erzeugt werden.

Beispiel

```
> I=eye(3);           % size(I) gibt das  
> O=zeros(size(I));  % Format der Matrix aus  
> M=[I O;ones(size(O)) I]
```

M =

```
 1  0  0  0  0  0  
 0  1  0  0  0  0  
 0  0  1  0  0  0  
 1  1  1  1  0  0  
 1  1  1  0  1  0  
 1  1  1  0  0  1
```



Weitere nützliche Funktionen

<code>sum(v)</code>	% Summe der Elemente von v
<code>prod(v)</code>	% Produkt der Elemente von v
<code>norm(v,p)</code>	% p-Norm von v
<code>length(v)</code>	% Anzahl der Elemente von v
<code>fliplr(v)</code>	% Umkehr der Reihenfolge bei Zeilenvektor
<code>flipud(v)</code>	% Umkehr der Reihenfolge bei Spaltenvektor

Diese Funktionen können auch auf Matrizen angewendet werden.

Für nähere Informationen kann die `help`-Funktion angewendet werden.



Die help-Funktion

Beispiel

```
> help fliplr
'fliplr' is a function from the file /usr/local/share/octave/3.4.2/m/general/fliplr.m
```

```
-- Function File: fliplr (X)
```

```
Return a copy of X with the order of the columns reversed. In
other words, X is flipped left-to-right about a vertical axis. For
example:
```

```
fliplr ([1, 2; 3, 4])
=>  2  1
    4  3
```

```
Note that 'fliplr' only works with 2-D arrays. To flip N-D arrays
use 'flipdim' instead.
```

```
See also: flipud, flipdim, rot90, rotdim
```



Benutzerinteraktion

Überblick über nützliche Funktionen

<code>input</code>	<code>% Wartet auf interaktive Eingabe von Daten</code>
<code>disp</code>	<code>% Bildschirmausgabe</code>
<code>pause</code>	<code>% Pausieren bis Tastendruck</code>
<code>pause(n)</code>	<code>% Pausiere n Sekunden</code>



Benutzerinteraktion

Interaktive Eingabe von Daten

```
> n=input('Geben Sie eine ganze Zahl ein: ');  
Geben Sie eine ganze Zahl ein: 2  
> n  
n = 2
```

Ausgabe von Text

```
> disp('Programm zur Demonstration')  
Programm zur Demonstration
```



Umgang mit Variablen

Löschen aller Variablen

Der Befehl `clear` löscht alle Variablen. Die Variable `var` kann explizit mit `clear var` gelöscht werden.

Speichern und Laden von Variablen

Mit den Befehlen `save` und `load` können Variablen zur späteren Nutzung gespeichert und dann wieder geladen werden.

Alle Variablen anzeigen

Durch Aufruf der Funktion `who` erhält man eine Liste aller aktuellen Variablen.



Umgang mit Variablen

Beispiel

```
> a=5;b=7;A=[1 2;3 4];  
> save 'Daten.mat' a b A  
> clear  
> who  
> load 'Daten.mat'  
> who  
Variables in the current scope:  
A a b  
> clear a  
> who  
Variables in the current scope:  
A b
```



Umgang mit Variablen

Achtung!

Werden Variablen aus einer Datei geladen, so erhalten sie den gleichen Namen wie vor dem Abspeichern. Eventuell vorhandene Variablen gleichen Namens werden überschrieben.

Bemerkung

Das Abspeichern von Variablen wird insbesondere dann sinnvoll, wenn ihre Berechnung viel Zeit benötigt, und diese wiederholt weiterverwendet werden sollen.



Skripte und Funktionen

Abspeichern von `octave`-Befehlen

- Speicherung in Dateien mit der Endung `.m`
- Zwei verschiedene Arten: **Skripte** und **Funktionen**
- Bearbeitung mit jedem Texteditor möglich, z. B.
 - Windows:** Notepad, Notepad++
 - Linux:** gedit, emacs
 - Mac:** ?
- Es gibt auch graphische Oberflächen für `octave`, die einen Editor enthalten, unter Linux z. B. `qtoctave`

Um eine Datei zu verwenden, muss `octave` in dem Verzeichnis gestartet oder per `cd` dorthin navigiert werden, in dem sich die Datei befindet.



