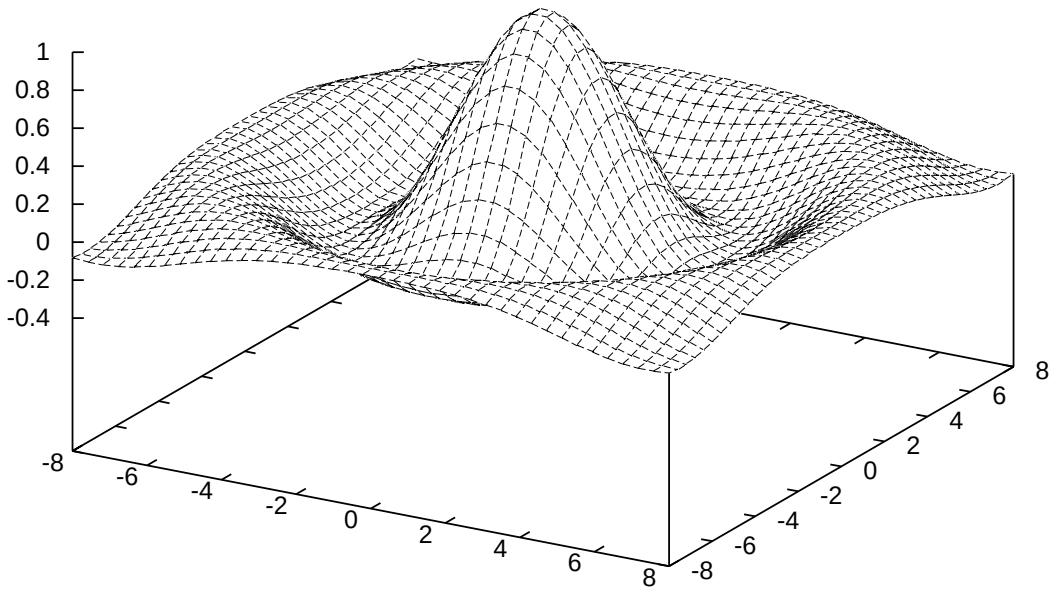
Einführung in

Matlab



nach der Vorlage von Hubert Selhofer

April 2001

Was ist Matlab™?

Matlab™ ist eine interaktive Interpreter Sprache, die einen einfachen Zugang zu grundlegenden numerischen Verfahren – wie bspw. der Lösung linearer Gleichungssysteme oder ähnlichem – bietet.

Der Syntax von **Matlab™** ist dem der Sprache **Octave** sehr ähnlich, d.h. ein **Octave** Programm wird auch von **Matlab™** ausgeführt.

Die Idee hinter diesen beiden Programmen ist, einen möglichst einfachen Zugang zu vielen Standardroutinen der numerischen Mathematik – bspw. aus EISPACK oder LAPACK – zu bieten.

Hilfe!

- `help` listet alle Befehle und internen Variablen auf.

- `help name` gibt Hilfetext zur Variable oder Funktion „name“ aus.

```
>> help eig
```

Variablen

- `varname = expression` weist der Variablen „varname“ den Wert von „expression“ zu.

Es können mehrere Befehle in einer Zeile zusammengefaßt stehen:

- `;` trennt Befehle und unterdrückt eine Ausgabe.

- `,` trennt Befehle, gibt aber Werte aus.

Beispiel:

```
>> x12 = 1/8, long_name = 'Ein String'
x12 = 0.12500
long_name = Ein String
>> sqrt(-1)-i
ans = 0
```

Vorsicht: **Matlab**$^{\mathrm{TM}}$ unterscheidet zwischen Groß- und Kleinbuchstaben.

Es gibt die „üblichen“ Funktionen (`+`, `-`, `*`, `/`, `^`, `sin`, `cos`, `exp`, `acos`, `abs`, ...).

```
>> x = sqrt(2); sin(x)/x
ans = 0.69846
```

Vektoren

Matrizen und Vektoren sind die wichtigsten Grundbausteine zur Programmierung in **Matlab**^{TM}.

Zeilenvektoren:
`v = [ 1 2 3 ]` für $v = (1, 2, 3)$.

Spaltenvektoren:
`v = [ 1; 2; 3 ]` für $v = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$.

