

Einführung in Matlab - Einheit 5

Mehrdimensionale Arrays, Funktionen, Dünnbesetzte Matrizen,
Numerische Mathematik

Jochen Schulz

Georg-August Universität Göttingen



Mehrdimensionale Arrays

- mehrdimensionale Arrays (Dim. > 2).

```
A(:, :, 1) = ones(3);  
A(:, :, 2) = 2*ones(3);  
whos
```

Name	Size	Bytes	Class
A	3x3x2	144	double

- `cat(<dim>,<A1>,<A2>,...)` fügt die Arrays `A1`, `A2`,... entlang der Dimension `dim` zusammen.

```
A = cat(3,ones(3), 2*ones(3))
```

- Befehle wie `zeros`, `ones` oder `repmat` funktionieren auch im multidimensionalen Kontext.

Umsortieren von Arrays

```
reshape(X,n1,...,ns)
```

Der Befehl liest X spaltenweise aus, und schreibt die Elemente spaltenweise in ein (n_1, \dots, n_s) -Array.

- X muss $n_1 \cdots n_s$ Elemente enthalten.
- Der Befehl ist sehr nützlich.

Beispiele:

```
reshape(hilb(4), 8,2)  
reshape(hilb(4), 4,2,2)
```

Zugriff auf mehrdim. Arrays

Intern werden Arrays als Spalten abgespeichert. Zugriff durch linearen Index möglich.

```
B = reshape(1:12,2,3,2)
```

```
B(:, :, 1) =
```

1	3	5
2	4	6

```
B(:, :, 2) =
```

7	9	11
8	10	12

```
B(7:9)
```

```
ans =
```

7	8	9
---	---	---

Nützliche Befehle

- Anzahl der Dimensionen von X :

```
ndims(X)
```

- Größe von X :

```
size(X)
```

- Umwandlung von linearer Indizierung in Array-Indizierung:

```
ind2sub
```

- Umwandlung von Array-Indizierung in lineare Indizierung:

```
sub2ind
```

```
A = reshape(1:12,2,3,2);  
A(ind2sub(size(A),5))
```

```
ans =  
5
```

- Man kann auch mit mehrdimensionalen Arrays rechnen.

- Funktions-Typen
 - m.-File
 - inline
 - anonyme
 - string
- Funktionen werden in einem eigenen *Workspace* verwaltet.
- Beim ersten Aufruf speichert MATLAB die Funktion im Workspace bis MATLAB verlassen wird oder die Funktion `fun` mit `clear fun` gelöscht wird.
- Namen: Buchstaben mit 1-63 Zeichen (Ohne -, +, *, / !).

Function-Handles

Ein *Function Handle* ist ein MATLAB Datentyp, das alle Informationen enthält, die zur Auswertung einer Funktion nötig sind.

- Definition, z.B.

```
Sinus = @sin
```

- Anwendung bei der Übergabe von Funktionen:

```
quad(Sinus,0,1)
```

- m-File Funktionen haben ihren Namen als Handle (`@func_name` für `func_name.m`)

- Eingabeparameter als Cell-Array

```
varargin
```

- Die Anzahl der Inputvariablen

```
nargin
```

- Cell-Array der Ausgabewerte

```
varargout
```

- Die Anzahl der Outputvariablen

```
nargout
```

Beispiel: varargin

```
function result = integral(varargin)
% berechnet approximativ ein Integral ueber (a,b)
% durch die Mittelpunkregel mit Hilfe von N Punkten
% Eingabe: 0 Parameter:      (N=20, a=0, b=1)
%           1 Parameter: N   (a=0,b=1)
%           3 Parameter: N,a,b
% Jochen Schulz 16.08.2009
N = 20; a = 0; b = 1; % Default-Einstellung
anzahl_parameter = nargin; % Anz. Input-argumente
if anzahl_parameter == 1
    N = varargin{1};
end
if anzahl_parameter == 3
    N = varargin{1}; a = varargin{2};
    b = varargin{3};
end
if anzahl_parameter ~= [0 1 3]
    error('Falsche Anzahl an Input-Argumenten');
end
```

Beispiel: varargin

```
x = (a+(b-a)/(2*N)):(b-a)/N:(b-(b-a)/(2*N));
y = x.^3;
% Berechnung des Integrals
result = (b-a)*sum(y)*(1/N);

close all; % Plot
x1 = linspace(a,b,N+1);
for i = 1:N
    fill([x1(i) x1(i) x1(i+1) x1(i+1)], [0 y(i) y(i)
        0], 'r');
    hold on;
end
plot(a:(b-a)/100:b,(a:(b-a)/100:b).^3,'LineWidth',3);
title(strcat('\int x^3 = ',num2str(result),...
' fuer N =', num2str(N)));
```