Skripte werden von `octave` von oben nach unten durchlaufen, als hätte man die darin enthaltenen Befehle direkt auf der Kommandozeile eingegeben.

Beispiel

Inhalt der Datei `fibonacci.m`:

```
% Programm zur Berechnung der k-ten Fibonacci-Zahl
%  $a_{k+1} = a_k + a_{k-1}$ ,  $a_1=a_2=1$ 
k=input('Geben Sie eine natuerliche Zahl ein: ');
x=[1 1]';      % Anfangswerte der Rekursion
M=[1 1;1 0];  % Rekursionsmatrix
x=M^(k-2)*x;  % Ermittlung der Folgenglieder
disp(x(1));   % Ausgabe der k-ten Fibonacci-Zahl
```

Der Aufruf erfolgt dann über den Befehl `fibonacci`, wenn `octave` im `fibonacci.m` enthaltenden Ordner ausgeführt wird.



Funktionen

Mit so genannten `function`-Files können selbst definierte Funktionen frei aufgerufen werden.

Allgemeiner Aufbau einer Funktion

```
% Kommentare zur Funktion
% werden von help angezeigt
function [Ausgabevektor] = Name(Argumentenvektor)

Anweisungen;                                % Funktionskörper

end
```

`Name` legt den Namen der Funktion fest, unter der die Funktion dann auch aufgerufen werden kann. Die Datei **muss** unter dem Namen `Name.m` abgespeichert werden.



Beispiel

Inhalt der Datei fib.m:

```
% Funktion zur Berechnung der k-ten Fibonacci-Zahl
% mit den Rekursionsanfangswerten x1, x2
function x=fib(k,x1,x2)
    a=[x2 x1]';
    M=[1 1;1 0];
    y=M^(k-2)*a;
    x=y(1);
endfunction
```

Aufruf der Funktion etwa über `fk=fib(k,x1,x2)`, wobei die Argumente direkt mit angegeben werden.



Regeln für die Erstellung von Funktionen

- Ein `function`-File ist für `octave` erkennbar am Schlüsselwort `function` vor der ersten Anweisung
- Der Funktionskörper wird üblicherweise durch `end` oder `endfunction` abgeschlossen
- Funktionsname und Dateiname müssen übereinstimmen
- Funktionen müssen weder Ein- noch Ausgabevektoren haben
- Funktionen können andere Funktionen enthalten/aufrufen
- Das `function`-File muss grundsätzlich im selben Ordner liegen, in dem `octave` aufgerufen wird
- Durch `help Name` werden die Kommentare in den ersten Zeilen ausgegeben
- Hat der Ausgabevektor $N > 1$ Elemente und es werden nur $k < N$ Elemente durch den Funktionsaufruf abgefragt, so werden die ersten k Elemente übergeben



Skripte vs. Funktionen

Wann setzt man Skripte, wann Funktionen ein?

Funktionen setzt man u.a. ein,

- wenn die gleiche Rechnung für verschiedene Eingabe-Parameter durchgeführt werden soll.
- wenn es sein kann, dass man die Funktion später nochmal wiederverwenden will.

Skripte setzt man insbesondere ein,

- um kleinere alleinstehende Programme zu realisieren.
- als Rahmen für grössere Programme.



Logische und Vergleichsoperationen

Logische Operationen

- & Logisches UND
- | Logisches ODER
- ~ Logisches NICHT

Vergleichsoperationen

- == gleich
- ~= ungleich
- > größer
- >= größer/gleich
- < kleiner
- <= kleiner/gleich

Ab octave-Version 3.4.0 wird das Symbol `~` auch zum Ignorieren von Funktionsausgaben benutzt. So bewirkt der Aufruf `[~,n] = size(A)`, dass in der Variable `n` die Anzahl der Spalten gespeichert, die Anzahl der Zeilen aber verworfen wird.