Automatische Erzeugung einiger wichtiger Vektoren:

- `Anfang [: Inkrement] : Ende`

Beispiel:

```
>> x = 3:6
x =
  3  4  5  6
>> y = 0:.15:.7
y =
0.00000  0.15000  0.30000  0.45000  0.60000
```

```
>> z = pi:-pi/4:0
z =
3.14159  2.35619  1.57080  0.78540  0.00000
```

Matrizen

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

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

Für einige wichtige $(m \times n)$-Matrizen existieren eigene Befehle. Falls m gleich n ist, muß nur ein Argument angegeben werden.

- `eye(m,n)` erzeugt eine Matrix mit 1-ern auf der Hauptdiagonale. Im Spezialfall $m = n$ ergibt sich die Einheitsmatrix.

- `zeros(m,n)` erzeugt eine Nullmatrix der Dimension $(m \times n)$.

- `ones(m,n)` erzeugt eine Matrix der Dimension $(m \times n)$ mit $a_{ij} = 1 \ \forall i, j$.

- `rand(m,n)` erzeugt eine Zufallsmatrix der Dimension $(m \times n)$ mit gleichverteilten Einträgen. Andere Verteilungen siehe **`help randn`**.

Bestimmung der Dimension einer Variablen:

- `length(v)` gibt die Länge des Vektors v zurück.

- `[Zeilen,Spalten] = size(A)` gibt die Anzahl der Zeilen und Spalten von A zurück.

Grundlegende Operationen

Addition und Subtraktion bzw. Multiplikation erfolgen nun einfach durch $+$, $-$, $*$.

```
>> A = [1 2; 3 4]; B = 2*ones(2,2);
>> A+B, A-B, A*B
ans =
  3  4
  5  6
ans =
  -1   0
   1   2
ans =
   6   6
  14  14
```

- `A'` transponiert und konjugiert A.
- `A.'` transponiert A.

Elementweise Operationen

`+, -, *, /, ^` gibt es auch noch mit einem `.` vorgestellt, dann wird die Verknüpfung elementweise durchgeführt.

Beispiele:

```
>> x = 1:2; A = [1 2; 3 4];
>> [A.^2 A^2]
ans =
   1   4   7  10
   9  16  15  22
>> x.^2-x.*x
ans =
  0  0
```

Matrixmanipulationen

- `v(index)` wählt die durch „index“ spezifizierten Elemente aus dem Zeilen- oder Spaltenvektor „v“ aus.

- `A(index1,index2)` wählt die durch „index1“ und „index2“ spezifizierten Elemente aus der Matrix „A“ aus.

- `reshape(A,m,n)` formt A in eine $(m \times n)$-Matrix um.

- `diag(A)` liefert die Diagonale a_{jj} der Matrix A in einem Spaltenvektor.

- `diag(v[,k])` erzeugt eine Matrix mit dem Vektor v in der k-ten Diagonalen über der Hauptdiagonalen.

- `A(k,:) = []` löscht die k-te Zeile aus A.

- `A(:,k) = []` löscht die k-te Spalte aus A.

Was ist `diag(ones(3,1))`?

Beispiele: $A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}, \quad v = \begin{pmatrix} 7 \\ 8 \end{pmatrix}.$

```
>> A = [1 2 3; 4 5 6]; v = [7; 8];
>> v(2)
ans = 8
```

```
>> A(2,3)
ans = 6
>> A(2,2:3)
ans =
  5  6
>> A(1,:)
ans =
  1  2  3
```

Lösung linearer Gleichungssysteme

- `A\b` löst die Gleichung $Ax = b$.

Falls A eine $(n \times n)$-Matrix ist, wird Gaußelimination verwendet, sonst wird via QR-Zerlegung eine Lösung im Sinne der kleinsten Fehlerquadrate berechnet. Ist A schlecht konditioniert oder singulär wird eine Warnung ausgegeben.

Zerlegungen, Eigenwerte, ...

- `[L,U,P] = lu(A)` berechnet eine LU-Zerlegung von A, wobei P eine Permutationsmatrix ist mit: $LU = PA$.

- `[Q,R] = qr(A)` berechnet die QR-Zerlegung von A, es gilt also $QR = A$.

- `R = chol(A)` berechnet die Choleskyzerlegung von A.

- `S = svd(A)` berechnet die Singulärwerte von A.

- `H = hess(A)` bringt A in Hessenbergform.

- `E = eig(A)` berechnet alle Eigenwerte von A.

