Einführung in Matlab und Python - Einheit 5

Mehrdimensionale Arrays, Funktionen, Numerische lineare Algebra, Dünnbesetzte Matrizen

Jochen Schulz

Georg-August Universität Göttingen

Aufbau

Mehrdimensionale Arrays

- mehrdimensionale Arrays (Dim. > 2).

```
A = ones(4,3,2);
whos
```

```
Name    Size    Bytes  Class
A       4x3x2     192  double
```

```
cat(<dim>,<A1>,<A2>,..)
```

fügt die Arrays A1, A2,.. entlang der Dimension dim zusammen.

```
A = cat(3,ones(4,3),ones(4,3))
```

```
A = concatenate((ones((4,3))[...,None],ones((4,3))
    [...,None]), axis=2)
```

- 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 $(n_1, \ldots, n_s)$-Array.

- X muss $n_1 \cdots n_s$ Elemente enthalten.

Beispiele:

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

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

```
B.ravel()[6:9]
```

```
ans =
     7     8     9
```

Nützliche Befehle

- ndims(X)| ndim(X): Anzahl der Dimensionen von X
- size (X)|X.shape: Größe von X
- ind2sub: Umwandlung von linearer Indizierung in Array-Indizierung
- sub2ind: Umwandlung von Array-Indizierung in lineare Indizierung

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

```
ans =
     5
```

- Man kann auch mit mehrdimensionalen Arrays rechnen.

Aufbau

Aufbau

Funktionen

- Funktions-Typen
  - normal
  - anonym/lambda
  - (Matlab) inline
  - (Matlab) string
- (Matlab) Funktionen werden in einem eigenen *Workspace* verwaltet.
- (Matlab) Beim ersten Aufruf speichert MATLAB die Funktion im Workspace bis MATLAB verlassen wird oder die Funktion `fun` mit `clear fun` gelöscht wird.

Function-Handles

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

- Definition, z.B.

```
Sinus = @sin
```

```
Sinus = sin
```

- Anwendung bei der Übergabe von Funktionen:

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

```
from scipy.integrate import quad
quad(Sinus,0,1)
```

Anonyme Funktion

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

```
lambda <x> : <funktion(x)>
```

- Funktion mit Parameter

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

```
y = 1; f = lambda 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)
```

```
k = lambda s: quad( lambda x: x**(s-1)*exp(-x)
    ,0.1,500); k(4),k(5)
```

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

Matlab: Inline-Funktionen

```
<fun> =  inline('<funktions-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
```

Matlab: Befehle für Funktionen

- Auswertung der Funktion fun an der Stelle $(x1, .., xn)$.

```
feval(<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.

Aufbau

Matlab: Ein-/Ausgabe - Argumente

- Eingabeparameter als Cell-Array

```
varargin
```

- Die Anzahl der Inputvariablen

```
nargin
```

- Cell-Array der Ausgabewerte

```
varargout
```

- Die Anzahl der Outputvariablen

```
nargout
```

Matlab: varargin Beispiel

```
function result = integral(varargin)
% berechnet approximativ ein Integral ueber (a,b)
% durch die Mittelpunktregel 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
```

Matlab: varargin Beispiel

```
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)));
```

Python: Funktions-Argumente

```
def <fun> (<arg1>,<arg2>=<defaultvalue>,... ,*args,**
    kwargs)
```

- *args: Tuple der Input- Argumente
- **kwargs: Dictionary der benannten Input-Argumente
- *: Entpackt Tuple in eine Liste von Argumenten
- **: Entpackt Dictionary in eine Liste von benannten Argumenten
- Argumente mit Defaultwert sind optional
- Argumente mit Namen können in beliebiger Reihenfolge angegeben werden

Python: Benannte Argumente Beispiel