Logische und Vergleichsoperationen

Beispiele

```
>x=1;y=0;
>x&y,x|y,~y
ans = 0
ans = 1
ans = 1
>x==x&y,x~=x|y
ans = 0
ans = 0
>x>=y,x<y
ans = 1
ans = 0
```

In `octave` steht der Wert 0 für Falsch, jeder andere Wert – insbesondere 1 – für Wahr.

Zur Eingabe können auch die Variablen `true` und `false` verwendet werden.



Übersicht

An Kontrollstrukturen stehen in `octave` insbesondere zur Verfügung:

- `for`-Schleifen,
- `while`-Schleifen,
- `if`-Anweisungen und
- `switch`-Anweisungen.



for-Schleifen

Allgemeiner Aufbau

```
for Index=Indexbereich  
    Anweisungen  
end
```

Beispiel

```
> x=0;  
> for i=1:10  
>     x+=i;  
> end  
> disp(x)  
55
```

Das Beispiel berechnet die Summe der ersten zehn natürlichen Zahlen. Einfacher wäre `sum(1:10)` gewesen.



while-Schleifen

Allgemeiner Aufbau

```
while Abbruchbedingung  
    Anweisung;  
end
```

while-Schleifen werden gerne verwendet, wenn die Anzahl der Durchläufe nicht von vornherein bekannt ist.

Beispiel

```
> x=0;i=1;  
> while (i<11)  
>     x+=i;i+=1;  
> end  
> disp(x)  
55
```

Bemerkung

Guter Programmier-Stil: **Einrückungen** vornehmen, um Code inner- und außerhalb von Schleifen zu unterscheiden!



Schleifen

Achtung!

Die Verwendung von Schleifen in `octave` ist (rechen-)zeitaufwändig. In Programmen sollten – wenn möglich – die in `octave` eingebauten Funktionen verwendet werden. Diese sind vorwiegend direkt in `C++` oder `Fortran` programmiert und optimiert, und deswegen performanter.

Beispiel

Verwendung von `sum(1:10)` statt der Schleifen aus den Beispielen auf den vorherigen Folien.



if-Anweisung

Allgemeiner Aufbau

```
if Bedingung1
    Anweisung1;
elseif Bedingung2
    Anweisung2;
...
else
    AnweisungSonst;
end
```

Es können beliebig viele (oder auch kein) `elseif` verwendet werden.

Beispiel

```
> if (b<0)
>   disp('b negativ');
> elseif (b==0)
>   disp('b ist Null');
> else
>   disp('b positiv');
> end
```

Das Beispiel überprüft, ob die Zahl `b` positiv, negativ oder Null ist.



switch-Anweisung

Allgemeiner Aufbau

```
switch Variable
  case Fall1
    Anweisung1;
  case Fall2
    Anweisung2;
  ...
  otherwise
    AnweisungSonst;
end
```

Beispiel

```
> switch vorz
>   case 1
>     disp('Positiv');
>   case 0
>     disp('Null');
>   case -1
>     disp('Negativ');
>   otherwise
>     disp('Fehler');
> end
```

Es können beliebig viele case-Anweisungen benutzt werden. Stimmt keine von diesen, so werden die Anweisungen bei otherwise ausgeführt.



Schleifen

`break` unterbricht den Schleifendurchlauf zum gewählten Zeitpunkt, `continue` springt zum Ende des aktuellen Durchlaufs.

Beispiele

break-Anweisung

```
> i=0;
> while true
>   i++;
>   if (i>10)
>     break;
>   end
> end
> disp(i);
11
```

continue-Anweisung

```
> v=zeros(1,5);
> for k=1:length(v)
>   if ((k==2) | (k==4))
>     continue;
>   end
>   v(k)+=k;
> end
> v
v =
1 0 3 0 5
```



Funktionen und Schleifen

Um aus einer Funktion herauszuspringen gibt es den Befehl `return`.