- `[V,D] = eig(A)` berechnet eine Diagonalmatrix D, die die Eigenwerte enthält und eine Matrix V mit den dazugehörigen Eigenvektoren. Es gilt also $AV = VD$.

- `norm(X,p)` berechnet die p-Norm des Vektors X. Falls X eine Matrix ist, kann p die Werte `1`, `2` oder `inf` haben. Wird p nicht angegeben, wird $p = 2$ standardmäßig verwendet.

- `cond(A)` berechnet die Konditionszahl von A in 2-Norm.

Bemerkung: Viele dieser Befehle unterstützen noch weitere Optionen. Diese können Sie mit `help funcname` erfahren.

Logische Operatoren

<	kleiner	<=	kleiner oder gleich
>	größer	>=	größer oder gleich
==	gleich	<>	ungleich
&	und	|	oder
~	nicht		

- `[ind1,ind2,v] = find(A)` findet die Indizes aller Elemente $\neq 0$ in A. Es gilt: $A_{\text{ind1}(l),\text{ind2}(l)} = v_l \neq 0$.

- `any(A)` ist 1, falls A Elemente $\neq 0$ enthält. Bei Matrizen operiert `any` auf den Spalten.

Funktionen und Scripts

- `whos` zeigt alle definierten Variablen und Funktionen an.

- `clear name` löscht „name“ aus dem Speicher; falls „name“ nicht angegeben wird,

werden *alle* Variablen und Funktionen gelöscht.

- `type name` gibt die Funktion „name“ am Bildschirm aus.

Matlab^{TM} läßt sich durch eigene Programme und Funktionen leicht erweitern. Sowohl Scripts als auch Funktionen sind einfache *Textdateien*, die den Suffix `.m` haben.
Funktionen können mit Argumenten aufgerufen werden und alle innerhalb der Funktion verwendeten Variablen sind *lokal*, d.h. sie beeinflussen die vorher definierten Variablen nicht.
Scripts hingegen verhalten sich so, als ob deren Inhalt Befehl für Befehl am Prompt eingegeben würde.

Eine Funktion `dolittle`, die in der Datei `dolittle.m` gespeichert ist, könnte etwa folgendermaßen aussehen:

```
function [out1,out2] = dolittle(x)
% Das ist ein Kommentar zu dolittle
out1 = x^2;
out2 = out1*x;
% dolittle endet hier
```

Aufruf der Funktion:

```
>> [x1,x2]=dolittle(2)
x1 = 4
x2 = 8
>> whos
  Name        Size              Bytes  Class

  x1          1x1                   8  double array
  x2          1x1                   8  double array

Grand total is 2 elements using 16 bytes
```

Offensichtlich waren die beiden Variablen `out1, out2` in `dolittle` lokal.
Wichtig: Eine vorher definierte Variable `out1` oder `out2` wird durch den Aufruf der Funktion *nicht* beeinflußt.

Ein Script `doless`, der in der Datei `doless.m` gespeichert ist, könnte etwa folgendermaßen aussehen:

```
  eins = 1;
  zwei = 2;
  drei = eins + zwei;
```

Aufruf dieses Scripts:

```
>> doless
>> whos
  Name        Size              Bytes  Class

  drei        1x1                   8  double array
  eins        1x1                   8  double array
  zwei        1x1                   8  double array

Grand total is 3 elements using 24 bytes
```

Es ist aber auch möglich, Variablen „global“ zu definieren:

- `global name` deklariert „name“ als globale Variable.

Eine Funktion `foo` „sieht“ die globale Variable durch folgenden Syntax:

```
function out = foo(arg1,arg2)
global N % verwende N als globale Variable
<Berechnung>
end
```

Wird N in der Funktion verändert, ändert das auch den ursprünglichen Wert von N im „workplace“.

Schleifen und ähnliches

Syntax	Beispiel

`for name = expr`
`  ...`
`end`
	`for n = 1:10`
	`  [x(n),y(n)]=dolittle(n);`
	`end`

`if condition`
`  ...`
`[else`
`  ...]`
`end`
	`if x==0`
	`  error('x ist 0!');`
	`else`
	`  y = 1/x;`
	`end`

`while condition`
`  ...`
`end`
	`while t<T`
	`  t = t+h;`
	`end`

`switch expression`
`  case label`
`  ...`
`  case label`
`  ...`
`  [otherwise`
`  ...]`
`end`
	`switch pnorm`
	`  case 1;`
	`  sum(abs(v))`
	`  case inf;`
	`  max(abs(v))`
	`  otherwise`
	`  sqrt(v'*v)`
	`end`

Funktionen von Funktionen

- `eval(string)` wertet „string“ als **Matlab**^{TM}-Code aus.