Anonyme Funktion

```
@(<x>) <funktion(x)>
```

- Funktion mit Parameter

```
y = 1; f = @(x) sin(x)./(x+y) ;  
f(2)
```

```
ans =  
    0.3031
```

- Gamma-Funktion $\Gamma(s) = \int_0^{\infty} x^{s-1} e^{-x} dx$.

```
k = @(s) quad( @(x) x.^(s-1).*exp(-x), 0.1, 500) ;  
k(4), k(5)
```

```
ans =  
    6.0000  
ans =  
   24.0000
```

String-Funktionen

```
<fun> = '<funktionen-string>'
```

- Eingabe als String:

```
a = 'exp(z)-1+z'
```

- Plotten der zugehörigen Funktion

```
ezplot(a, [-1 1])
```

Bemerkung:

Funktionen gegeben als Strings sind im allgemeinen zu vermeiden! Besser andere Konstrukte (wie Inline-Funktionen) benutzen!

Inline-Funktionen

```
<fun> = inline('<funktionen-string>')
```

Beispiele:

```
a = 'exp(z) - 1 + z';  
f = inline(a)
```

```
f =  
    Inline function:  
    f(z) = exp(z)-1+z
```

```
g = inline('x+y^2','x','y')
```

```
g =  
    Inline function:  
    g(x,y) = x+y^2
```

```
f(1),g(1,2),a(2)
```

```
ans =  
    2.7183  
ans =  
    5  
ans =  
    x
```


Befehle für Funktionen

- Auswertung der Funktion `fun` an der Stelle (x_1, \dots, x_n) .

```
f eval(<fun>, <x1>, ..., <xn>)
```

`fun` ist dabei entweder ein Funktionsname oder ein Function-Handle.

- Wandlung eines Strings `g` in eine Inline-Funktion (vgl. `inline`).

```
f = fcnchk(<g>)
```

Ist `g` ein Function-Handle oder eine Inline-Funktion so ist $f = g$.

- Strings oder Inline-Funktionen `f` *vektorisieren*

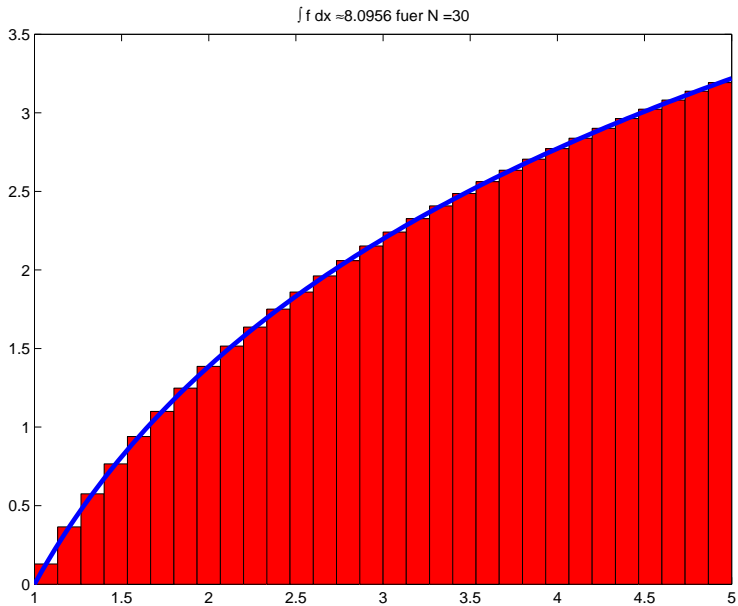
```
vectorize(<f>)
```

d.h. `'*'` \Rightarrow `'.*'`, `'^'` \Rightarrow `'.^'`, usw.

Beispiel: integral2.m (Auszug)

```
function result = integral2(varargin)
% Eingabe: 1 Parameter: f          (N=20, a=0, b=1)
%          2 Parameter: f,N      (a=0,b=1)
%          4 Parameter: f,N,a,b
N = 20; a = 0; b = 1; % Default-Einstellung
anzahl_parameter = nargin; % Anz. Input-argumente
if anzahl_parameter == 2
    N = varargin{2};
end;
if anzahl_parameter == 4
    N = varargin{2}; a = varargin{3}; b = varargin{4};
end;
if anzahl_parameter ~= [1 2 4]
    error('Falsche Anzahl an Input-Argumenten');
end;
% eventuelle Umwandlung von Strings
f = fcnchk(varargin{1}, 'vectorized');
x = (a+(b-a)/(2*N)):(b-a)/N:(b-(b-a)/(2*N));
y = feval(f,x);
```

`integral2('log(x.^2)',30,1,5)`



Beispiel: Sobolevsche Mittelungsfunktion

$$f(x) := \begin{cases} \exp(-\frac{1}{1-\|x\|^2}), & \|x\| < 1 \\ 0, & \|x\| \geq 1 \end{cases}$$

mit $\|x\|^2 := \sum_{i=1}^N x_i^2$, $x = (x_1, \dots, x_N) \in \mathbb{R}^N$.