Beispiel

Inhalt der Datei

```
nulltest.m
```

```
function val=nulltest(x)
    val=0;
    for i=1:length(x)
        if x(i)==0
            val=1;
            return;
        endif
        disp('Keine Null');
    endfor
endfunction
```

Aufruf der Funktion:

```
> nulltest([1 0 1])
```

```
Keine Null
```

```
ans = 1
```

```
> x=nulltest([1 1 1])
```

```
Keine Null
```

```
Keine Null
```

```
Keine Null
```

```
x = 0
```



Grafikausgabe

octave nutzt standardmäßig `gnuplot` für die Grafikausgabe. Sowohl 2D- als auch 3D-Ausgaben sind möglich. Wahlweise ist auch die Erzeugung einer PDF-Datei oder anderer Formate möglich (siehe `help print`).

2D-plot-Befehl

`plot(x,y)` trägt die Vektoren `x` und `y` gegeneinander in einem 2D-Koordinatensystem auf.

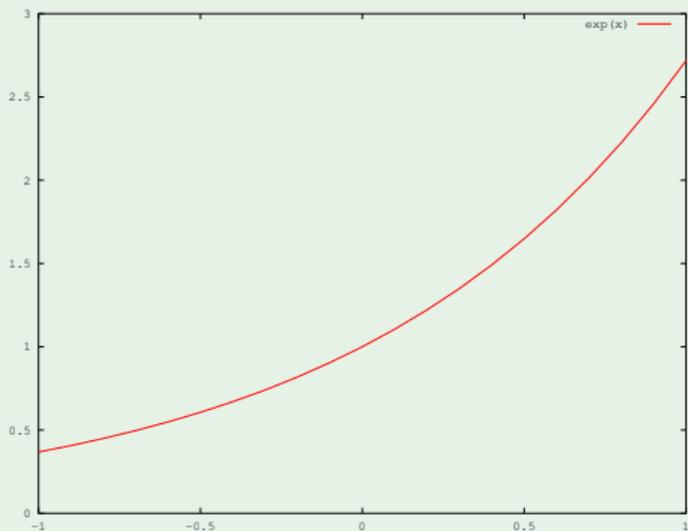
Optional können auch Einstellungen wie Farbe und Linienart oder auch mehrere Kurven gleichzeitig eingegeben werden (siehe `help plot`), z. B. `plot(x1,y1,'optional',x2,y2,...)`.

Der Befehl `hold on|off` legt fest, ob alte Grafikausgaben erhalten bleiben oder überschrieben werden. `clf` löscht alle grafischen Ausgaben.



Beispiel für den Befehl `plot`

```
> x=-1:.1:1;  
> y=exp(x);  
> plot(x,y,'r');  
> plot(x,y,'r');  
> plot(x,y,'r');  
> plot(x,y,'r');
```



3D-plot-Befehl

Kurven im \mathbb{R}^3 (Abbildungen $[a, b] \rightarrow \mathbb{R}^3$) können über den `plot3`-Befehl visualisiert werden. Die Syntax ist ähnlich zum 2D-Fall: `plot3(x1,y1,z1,'optional',x2,y2,z2,...)`.