- `feval(funcname,arg1,...)` ist äquivalent zum Aufruf der Funktion, die im String „funcname“ steht, mit den Argumenten „arg1, ...“.

Berechne mit der Mittelpunktsregel:

$$\int_a^b f(x)dx \approx \frac{b-a}{N} \sum_{j=0}^{N-1} f\left(a + (j+\frac{1}{2})\frac{b-a}{N}\right)$$

Die Funktionen `gauss.m`, `mpr1.m` und `mpr2.m` seien folgendermaßen definiert:

```
function y = gauss(x)
  y = exp(-x.^2/2);
end
```

```
function S = mpr1(fun,a,b,N)
  h = (b-a)/N;
  S = h*sum(feval(fun,[a+h/2:h:b]));
end
```

```
function S = mpr2(fun,a,b,N)
  h = (b-a)/N; S = 0;
  for k = 0:(N-1),
    S = S + feval(fun,a+h*(k+1/2));
  end
  S = h*S;
end
```

```
>> t = cputime;
>> Int1=mpr1('gauss',0,5,500); t1=cputime-t;
>> t = cputime;
>> Int2=mpr2('gauss',0,5,500); t2=cputime-t;
>> Int1-Int2, t2/t1
ans = 0
ans = 45.250
```

Fazit: Schleifen und Funktionsaufrufe, insbesondere über den Umweg `feval`, sind sehr teuer, deswegen möglichst alle Operationen „vektorisieren“!

Ein- und Ausgabe

- `save datei var1 [var2 ...]` speichert die Variablen „var1“ usw. in der Datei „datei“ ab.

- `load datei` liest die Datei „datei“.

- `fprintf(string[,var1,...])` lehnt sich stark an den Syntax in C an, siehe `man fprintf` unter Unix.

- `format [long|short]` vergrößert oder verkleinert die Anzahl der ausgegebenen Kommastellen. Nur `format` stellt das Standardverhalten wieder her.

Beispiel:

```
>> for k = .1:.2:.5,
>> fprintf('1/%g = %10.2e\n',k,1/k); end
1/0.1 =   1.00e+01
1/0.3 =   3.33e+00
1/0.5 =   2.00e+00
```

Grafik

2D-Grafiken:

- `plot(x,y[,fmt])` zeichnet eine Linie durch die Punkte (x_j, y_j). Mit dem String „fmt“ können Linienart und Farbe gewählt werden (siehe `help plot`).

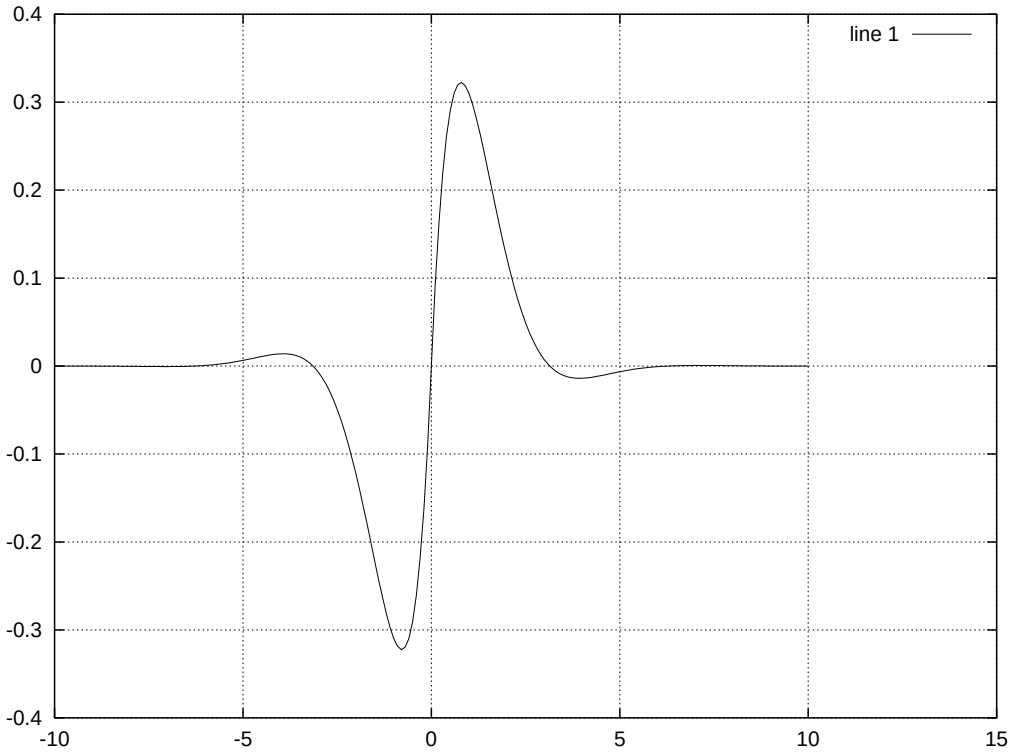
- `semilogx(x,y[,fmt])` Syntax wie `plot`, die x-Achse wird logarithmisch gezeichnet.

