Einführung in Matlab - Einheit 5

Mehrdimensionale Arrays, Funktionen, Numerische lineare Algebra, Dünnbesetzte Matrizen

Jochen Schulz

Georg-August Universität Göttingen

Aufbau

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.

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.

Aufbau

Funktionen

- 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.
- Bezeichner: Buchstaben mit 1-63 Zeichen (Ohne -,+,*,/ !).

Function-Handles

Ein Function Handle ist ein 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`)

Aufbau

Ein-/Ausgabe - Parameter

- 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 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
```

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)));
```

Aufbau

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

Aufbau

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*

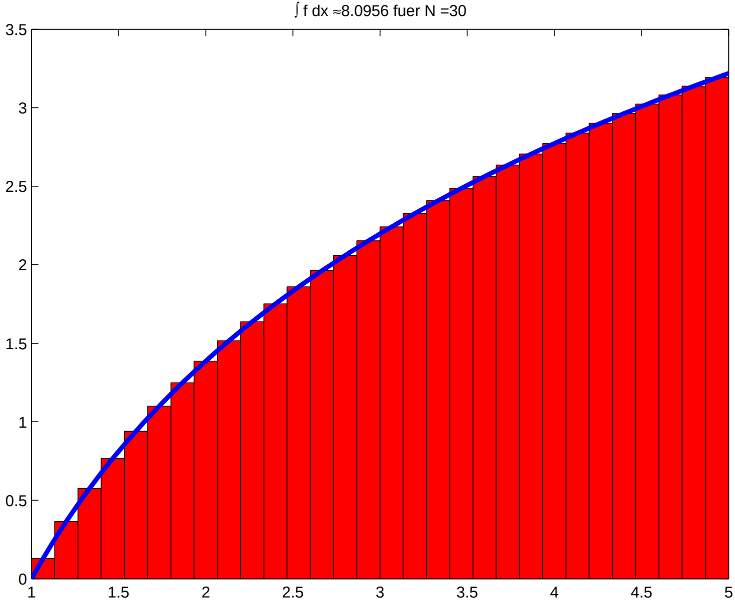
```
vectorize(<f>)
```

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

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

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: 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
```

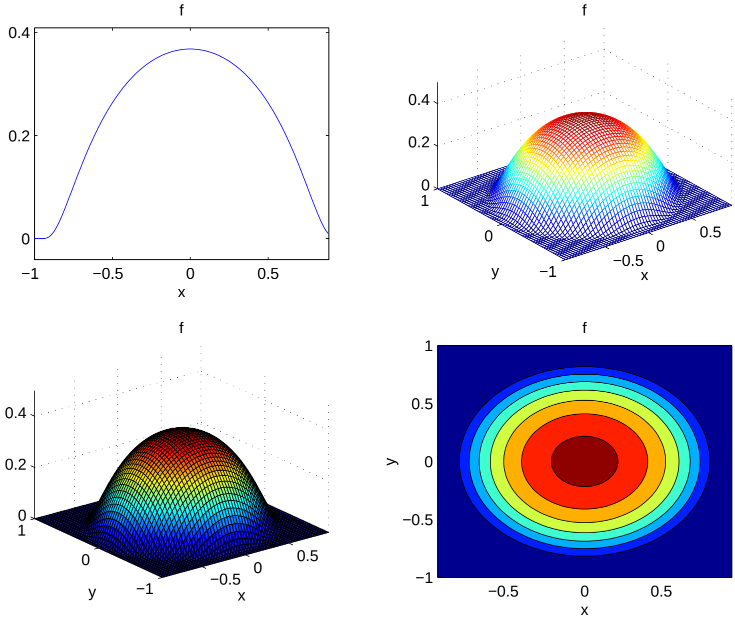
n-dimensionaler-Fall

```
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;
```

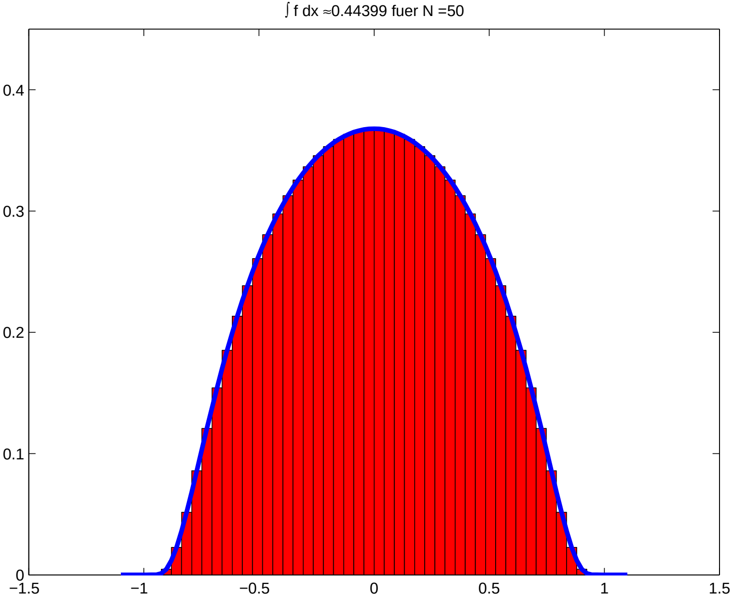
Programm zum Plotten

```
%    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),
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: $p = 2$)
$x = (x_1, \ldots, x_n)$

- 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

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

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:

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

```
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 $\text{cond}_p(A) >> 1$, 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 in MATLAB gelöst durch `x=A\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=inv(A)*b`, da das Berechnen von A^{-1} sehr aufwendig sein kann.

LU-Zerlegung

Was bedeutet `A\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
```

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

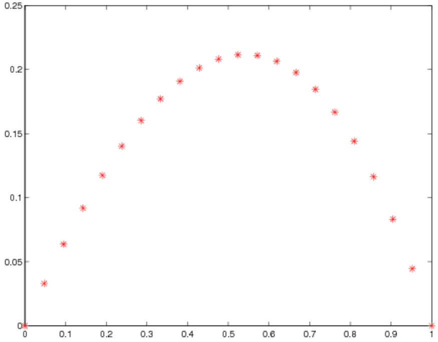
- Zufügen der Werte am Rand

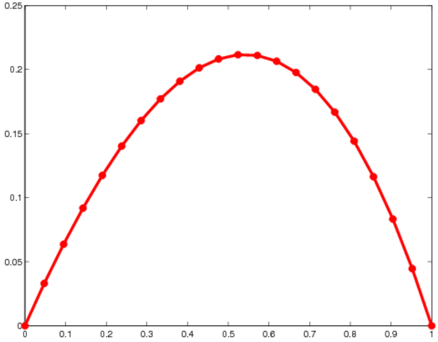
```
z = [0; z_i;0];
```

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)`
  $A = R^* R$ 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` (preconditioned conjugate gradient)
  - `bicgstab` (biconjugate gradients stabilized)
- $A \in \mathbb{C}^{n \times m}$, $n \neq m$ bei `A\b`:
  - $n > m$ (überbestimmter Fall): Least-Square Lösung, d.h. der Ausdruck `norm(A*x-b)` wird minimiert.
  - $n < m$ (unterbestimmter Fall): Grundlösung.

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.
- 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 Übesicht aller Funktionen für dünnbesetzte Matrizen erhält man durch `help sparfun`.