Beispiel

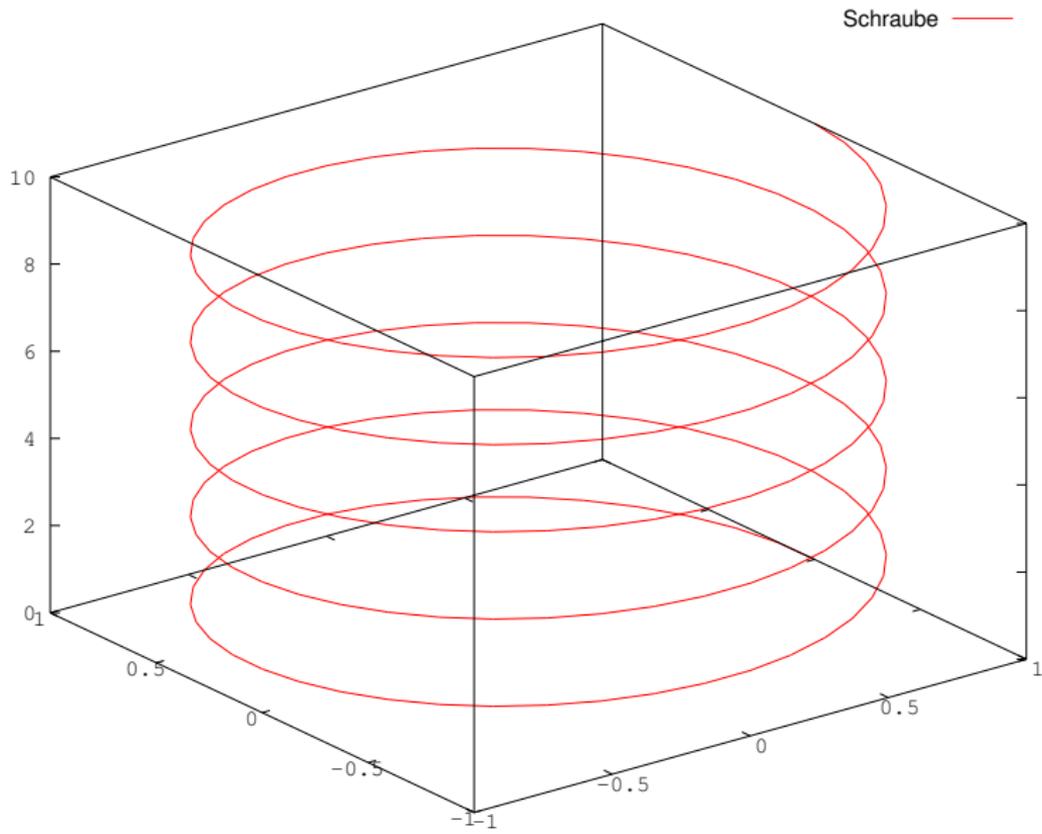
Abbildung $[0, 10] \rightarrow \mathbb{R}^3$, $t \mapsto (\cos(\pi t), \sin(\pi t), t)^T$.

```
> t=0:.05:10;
```

```
> plot3(cos(pi*t),sin(pi*t),t,';Schraube;r');
```



Grafikausgabe



Grafikausgabe

mesh-Befehl

3D-Plots von Flächen (Abbildungen $\mathbb{R}^2 \supset U \rightarrow \mathbb{R}^3$) können mit dem `mesh`-Befehl erzeugt werden. Dazu muss ein 2D-Gitter als Definitionsbereich mit dem `meshgrid`-Befehl erzeugt werden.