2 Versionen:

- eindimensionale Version
- N-dimensionale Version

1d-Fall

```
function result = f_1d(x)
% Sobolevsche Mittelungsfunktion (1d)
%  $f(x)=\exp(-1/(1-|x|^2))$ ,  $|x|<1$ , und  $f(x)=0$  sonst
%
% Eingabe: Vektor x
% Ausgabe: Vektor f(x)
%
% Gerd Rapin      7.12.2003

% Berechnen des Funktionswerts
result = zeros(1,length(x));
for k = 1:length(x)
    if abs(x(k))<1
        result(k) = exp(-1/(1-x(k)^2));
    else
        result(k) = 0;
    end
end
```

n-dimensionaler-Fall

```
function result = f(varargin)
% f.m      Sobolevsche Mittelungsfunktion
%          Eingabe: Matrizen x1,x2,x3,..
%          Ausgabe: Matrix result=f(x1,x2,...)
betrag = varargin{1}.^2;
for i = 2:nargin
    betrag = betrag+varargin{i}.^2;
end
dimension = size(varargin{1});
result = zeros(dimension(1),dimension(2));
for j = 1:dimension(1)
    for k = 1:dimension(2)
        if betrag(j,k) < 1
            result(j,k) = exp(-1/(1-betrags(j,k)));
        else
            result(j,k) = 0;
        end;
    end;
end;
end;
```


Programm zum Plotten

```
%      plot_f.m

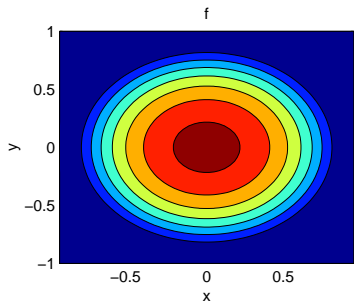
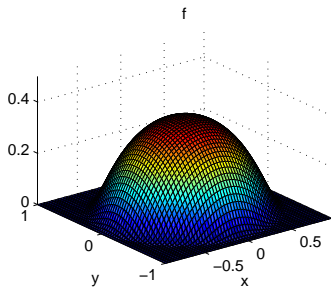
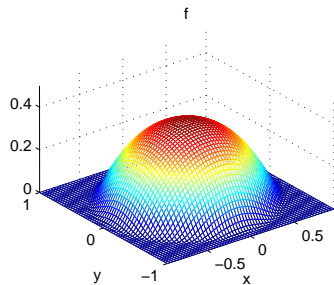
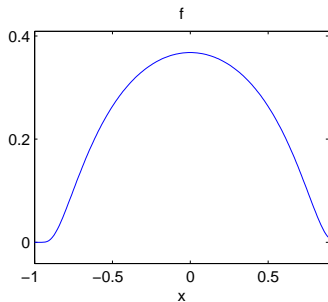
% Eindimensionaler Plot
subplot(2,2,1),
ezplot(@f);

% Zweidimensionaler Plot
subplot(2,2,2),
ezmesh(@f);

% Zweidimensionaler Plot
subplot(2,2,3),
ezsurf(@f);

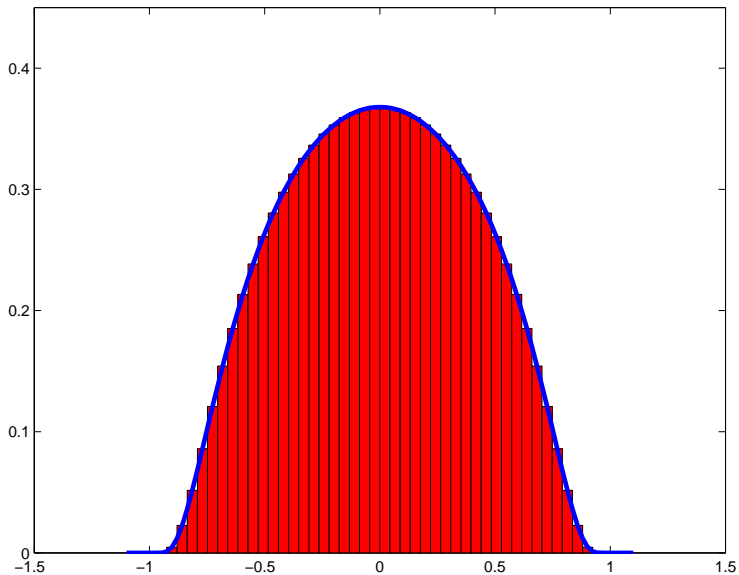
% Zweidimensionaler Plot
subplot(2,2,4),
ezcontourf(@f);
```