```
def integral(N=20,a=(0.,1.)):
    h = (a[1]-a[0])/N
    x = linspace(a[0]+h/2,a[1]-h/2,N)
    y = x**3
    result = (a[1]-a[0])*sum(y)*(1./N)
    x1 = linspace (a[0],a[1],N+1)
    figure()
    for i in range(0,N):
        fill_between([ x1[i],x1[i+1]], [y[i],y[i]] ,
            facecolor='r')
    xplot = arange(a[0],a[1],(a[1]-a[0])/100)
    plot(xplot,xplot**3,linewidth=3)
    title('\int_0^1 x^3 = {} fuer N = {}'.format(result,N
        ),fontsize=20)

integral(N=20,a=(3.,4.))
integral(a=(3.,4.),N=20)
integral(20,(3.,4.))
```

Aufbau

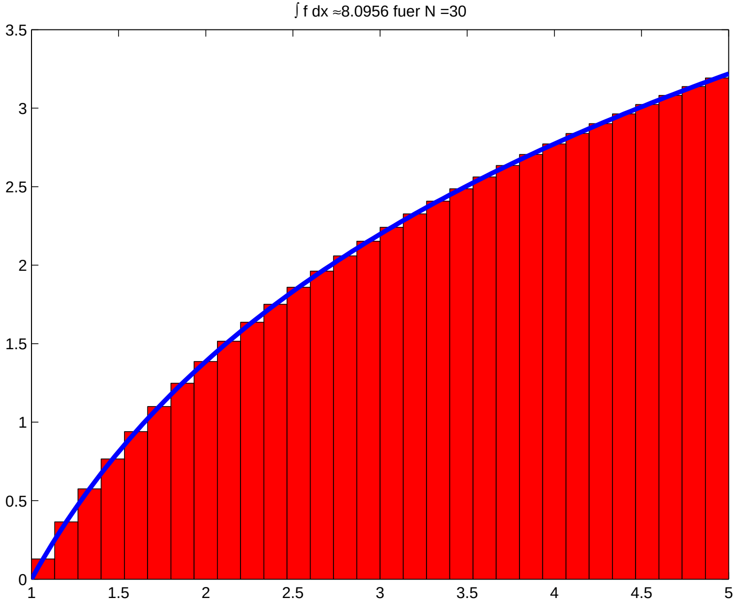
Matlab: integral2.m (Auszug)

```
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);
```

Python: function integral2

```
def integral2(f=lambda x:x**3,N=20,a=(0.,1.),fstr='x^3'):
    h = (a[1]-a[0])/N
    x = linspace(a[0]+h/2,a[1]-h/2,N)
    y = f(x)
    # Berechnung des Integrals
    result = (a[1]-a[0])*sum(y)*(1./N)

    # Plot
    x1 = linspace (a[0],a[1],N+1)
    figure()
    for i in range(0,N):
        fill_between([ x1[i],x1[i+1]], [y[i],y[i]] ,
            facecolor='r')
    xplot = arange(a[0],a[1],(a[1]-a[0])/100)
    plot(xplot,f(xplot),linewidth=3)
    title('\int_{{{}}}^{{{}}} {} = {} fuer N = {}'.format
       (a[0],a[1],fstr,result,N),fontsize=20)
```

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

Matlab: 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: Vektot 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
```

Matlab: 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-betrag(j,k)));
    else
      result(j,k) = 0;
    end;
  end;
end;
```

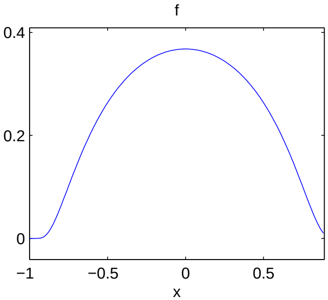
Python: n-dimensionaler Fall

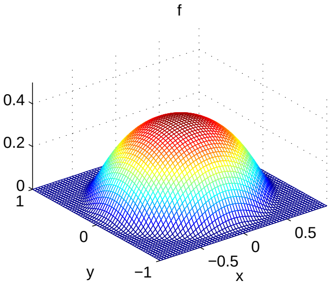
```
def sobnd(*args):
    """ Sobolevsche Mittelungsfunktion
    Eingabe : Matrizen x1 ,x2 ,x3 ,..
    Ausgabe : Matrix result =f(x1 ,x2 ,...)"""