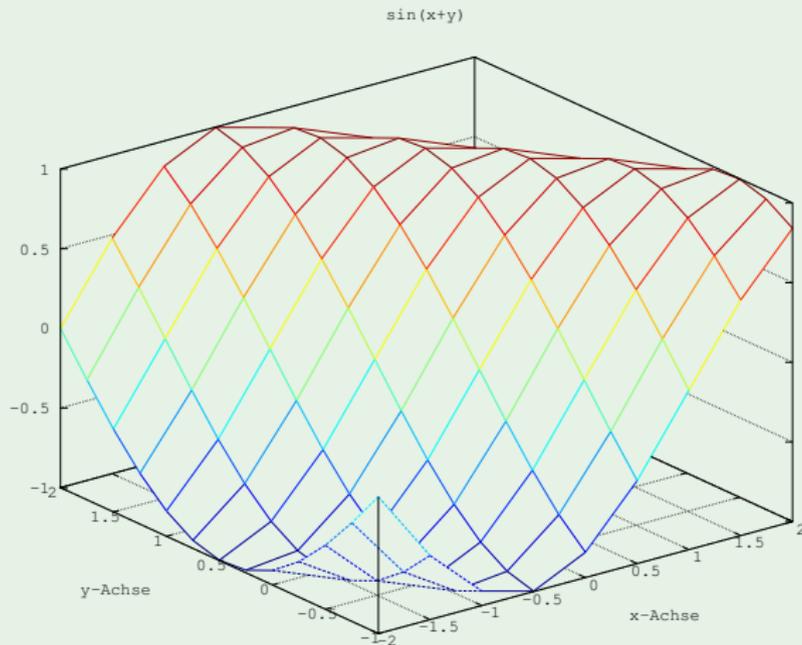
Beispiel für den `mesh`-Befehl

Abbildung $[-2, 2] \times [-1, 1] \rightarrow \mathbb{R}^3$, $(x, y) \mapsto (x, y, \sin(x + y))^T$.

```
> x=-2:.5:2; % Bereich x-Achse
> y=-1:.25:2; % Bereich y-Achse
> [X,Y]=meshgrid(x,y); % x-y-Gitter erzeugen
> z=sin(X+Y); % Zu plottende Funktion
> mesh(x,y,z); % Erzeugung des Bildes
> xlabel('x-Achse'); ylabel('y-Achse'); % Achsenbeschriftung
> title('sin(x+y)'); % Titel der Grafik
```



Ausgabe des Beispiels



Informationen zu `gnuplot`

- Bewährtes Plotprogramm
- Freie Software für alle Plattformen
- Kommandozeilenbasiert
- Wird standardmäßig von `octave` für grafische Ausgaben genutzt

Weitere Informationen unter <http://www.gnuplot.info/>



Funktionsauswertungen

Die in einer Datei `func.m` hinterlegte Funktion kann über den Befehl `feval` direkt ausgewertet werden.

Beispiel

Datei `func.m`

```
> function y=func(x)
>   y=x.^2;
> end
```

Anwendung

```
> X=feval('func',2)
X = 4
```

Außerdem können auch Zeichenfolgen direkt über die Funktion `eval` wie Code ausgewertet werden.

Beispiel

```
> eval('bsp = asinh(exp(200))');
bsp = 200.69
```



Funktionsauswertungen

Weiterhin können auch *function handles* genutzt werden. Diese sind besonders nützlich, wenn einer Funktion als Argument ebenfalls eine Funktion übergeben werden soll.

Allgemeiner Aufbau

```
Name = @(Arg) Ausdruck
```

Beispiel

```
> f=@(t) sin(t);  
> y=f(pi)  
y = 1.2246e-16
```



Funktionsauswertungen

Besonders einfach ist in `octave` die Verwendung von Polynomen. Diese werden durch einen Vektor, der ihre Koeffizienten in absteigender Reihenfolge enthält, identifiziert.

Beispiel

```
> p=[-2 -1 0 1 2]; % Eingabe durch Koeffizienten
> polyout(p,'x'); % Darstellung als Text
-2*x^4 - 1*x^3 + 0*x^2 + 1*x^1 + 2
> polyval(p,1:3) % Auswertung an mehreren Stellen
ans =
    0   -36  -184
```

Weitere Befehle zum Umgang mit Polynomen: `conv`, `deconv`, `polyderiv`, `polyint`, `roots` (Informationen über `help`)



Formelsammlung Lineare Algebra

Satz (Allgemeine Lösung eines LGS)

Es sei y eine Lösung des linearen Gleichungssystems $Ax = b$, wobei $b \neq 0$ gelten soll. Weiter sei

$$\ker(A) := \{x \mid Ax = 0\}$$

der Kern der Matrix A . Dann ist die Lösungsmenge L dieses linearen, inhomogenen Gleichungssystems gegeben durch

$$L = \{y + x_0 \mid x_0 \in \ker(A)\}.$$

Insbesondere ist ein lineares Gleichungssystem genau dann eindeutig lösbar, wenn das zugehörige homogene System nur die triviale Lösung besitzt.



Formelsammlung Lineare Algebra

Definition (Standardskalarprodukt)

Es seien $x = (x_1, \dots, x_n)$ und $y = (y_1, \dots, y_n)$ Vektoren aus dem \mathbb{R}^n . Dann ist

$$\langle x, y \rangle := \sum_{i=1}^n x_i y_i$$

das Standardskalarprodukt. Das Skalarprodukt hat unter anderem folgende Eigenschaften:

- $\langle x, x \rangle = \sum_{i=1}^n x_i^2 = \|x\|^2$.
- $\langle x, y \rangle = \|x\| \cdot \|y\| \cdot \cos(\alpha)$, wobei $\alpha := \angle(x, y)$.
- Für $x, y \neq 0$: $x \perp y \Leftrightarrow \alpha = \frac{\pi}{2} \Leftrightarrow \langle x, y \rangle = 0$.