Plots der Funktion



`integral2(@f,50,-1.1,1.1)`

$\int f \, dx \approx 0.44399$ fuer $N = 50$



- Bei *Dünnbesetzten Matrizen* (*sparse matrices*) sind fast alle Einträge 0.
- In vielen Anwendungen, z.B. bei der Diskretisierung von Differentialgleichungen oder in der Graphentheorie, treten sehr grosse, dünnbesetzte Matrizen auf.
- In MATLAB steht dafür ein eigener Datentyp zur Verfügung, der zu jedem Nichtnullelement der Matrix, die zugehörige Zeile und Spalte speichert.

Beispiel

```
A = 2*diag(ones(10,1),0) ...  
    - diag(ones(9,1),-1) ...  
    - diag(ones(9,1),1);  
B = sparse(A)
```

```
B =      (1,1)      2  
        (2,1)     -1  
        (1,2)     -1  
        (2,2)      2
```

```
C = 2*diag(ones(100,1),0) ...  
    - diag(ones(99,1),-1) ...  
    - diag(ones(99,1),1);  
D = sparse(C); whos
```

Name	Size	Bytes	Class
A	10x10	800	double array
B	10x10	380	sparse array
C	100x100	80000	double array
D	100x100	3980	sparse array

Einige Befehle

- Erzeugung einer dünnbesetzten Matrix der Grösse $n \times m$. Alle Einträge sind 0.

```
sparse(n,m)
```

- Konvertierung der dichtbesetzten Matrix A in eine dünnbesetzte Matrix.

```
sparse(A)
```

- Die Struktur der Matrix A visualisieren.

```
spy(A)
```

- Die meisten Standardoperationen funktionieren auch mit dünnbesetzten Matrizen.

Dichte und dünnbesetzte Matrizen

- Konvertierung der dünnbesetzten Matrix A in eine dichtbesetzte Matrix B .

```
B = full(A)
```

- Bei binären Operationen, z.B. $A + B$ oder $A * B$ ist das Ergebnis bei dünnbesetzten Matrizen A und B wieder eine dünnbesetzte Matrix. Ist eine der Matrizen dichtbesetzt, so ist auch das Ergebnis dichtbesetzt.
- Berechnung der k betragsmäßig grössten Eigenwerte (Default: $k = 6$):

```
eigs(A,k)
```


Dünnbesetzte Matrizen

- Norm- und Konditionsberechnung:

```
normest(<A>) , condest(<A>)
```

- Alle iterativen Verfahren funktionieren auch mit dünnbesetzten Matrizen.
- Indizes aller Zeilen und Spalten erhalten, in denen Nichtnullelemente stehen:

```
[I,J] = find(X)
```

- Eine Übersicht aller Funktionen für dünnbesetzte Matrizen erhält man durch `help sparsfun`.

Die p -Norm eines Vektors $x = (x_1, \dots, x_n)$

$$\|x\|_p := \left(\sum_{i=1}^n |x_i|^p \right)^{1/p}$$

(definiert für $p \geq 1$).

- in MATLAB: `norm(x,p)` (Default: $p = 2$)
- $p = \infty$ entspricht der Maximum-Norm

$$\|x\|_\infty = \max_{i=1,\dots,n} |x_i|.$$

Seien $A \in \mathbb{C}^{n \times m}$ und $p \geq 1$. Die *Matrixnorm* ist definiert durch

$$\|A\|_p = \sup_{x \in \mathbb{C}^m \setminus \{0\}} \frac{\|Ax\|_p}{\|x\|_p}.$$

- In MATLAB: `norm(A,p)` (Default $p = 2$).
- $p = \infty$ kann charakterisiert werden durch

$$\|A\|_\infty = \max_{1 \leq j \leq m} \sum_{i=1}^n |a_{ij}|, \quad \text{Zeilensummennorm.}$$

Kondition einer quadratischen Matrix A :

$$\text{cond}_p(A) := \|A\|_p \|A^{-1}\|_p.$$

- In MATLAB: `cond(A,p)` (Default $p = 2$)
- Es gilt $\text{cond}_p(A) \geq 1$.
- Die Kondition mißt die Empfindlichkeit der Lösung x von $Ax = b$ gegenüber Störungen von A und b .
- Ist $\text{cond}_p(A) \gg 1$, so ist die Matrix beinahe singular. Die Matrix ist *schlecht konditioniert*.