    betrag = 0.
    for i in args:
        betrag += i**2
    dimension = args[0].shape
    result = zeros((dimension[0] ,dimension[1]))
    for j  in range(0,dimension[0]):
        for k in range(0,dimension[1]):
            if betrag[j,k] < 1:
                result[j,k] = exp ( -1/(1 - betrag[j,k]))
            else:
                result[j,k] = 0
    return result
```

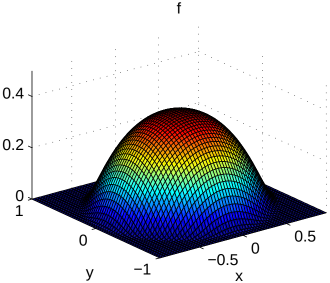
Matlab: Programm zum Plotten

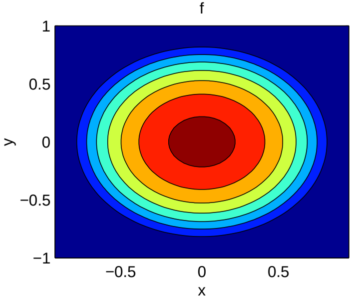
```
% Eindimensionaler Plot
subplot(2,2,1),
ezplot(@f);

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

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

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

Plots der Funktion

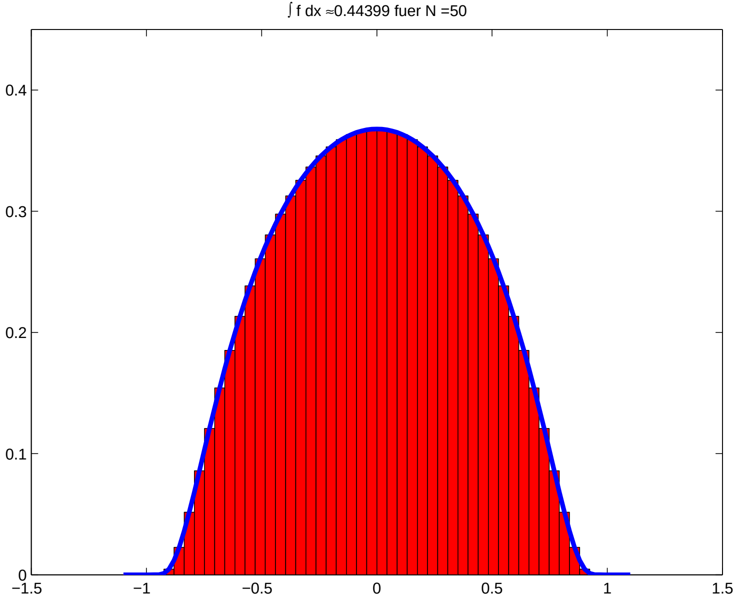








integral2 (@f,50,−1.1,1.1)



Aufbau

Aufbau

Vektornorm

```
norm(x,p)
```

(Default: $p = 2$)
$x = (x_1, \dots, x_n)$

- Die p-Norm (definiert für $p \geq 1$).

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

- $p = \infty$ Maximum-Norm

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

Matrixnorm

```
norm(A,p)
```

(Default $p = 2$).
Seien $A \in \mathbb{C}^{n \times m}$ und $p \geq 1$.

- $p = 1$ Spaltensummennorm:

$$\|A\|_1 = max_j \sum_{i=1}^{n} |a_{ij}|$$

- $p = 2$ Spektralnorm (λ_{max}: größter Eigenwert):

$$\|A\|_2 = \sqrt{\lambda_{max}(A^H \cdot A)}$$

- $p = \infty$ Zeilensummennorm:

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

Kondition

Kondition einer quadratischen Matrix A:

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

```
cond(A,p)
```

(Default $p = 2$)

- Es gilt $\mathrm{cond}_p(A) \geq 1$.
- Die Kondition mißt die Empfindlichkeit der Lösung x von $Ax = b$ gegenüber Störungen von A und b.
- Ist $\mathrm{cond}_p(A) >> 1$, so ist die Matrix beinahe singulär. Die Matrix ist *schlecht konditioniert*.

Beispiele

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

```
>> 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
```

Aufbau

Lineare Gleichungssysteme

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

$$Ax = b$$

wird gelöst durch x=A\b| solve(A,b).