- `semilogy(x,y[,fmt])` Syntax wie `plot`, die y-Achse wird logarithmisch gezeichnet.

- `loglog(x,y[,fmt])` Syntax wie `plot`, beide Achsen werden logarithmisch gezeichnet.

Beispiel:

```
>> x = -10:.1:10;
>> y = sin(x).*exp(-abs(x));
>> plot(x,y); grid; replot
```



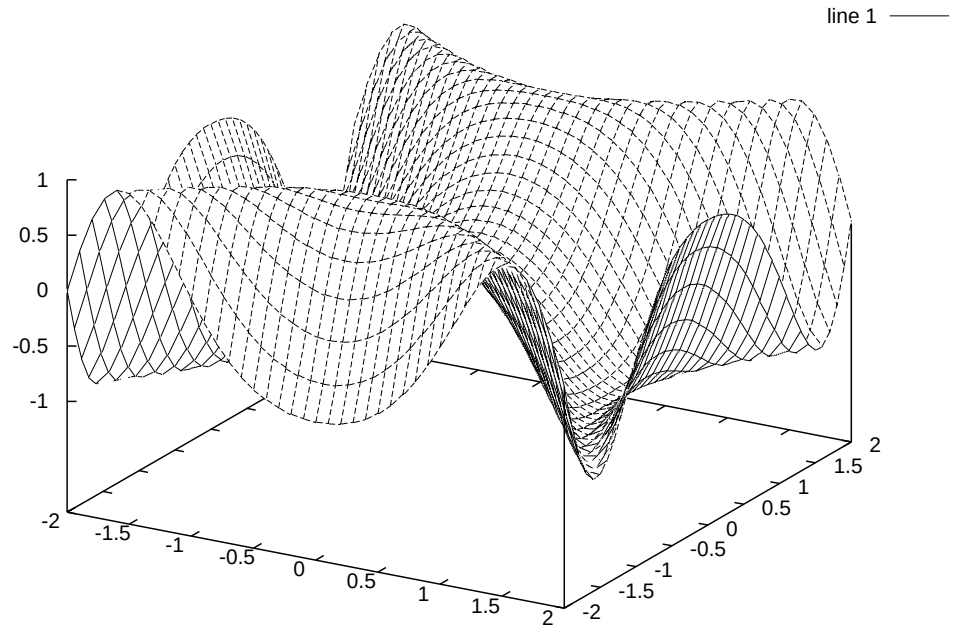
3D-Grafiken:

- `[xx,yy] = meshgrid(x,y)` erzeugt aus den Vektoren x und y Gitterdaten.

- `mesh(x,y,z)` zeichnet eine Fläche in 3D.

Beispiel:

```
>> x = -2:0.1:2;
>> [xx,yy] = meshgrid(x,x);
>> z = sin(xx.^2-yy.^2);
>> mesh(x,x,z);
```



Befehle für 2D- und 3D-Grafiken:

- `title(string)` schreibt „string“ als Titelzeile in die Grafik.
- `xlabel(string)` beschriftet die x-Achse mit „string“.
- `ylabel(string)` beschriftet die y-Achse mit „string“.
- `zlabel(string)` beschriftet die z-Achse mit „string“.

- `axis(v)` legt den Ausschnitt fest. v ist ein Vektor der Form v = (xmin, xmax, ymin, ymax[, zmin zmax]).

- `hold [on|off]` legt fest, ob die nächste Grafikausgabe die alte Grafik löschen soll.