- Vektornormen für $x = (1/100)(1, 2, \dots, 100)$

```
>> x = (1:100)/100; [norm(x,1) norm(x,2) norm(x,inf)]  
ans =      50.5000      5.8168      1.0000
```

- Matrixnorm für die Hilbert-Matrix $H = (\frac{1}{i+j-1})_{ij}$

```
>> H = hilb(10); [norm(H,1) norm(H,2) norm(H,inf)]  
ans =      2.9290      1.7519      2.9290
```

- Kondition der Hilbert-Matrix

```
>> H = hilb(10); [cond(H,1) cond(H,2) cond(H,inf)]  
ans =  
      1.0e+13 *  
      3.5354      1.6025      3.5354
```


Lineare Gleichungssysteme

Seien $A \in \mathbb{C}^{n \times n}$ und $b \in \mathbb{C}^n$. Das lineare Gleichungssystem

$$Ax = b$$

wird in MATLAB gelöst durch $x=A \backslash b$.

```
>> x = ones(5,1); H = hilb(5); b = H*x; y = (H\b) '
y =
    1.0000    1.0000    1.0000    1.0000    1.0000
```

Warnung: Benutze nie $x=\text{inv}(A)*b$, da das Berechnen von A^{-1} sehr aufwendig sein kann.

Was bedeutet $A \backslash b$?

MATLAB berechnet die LU-Zerlegung von A (Gaussverfahren):

- obere Dreiecksmatrix U
- untere Dreiecksmatrix L mit Einsen auf der Diagonalen

so dass $PA = LU$ gilt (P Permutationsmatrix).

Dann wird das LGS durch Rückwärts- und Vorwärtseinsetzen gelöst
($Lz = Pb$, $Ux = z$)

```
>> [L,U,P]=lu(hilb(5)); norm(P*hilb(5)-L*U)
ans = 2.7756e-17
```

Inverse, Determinante

- Berechnung der Inversen

```
>> A=pascal(3)
```

```
A =
```

1	1	1
1	2	3
1	3	6

```
>> X=inv(A)
```

```
X =
```

3	-3	1
-3	5	-2
1	-2	1

- Berechnung der Determinante

```
>> det(A)
```

```
ans = 1
```

Pseudoinverse

(Moore-Penrose) Pseudoinverse

Sei A singulär, Bestimme X so dass

$$AXA = A, XAX = X, (XA)^* = XA, (AX)^* = AX$$

```
>> pinv(ones(3,3))  
ans =  
    0.1111    0.1111    0.1111  
    0.1111    0.1111    0.1111  
    0.1111    0.1111    0.1111
```


Zwei-Punkt-Randwert-Aufgabe

Suche eine Funktion

$$u : [0, 1] \rightarrow \mathbb{R},$$

so dass

$$\begin{aligned} -u''(x) &= e^x, & x \in (0, 1) \\ u(0) &= u(1) = 0 \end{aligned}$$

Problem: Es kann i.A. keine geschlossene Lösungsdarstellung angegeben werden.

Ausweg: Approximation der Lösung.

Finite Differenzen Verfahren

Diskretisierung: $0 = x_0 < \dots < x_n = 1$ mit $x_i = \frac{i}{n}$

Differenzenquotient:

$$u''(x_i) \sim \frac{u(x_{i-1}) - 2u(x_i) + u(x_{i+1}))}{h^2}, \quad h := \frac{1}{n}$$

Einsetzen in $-u''(x) = e^x$ ergibt

$$-u(x_{i-1}) + 2u(x_i) - u(x_{i+1})) = h^2 e^{x_i}, \quad i = 1, \dots, n-1$$

Randbedingungen $\Rightarrow u(x_0) = u(x_n) = 0$.

\Rightarrow Lineares Gleichungssystem für $u(x_1), \dots, u(x_{n-1})$.

Diskretes Problem

Setze $z = (z_1, \dots, z_{n-1})^t = (u(x_1), \dots, u(x_{n-1}))^t$.