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

```
x = ones((5,1)); H = hilbert(5); b = dot(H,x); y = solve(
    H,b)
```

```
y = 1.0000    1.0000    1.0000    1.0000    1.0000
```

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

LU-Zerlegung

Welche Verfahren werden benutzt?

Meistens 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, Pseudoinverse, Determinante

- Berechnung der Inversen

```
inv(A)
```

- (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
```

- Berechnung der Determinante

```
det(A)
```

Aufbau

Zwei-Punkt-Randwert-Aufgabe

Suche eine Funktion

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

so dass

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

Problem: Es kann i.A. keine geschlossene Lösungsdarstellung angegeben werden.
Ausweg: Approximation der Lösung.

Finite Differenzen Verfahren

- Diskretisierung: $0 = x_0 < \cdots < x_n = 1$ 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

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

  Randbedingungen $\Rightarrow$ $u(x_0) = u(x_n) = 0.$
- $\Rightarrow$ Lineares Gleichungssystem für $u(x_1), \ldots, u(x_{n-1})$.

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

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

Implementation für $n = 21$

- Zerlegung des Intervalls $[0, 1]$

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

- Eleminieren der Randpunkte

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

- Erzeugen der Matrix A (Übungsaufgabe)

Lösung für $n = 21$

- Berechnen der rechten Seite:

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

- Lösen des linearen Gls.

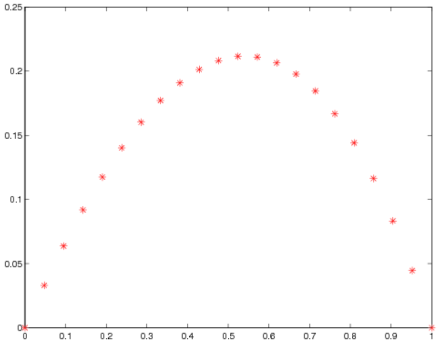
```
z_i = A\F;
```

- Zufügen der Werte am Rand

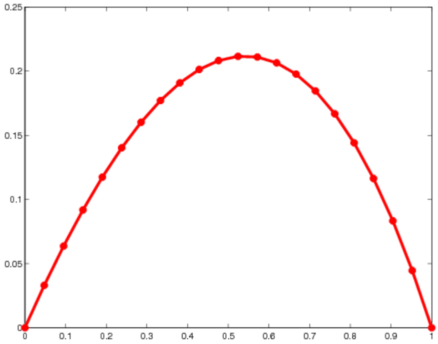
```
z = [0; z_i;0];
```

Lösung für $n = 21$

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




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



Aufbau

Eigenwerte

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

Bemerkungen

- LGS können auch mit Hilfe iterativer Verfahren gelöst werden:
  - gmres (generalized minimum residual)
  - pcg| cg (preconditioned conjugate gradient)
  - bicgstab (biconjugate gradients stabilized)
  - Jacobi
  - overrelaxed Gauss-Seidel (SOR)
  - ...

Aufbau

Dünnbesetzte Matrizen

- 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.
- eigener Datentyp, 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
```

```
import scipy.sparse as sparse
A = 2*diag(ones((10,)),0)+diag(ones((9,)),-1)+diag(ones
    ((9,)),1)
sparse.csr_matrix(A)
```

```
<10x10 sparse matrix of type '<type 'numpy.float64'>'
  with 28 stored elements in Compressed Sparse Row format>
```

Einige Befehle

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

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

```
sparse.csr_matrix((n,m))
```

- Konvertierung der dichtbesetzten Matrix A in eine dünnbesetzte Matrix.

```
sparse(A)
```

```
sparse.csr_matrix(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)
```

```
A.todense()
```

- 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>)
```

```
norm(<A>), cond(<A>)
```

- Alle iterativen Verfahren funktionieren auch mit dünnbesetzten Matrizen (oder haben eine spezielle Version)
- Indizes aller Zeilen und Spalten erhalten, in denen Nichtnullelemente stehen:

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

- Eine Übesicht aller Funktionen für dünnbesetzte Matrizen erhält man durch help sparfun | scipy.sparse .