- `clg` löscht das Grafikfenster.

- `figure(n)` erzeugt Grafikfenster mit Nummer 'n'.

- `subplot(m,n,i)` unterteilt das Grafikfenster in m$\times$n Unterfenster

Übungen

Erzeugen Sie eine Matrix A und einen Vektor b mit

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \text{ und } b = \begin{pmatrix} 36 \\ 88 \end{pmatrix}.$$

Lösen Sie nun das Gleichungssystem $Ax = b$ durch Eingabe von 4 Zeichen. Berechnen Sie danach die LU- und QR-Zerlegung sowie die Eigenwerte mit Eigenvektoren von A. Rechnen Sie die Choleskyzerlegung von $A^T A$ aus, und überzeugen Sie sich, daß $\text{cond}(A^T A) = \text{cond}(A)^2$ ist.

Lösungsvorschlag:

```
A = reshape(1:4,2,2).'; b = [36; 88];
A\b
[L,U,P] = lu(A)
[Q,R] = qr(A)
[V,D] = eig(A)
A2 = A.'*A;
R = chol(A2)
cond(A)^2 - cond(A2)
```

Berechnen Sie ein Matrix-Vektor-Produkt einer (100×100)-Zufallsmatrix mit einem Zufallsvektor, indem Sie einerseits die eingebaute Funktion `*`, andererseits FOR-Schleifen, verwenden.

Lösungsvorschlag:

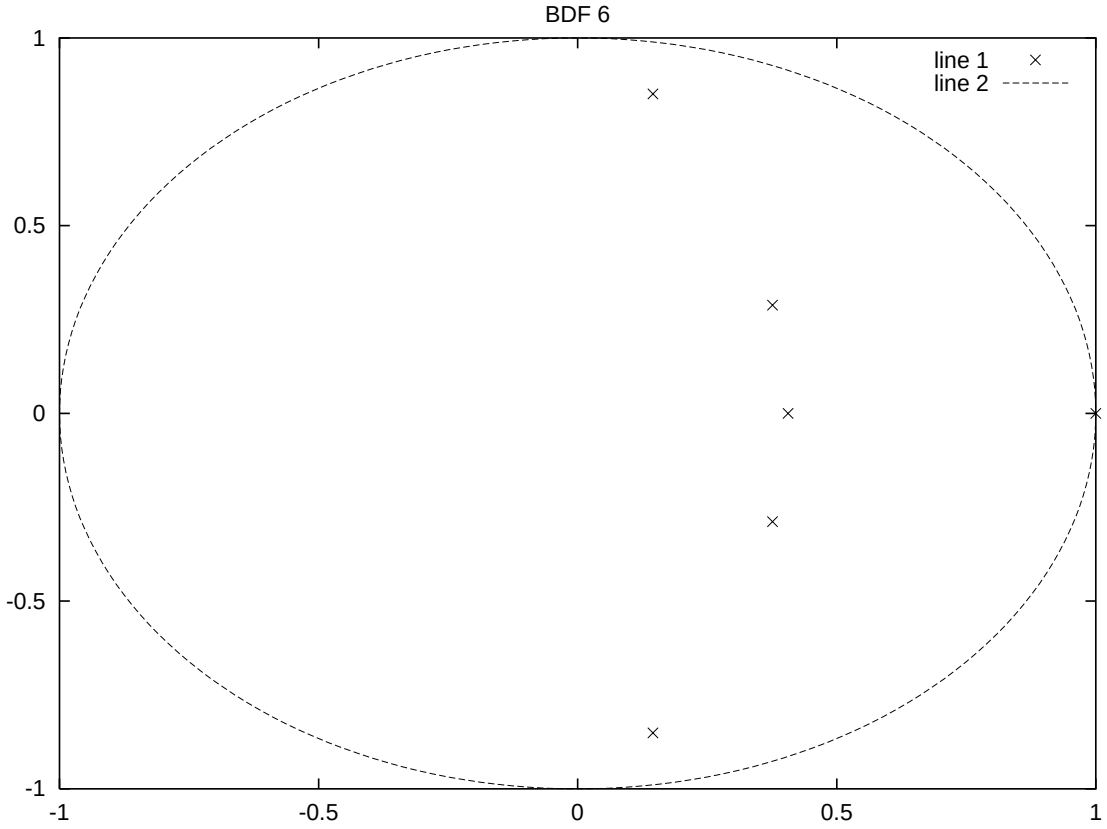
```
A = rand(100); b = rand(100,1);
t = cputime;
v = A*b; t1 = cputime-t;
w = zeros(100,1);
t = cputime;
for n = 1:100,
  for m = 1:100
    w(n) = w(n)+A(n,m)*b(m);
  end
end
t2 = cputime-t;
norm(v-w), t2/t1
ans = 0
ans = 577.00
```