Löse das Gleichungssystem $Az = F$ mit

$$A := \begin{pmatrix} 2 & -1 & & & 0 \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ 0 & & & -1 & 2 \end{pmatrix}, \quad F := h^2 \begin{pmatrix} e^{\frac{1}{n}} \\ \vdots \\ e^{\frac{n-1}{n}} \end{pmatrix}.$$

Lösung für $n = 21$

- Zerlegung des Intervalls $[0, 1]$

```
x = 0:(1/21):1
```

- Eliminieren der Randpunkte

```
x_i = x(2:21)
```

- Erzeugen der Matrix A (Übungsaufgabe)

Lösung für $n = 21$

- Berechnen der rechten Seite:

```
F = (1/21)^2*transpose(exp(x_i));
```

- Lösen des linearen Gl.

```
z_i = A\F;
```

- Zufügen der Werte am Rand

```
>> z = [0; z_i; 0];
```

Lösung für $n = 21$

```
plot(x,z,'r*','MarkerSize',8)
```

```
plot(x,z,'r*-', 'LineWidth',3, 'MarkerSize',8)
```


Eigenwert

Sei $A \in \mathbb{C}^{n \times n}$. $\lambda \in \mathbb{C}$ ist Eigenwert von A , falls ein Vektor $x \in \mathbb{C}^n$ ungleich 0 existiert, so dass $Ax = \lambda x$ gilt. x heißt Eigenvektor.

- `x=eig(A)`
berechnet die Eigenwerte von A und schreibt sie in den Vektor x .
- `[V,D]=eig(A)`
 D ist eine Diagonalmatrix mit den Eigenwerten auf der Diagonalen.
Die Spalten von V bilden die zugehörigen Eigenvektoren.

Weitere Zerlegungen

- **QR-Zerlegung:** $[Q,R]=\text{qr}(A)$

$m \times n$ - Matrix A eine Zerlegung $A = QR$ erzeugt, (Q eine unitäre $m \times m$ -Matrix, R eine obere $m \times n$ Dreiecksmatrix).

- **Singulärwertzerlegung:** $[U,S,V]=\text{svd}(A)$

$A = U\Sigma V^*$. ($\Sigma \subset \mathbb{C}^{m \times n}$ eine Diagonalmatrix
 $U \subset \mathbb{C}^{m \times m}$, $V \subset \mathbb{C}^{n \times n}$ unitäre Matrizen).

- **Cholesky-Zerlegung:** $R=\text{chol}(A)$

$A = R^*R$ zu einer hermiteschen, positiv definiten Matrix A (R ist eine obere Dreiecksmatrix mit reellen, positiven Diagonalelementen).

- LGS können auch mit Hilfe iterativer Verfahren gelöst werden, z.B. `gmres`, `pcg`, `bicgstab`.
- $A \in \mathbb{C}^{n \times m}$, $n \neq m$ bei $A \backslash b$:
 - $n > m$ (überbestimmter Fall): Least-Square Lösung, d.h. der Ausdruck `norm(A*x-b)` wird minimiert.
 - $n < m$ (unterbestimmter Fall): Grundlösung.

Poisson Problem

- Poisson Problem beschreibt stationäre Wärmeverteilungen.
- *Poisson Problem*: Suche $u \in C^2(\Omega) \cap C(\overline{\Omega})$ mit

$$\begin{cases} -\Delta u &= f & \text{in } \Omega \\ u &= 0 & \text{auf } \partial\Omega \end{cases}$$

für $\Omega = (0, 1)^2$ und $f \in C(\Omega)$.

- *Laplace-Operator* $\Delta u := \sum_{i=1}^d \frac{\partial^2 u}{\partial x_i^2}$

- Äquidistante Gitterweite $h = \frac{1}{N}$, $N \in \mathbb{N}$
- Menge aller Gitterpunkte

$$Z_h := \left\{ (x, y) \in \overline{\Omega} \mid x = z_1 h, y = z_2 h \text{ mit } z_1, z_2 \in \mathbb{Z} \right\}.$$

- Innere Gitterpunkte: $\omega_h := Z_h \cap \Omega$

- Approximation von $\frac{\partial^2 u}{\partial x^2}(x, y)$

$$\frac{u(x-h, y) - 2u(x, y) + u(x+h, y)}{h^2} = \frac{\partial^2 u}{\partial x^2}(x, y) + \mathcal{O}(h^2)$$

- Approximation von $\frac{\partial^2 u}{\partial y^2}(x, y)$

$$\frac{u(x, y-h) - 2u(x, y) + u(x, y+h)}{h^2} = \frac{\partial^2 u}{\partial y^2}(x, y) + \mathcal{O}(h^2)$$

- Addition ergibt für $\Delta u(x, y)$ die Näherung

$$\frac{1}{h^2} (u(x, y-h) + u(x-h, y) - 4u(x, y) + u(x, y+h) + u(x+h, y))$$

- Definition $u_{i,j} := u(ih, jh)$ ergibt an Gitterpunkten (ih, jh)

$$-u_{i,j-1} - u_{i-1,j} + 4u_{i,j} - u_{i+1,j} - u_{i,j+1} = h^2 f_{ij}$$

mit $i, j \in \{1, \dots, N-1\}$ und $f_{ij} := f(ih, jh)$.