Der Aufruf in `octave` erfolgt über `x'*y`, falls `x` und `y` Spaltenvektoren sind.



Formelsammlung Lineare Algebra

Definition (Kreuzprodukt)

Es seien $x = (x_1, x_2, x_3)$ und $y = (y_1, y_2, y_3)$ Vektoren aus dem \mathbb{R}^3 . Dann ist

$$x \times y := \begin{pmatrix} x_2 y_3 - x_3 y_2 \\ x_3 y_1 - x_1 y_3 \\ x_1 y_2 - x_2 y_1 \end{pmatrix}$$

das Kreuzprodukt dieser Vektoren. Für dieses gilt:

$$x \times y = \|x\| \cdot \|y\| \cdot \sin(\alpha) \cdot n,$$

wobei $\alpha := \angle(x, y)$ und n senkrecht zu x und y ist, so dass $\|n\| = 1$ und die drei Vektoren ein rechthändiges Koordinatensystem bilden.

Der Aufruf in octave erfolgt über `cross(x, y)`.



Formelsammlung Lineare Algebra

Definition (Parameterdarstellung einer Geraden)

Die Parameterdarstellung einer Geraden g im Raum \mathbb{R}^3 lautet

$$g(t) = \begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix} + t \cdot \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix}.$$

Dabei durchläuft der Parameter t die reellen Zahlen.



Formelsammlung Lineare Algebra

Definition (Parameterdarstellung einer Ebene)

Die Parameterdarstellung einer Ebene im \mathbb{R}^3 lautet

$$E(s, t) = \begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix} + s \cdot \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} + t \cdot \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} .$$

Die Parameter s und t durchlaufen die reellen Zahlen, die Spannvektoren u und v müssen linear unabhängig sein.



Formelsammlung Lineare Algebra

Definition (Normalenform und Koordinatendarstellung einer Ebene)

Die Normalenform einer Ebene im \mathbb{R}^3 mit Normalenvektor n lautet

$$\langle n, p \rangle = c$$

mit einer Konstanten $c \in \mathbb{R}$, wobei p ein beliebiger Punkt der Ebene sei. Ausgeschrieben ergibt sich die Koordinatendarstellung der Ebene:

$$n_1 p_1 + n_2 p_2 + n_3 p_3 = c.$$



Formelsammlung Lineare Algebra

Satz (Abstand windschiefer Geraden)

Es seien $g_1(t) = p + tu$, $g_2(s) = q + sv$ windschiefe Geraden im \mathbb{R}^3 und n_0 ein normierter Vektor senkrecht zu u und v . Dann ist

$$d = |\langle p - q, n_0 \rangle|$$

der Abstand der beiden Geraden.

Satz (Abstand von Ebenen und Punkten)

Sei $a_1x_1 + a_2x_2 + a_3x_3 = b$ eine Koordinatendarstellung der Ebene E und $r = (r_1, r_2, r_3)$ ein Punkt im \mathbb{R}^3 . Dann ist der Abstand des Punktes zur Ebene gegeben durch:

$$d_E(r) = \left| \frac{a_1r_1 + a_2r_2 + a_3r_3 - b}{\sqrt{a_1^2 + a_2^2 + a_3^2}} \right|.$$



Formelsammlung Lineare Algebra

Satz (Gram-Schmidtsches Orthogonalisierungsverfahren)

Es seien $v_1, \dots, v_m \in \mathbb{R}^n$ linear unabhängig. Dann kann durch folgendes Verfahren eine Orthonormalbasis $\{u_1, \dots, u_m\}$ von $\text{span}(v_1, \dots, v_m)$ konstruiert werden:

① $u_1 := \frac{v_1}{\|v_1\|}.$

② $\tilde{u}_{k+1} := v_{k+1} - \sum_{i=1}^k \langle v_{k+1}, u_i \rangle u_i, u_{k+1} := \frac{\tilde{u}_{k+1}}{\|\tilde{u}_{k+1}\|}.$

In Schritt 2 läuft k von 1 bis $m - 1$.