Berechnen Sie alle Nullstellen des Polynoms

$$\frac{147}{60}\zeta^6 - 6\zeta^5 + \frac{15}{2}\zeta^4 - \frac{20}{3}\zeta^3 + \frac{15}{4}\zeta^2 - \frac{6}{5}\zeta + \frac{1}{6}$$

Hinweis: Verwenden Sie den Befehl `compan`.

Zeichnen Sie diese Nullstellen in der komplexen Ebene als Punkte ein, und fügen Sie einen Einheitskreis hinzu.
Hinweis: `hold`, `real`, `imag`.



Lösungsvorschlag:

```
bdf6 = [147/60 -6 15/2 -20/3 15/4 -6/5 1/6];
R = eig(compan(bdf6));
plot(R,'+'); hold on
plot(exp(pi*i*[0:.01:2]));
if any(find(abs(R)>1))
  fprintf('BDF6 ist instabil\n');
else
  fprintf('BDF6 ist stabil\n');
end
```

Zeichnen Sie den Graphen der Funktion

$$f(x,y) = \exp(-x^2 - y^2) \quad .$$

Lösungsvorschlag:

```
x = -3:0.1:3;
[xx,yy] = meshgrid(x,x);
z = exp(-xx.^2-yy.^2);
mesh(x,x,z);
title('exp(-x^2-y^2)');
replot;
```

Berechnen sie für die $(n \times n)$-Hilbertmatrix H $(n = 1, \ldots, 15)$ die Lösung des linearen Gleichungssystems $Hx = b$, $b =$ `ones(n,1)`. Berechnen Sie den Fehler und die Kondition der Matrix und zeichnen Sie die beiden in einem Fenster halblogarithmisch auf.
Hinweis: `hilb, invhilb`.

Lösungsvorschlag:

```
err = zeros(15,1); co = zeros(15,1);
for k = 1:15
  H = hilb(k);
  b = ones(k,1);
  err(k) = norm(H\b-invhilb(k)*b);
  co(k) = cond(H);
end
semilogy(1:15,err,'r',1:15,co,'x');
```

Berechnen Sie zu beliebig vorgegebenen Punkten (x_j, y_j), die durch zwei Vektoren x und y gegeben sind, die Ausgleichsgerade (im Sinne der kleinsten Fehlerquadrate). Zeichnen Sie die Punkte und die Gerade.

Lösungsvorschlag:

```
function coeff = ausgl(x,y)
  n = length(x);
  A = [x ones(n,1)];
  coeff = A\y;
  plot(x,y,'x');
  hold on
  interv = [min(x) max(x)];
  plot(interv,coeff(1)*interv+coeff(2));
end
```

Schreiben Sie ein Programm, das beliebige Funktionen f in einer Variablen auf einem Intervall $[a, b]$ numerisch integriert. Verwenden sie die Trapezregel ($h = (b - a)/N$):

$$\int_a^b f(x)dx \approx h \left(\frac{f(a) + f(b)}{2} + \sum_{j=1}^{N-1} f(a + jh) \right)$$

Verifizieren Sie in einem doppellogarithmischen Bild anhand einer beliebigen Funktion f, daß die Trapezregel Ordnung 2 hat.

Lösungsvorschlag:

```
function S = trapez(fun,a,b,N)
  h = (b-a)/N;
  % fy = feval(fun,[a:h:b]); besser:
  fy = feval(fun,linspace(a,b,N+1));
  fy(1) = fy(1)/2;
  fy(N+1) = fy(N+1)/2;
  S = h*sum(fy);
end

function y = f(x)
  y = exp(x);
end

for k=1:15;
  err(k) = abs(exp(1)-1-trapez('f',0,1,2^k));
end
loglog(1./2.^[1:15],err);
hold on;
loglog(1./2.^[1:15],err,'x');
title('Trapezregel, f(x) = exp(x)');
xlabel('Schrittweite');
ylabel('Fehler');
replot;
```