- Randbedingungen ergeben $u_{0,i} = u_{N,i} = u_{i,0} = u_{i,N} = 0$, $i = 0, \dots, N$.

- Lexikografische Sortierung der inneren Unbekannten

$$\begin{array}{cccc}
 (h, (N-1)h) & (2h, (N-1)h) & \dots & ((N-1)h, (N-1)h) \\
 \vdots & \vdots & \vdots & \vdots \\
 (h, 2h) & (2h, 2h) & \dots & ((N-1)h, 2h) \\
 (h, h), & (2h, h) & \dots & ((N-1)h, h)
 \end{array}$$

ergibt Vektor $U_{i+(N-1)(j-1)} = u_{i,j}$.

Lineares Gleichungssystem für $U = (U_i)_{i=1}^{(N-1)^2}$

$$AU = F$$

mit

- $F := (f_i)_{i=1}^{(N-1)^2}$ mit $f_{i+(N-1)(j-1)} = f(ih, jh)$, $i, j \in \{1, \dots, N-1\}$,
-

$$A := \frac{1}{h^2} \text{tridiag}(-I_{N-1}, T, -I_{N-1}) \in \mathbb{R}^{(N-1)^2 \times (N-1)^2},$$

$$T := \text{tridiag}(-1, 4, -1) \in \mathbb{R}^{(N-1) \times (N-1)}.$$

Implementierung

```
function loes = poisson (f,n)
f = fcnchk(f);
A = gallery('poisson',n-1);
% Erzeuge rechte Seite und Mesh
mesh = zeros(2,(n-1)^2);
F = zeros((n-1)^2,1);
for i = 1:(n-1)
    for j = 1:(n-1)
        F(i+(n-1)*(j-1)) = (1/n)^2*f(i/n,j/n);
        loes.mesh(:,i+(n-1)*(j-1)) = [i/n; j/n];
    end
end
% Loese das lineare System
loes.x = A \ F;
```



```
% Ergaenze Randbedingungen
```

```
loes.x = [ loes.x; zeros(4*(n+1),1)];  
loes.mesh = [loes.mesh, [zeros(1,n+1); 0:1/n:1]];  
loes.mesh = [loes.mesh, [ones(1,n+1); 0:1/n:1]];  
loes.mesh = [loes.mesh, [0:1/n:1; ones(1,n+1)]];  
loes.mesh = [loes.mesh, [0:1/n:1; zeros(1,n+1)]];
```

```
% Plotten
```

```
plot3(loes.mesh(1,:),loes.mesh(2,:),loes.x,'*');  
figure;  
[X,Y] = meshgrid(0:1/n:1,0:1/n:1);  
Fi = TriScatteredInterp(loes.mesh(1,:)', loes.mesh(2,:)',  
    loes.x,'linear');  
Z = Fi(X,Y);  
surf(X,Y,Z);
```


Gewöhnliche Differentialgleichungen

Sei $I \subset \mathbb{R}$ ein Intervall. Bei einer gewöhnlichen Dgl. sucht man eine Funktion $y: I \rightarrow \mathbb{R}^n$, so dass

$$\frac{d}{dt}y(t) = f(t, y(t)), t \in I \quad y(t_0) = y_0,$$

wobei $y_0 \in \mathbb{R}^n$ ein vorgegebener Anfangswert an $t_0 \in I$ und $f: I \times \mathbb{R}^n \rightarrow \mathbb{R}^n$ die rechte Seite ist. Außerdem sei $\frac{d}{dt}y(t) := (\frac{\partial y_1(t)}{\partial t}, \dots, \frac{\partial y_n(t)}{\partial t})^t$.

Beispiele:

$$\frac{d}{dt}y(t) = y(t), \quad y(t_0) = y_0, \quad \text{Lösung: } y(t) = y_0 e^{t-t_0}$$

$$\frac{d}{dt}y(t) = e^y \sin(t), \quad \text{Lösung: } y(t) = -\log(\cos(x) + C), \quad C + \cos(x) > 0$$

Skalares Beispiel

Löse für $0 \leq t \leq 3$ mit `ode45` die Dgl.

$$\frac{d}{dt}y(t) = -y(t) - 5e^{-t}\sin 5t, \quad y(0) = 1.$$

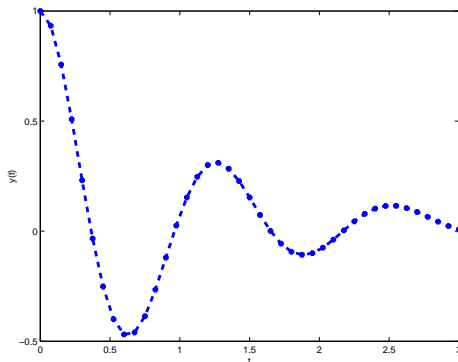
- Die rechte Seite als eigene Funktion:

```
function z = rechte_seite1(t,y)
% rechte_seite1    ODE Beispiel
%                z=rechte_seite1(t,y)
z = -y-5*exp(-t)*sin(5*t);
```

Skalares Beispiel

- Ausrechnen und Plotten

```
tspan = [0,3]; aw = 1;  
[t,y] = ode45(@rechte_seite1,tspan,aw);  
plot(t,y,'*--','Linewidth',3)  
xlabel('t'), ylabel('y(t)')
```



```
[<t>,<y>] = ode45(<@fun>, <tspan>, <aw>, <options>)
```

- **@fun** steht für die rechte Seite der Dgl. (m-File).
- $aw \in \mathbb{R}^n$ ist der Anfangswert.
- **tspan** gibt das Zeitintervall an, auf dem die Dgl. berechnet werden soll. Normalerweise ist es von der Form **tspan**=[t_0, t_1]. Dann wird die Dgl. auf dem Intervall $[t_0, t_1]$ berechnet (Anfangswert: $y(t_0) = aw$).
- Rückgabewerte: Vektoren t und Matrizen y . Dabei ist $y(:, i)$ die Lösung an der Stelle $t(i)$. Die Punkte t_i werden automatisch bestimmt.
- Durch die optionale Angabe von **options** kann der Löser gezielt eingestellt werden.
- Spezifiziert man mehr als zwei Zeitpunkte in **tspan**, so gibt MATLAB die Lösung genau an diesen Zeitschritten zurück.

Die genauen Parameter der ODE-Löser können durch

```
options = odeset('Eigenschaft 1','Spez. 1',...  
                'Eigenschaft 2','Spez. 2',...)
```

gesteuert werden. Die wichtigsten Parameter sind **AbsTol** (Default 10^{-6}) und **RelTol** (Default: 10^{-3}).

Beispiel:

```
options = odeset('AbsTol',1e-7,'RelTol',1e-4)
```

Löser	Steifigkeit	Algorithmus	Ordnungen
ode45	nicht steif	Expliziter Runge-Kutta Löser	4, 5
ode23	nicht steif	Expliziter Runge-Kutta Löser	2, 3
ode113	nicht steif	Explizites Mehrschrittverfahren	1 - 13
ode15s	steif	Implizites Mehrschrittverfahren	1 - 5
ode23s	steif	Modifiziertes Rosenbrockverfahren	2, 3
ode23t	mittel steif	implizite Trapez Regel	2, 3
ode23tb	steif	Implizites Runge-Kutta Verf.	2, 3

Die Lorenz-Gleichungen

- Chaostheorie / Schmetterlingseffekt.

$$\frac{d}{dt}y_1(t) = 10(y_2(t) - y_1(t))$$

$$\frac{d}{dt}y_2(t) = 28y_1(t) - y_2(t) - y_1(t)y_3(t)$$

$$\frac{d}{dt}y_3(t) = y_1(t)y_2(t) - 8y_3(t)/3$$

rechte Seite:

```
function z = lorenz_rechte_seite(t,y)
z = [10*(y(2)-y(1)); ...
     28*y(1)-y(2)-y(1)*y(3); ...
     y(1)*y(2)-8*y(3)/3];
```

Die Lorenz-Gleichungen

```
%-----  
%   lorenz_gl.m  
%   Eine Approximation der Lorenzgleichungen  
%-----  
tspan = [0,30]; aw = [0;1;0];  
options = odeset ('AbsTol',1e-7,'RelTol',1e-4);  
[t,y] = ode45(@lorenz_rechte_seite,tspan,aw, options);  
  
subplot(2,2,1),plot3(y(:,1),y(:,2),y(:,3)),  
subplot(2,2,2),plot(y(:,1),y(:,2)),xlabel('y_1'),ylabel('y_2');  
subplot(2,2,3),plot(y(:,1),y(:,3)),xlabel('y_1'),ylabel('y_3');  
subplot(2,2,4),plot(y(:,2),y(:,3)),xlabel('y_2'),ylabel('y_3');
```

Die Lorenz-Gleichungen

