MATLAB Tutorial

Fabian Müller

Seminar for Applied Mathematics, ETH Zurich

Eingabe von Befehlen

- Operationsfenster: Eingabe von Befehlen möglich.

```
>> A = rand(4,4)
>> v = rand(4,1)
>> vMin = min(v)
>> [vMin, i] = min(v)
>> help min
>> A/v
>> x = A\v
>> help \
>> help /
>> x*v
>> x.*v
>> x'*v
```

Workspace

- **Workspace:** Alle gespeicherten Variablen: `A`, `v`, `vMin`, `i`

```
>> vMin
>> clear vMin
>> vMin
>> A
>> clear all
>> A
```

Matrizen erzeugen

```
>> A=[1 2 3 4; 5 6 7 8; 9 1 2 3; 4 5 6 7]
>> zeros(2,3)
>> ones(2,3)
>> eye(4)
>> eye(4,2)
>> diag([1 2 3])
>> diag([1 2 3], 2)
>> diag(A)
>> diag(A,-2)
>> B = [zeros(2,2) eye(2); ...
        ones(2,2)  rand(2,2)]
>> B = [zeros(2,2) eye(2); ...
        [ones(1,3); zeros(1,3)] rand(2,1)]
```

Matrizen manipulieren

- Werte extrahieren und einfügen

```
>> A(1,2)
>> A(3,2) = 10; A
>> 3:10
>> 1.2:.1:2.5
>> A(1,2:4)
>> A(1,:)
>> A(:,1)
>> v = [2 2 2 2];
>> A(:,1) = v
>> A(1,:) = v
```

Lineare Gleichungssysteme lösen I

- Backslash operator \

```
>> A = [1 2 3; 4 5 6; 7 8 9];
>> v = 100*ones(3,1);
>> A\v
```

Lineare Gleichungssysteme lösen II

Backslash löst Gleichungssysteme so optimal wie möglich.
Falls nicht quadratisch: Sucht x, so dass $Bx - v$ so klein wie möglich ist.

```
>> B = [1 2 3 4; 5 6 7 8; 9 1 2 3];
>> x = B\v
>> B*x - v

>> C = [1 2 3 4; 5 6 7 8; 2 4 6 8]
>> x = C\v
>> B*x - v
```

Elementare Funktionen

- Liste der Elementarfunktionen: `help elfun`

```
>> sin(2)
>> sin(pi)
>> sin(Pi)
>> sqrt(4)
>> sqrt(-4)
>> i
>> 1i
>> (25+1i)^2
>> sqrt(2:9)
>> (2:9)^2
>> (2:9).^2
>> help elfun
```

if-Statements I

```
>> k = 20;
>> k>0
>> k<0
>> if k<100
     disp('k<100 hat in MATLAB den Wert 1')
   else
     disp('k<100 hat in MATLAB den Wert 0')
   end
```

if-Statements II

```
>> if (k<100)&(k>30)
       disp('30<k<100')
   elseif (k<100)&(k>20)
       disp('20<k<100')
   elseif k==20
       disp('k=20')
   else
       disp('sonstwas')
   end
```

for-Schleifen I

- Verständlich, aber langsam.

```
>> clear all
>> for i = 1:100
       k(i) = i^2 + 1i;
    end
>> k(12)
>> for i = 1:10
       why
   end
```

for-Schleifen II

```
>> ind = [2 40 1e3];
>> for i = ind
       disp(['i = ', num2str(ind)])
       disp(['sqrt(i) = ',
           num2str(sqrt(ind))])
       if isprime(i)
           disp('i ist Primzahl')
       end
   end
```

while-Schleifen

```
>> i = 1.001;
>> while i<50
       i = i^2

   end

>> clear all
>> count = 0; i=1.001;
>> while i<1e100
       i = i^2;
       count = count + 1;
   end
>> count
>> i
```

Timing I

```
>> A = rand(5000);
>> tic;
>> [L,U] = lu(A);
>> toc;
>> tic;
>> [L,U] = lu(A);
>> t=toc;
>> t
```

Timing II

```
>> clear all
>> tic;
>> A = ones(1e4, 1e4);
>> toc;
>> B = zeros(1e4,1e4);
>> tic;
>> for i = 1:1e4        % Zeilen
       for j = 1:1e4        % Spalten
           B(i,j) = 1;
       end
   end
>> toc;
```

Timing III

Das schlimmste Beispiel:

```
>> tic;
>> for i = 1:5e3        % Zeilen
       for j = 1:5e3        % Spalten
           C(i,j) = 1;
       end
   end
>> toc;
```

2D Plots I

- 2-dimensionale plots: 1d-Punkte vs. 1d-Werte

```
>> N = 1000;
>> x = linspace(-7,7, N);
>> y = (abs(x)<1).*exp(-1./(1-x.^2));
>> figure(1)
>> plot(x,y)
>> close 1
```

2D Plots II

- Optionen des **plot** Befehls

```
>> x2 = -2:2; y2 = 0.5*(-1).^x2;
>> plot(x, y, 'r-', x2, y2, 'bo--')
>> axis([-2 2 -1 1.1])
>> legend('Bump', 'Blue dots with dashed lines')
>> title('ETH Collection of modern art')
>> text(-1.8, -.8, 'F. Mueller: Bump and blue dots with dashed lines, 2014' )
>> xlabel('x-axis'); ylabel('y-axis')
>> help plot
```

2D Plots III

- Mehrere plots in ein Fenster:
  subplot(#Zeilen, #Spalten, index)

```
>> figure()
>> subplot(3,2,1)
>>  plot(x, sin(x))
>> subplot(3,2,3)
>>  plot(x, x.^2, 'r-.')
>> subplot(3,2,2)
>>  plot(rand(10,1), rand(10,1), 'mo')
>> subplot(3,2, 5:6)
>>  plot(x, sin(1./x), 'b-')
```

3D Plots I

- Kurven in 3D: $\gamma : [a, b] \to \mathbb{R}^3$

```
>> t = -5:.005:5;   % Parameter
>> x = (1+t.^2).*sin(20*t);
>> y = (1+t.^2).*cos(20*t);
>> z = t;
>> plot3(x, y, z)
>> grid on
```

3D Plots II

- Flächen in $\mathbb{R}^3$: Graph von Funktionen $\mathbb{R}^2 \to \mathbb{R}$
- **mesh**, **surf**, **contour**: Drei Matrix-inputs `X`,`Y`,`F`.
- Zeichnet den Wert `F(i,j)` an den Punkt mit Koordinaten (`X(i,j)`, `Y(i,j)`).
- Gegeben: Vektoren `x`, `y` mit Koordinaten. Ziel: Fläche der Funktion `F(i,j)` $= F(x(i), y(j))$. $\to$**meshgrid**

```
>> x = 0:.2:1;  y = -1:.25:1;
>> [X, Y] = meshgrid(x, y)
```

3D Plots III

surf, mesh, contour:

```
>> x = 0:.01:1;  y = -1:.05:1;
>> [X, Y] = meshgrid(x, y);
>> Z = sin(pi*X).*(1-Y.^2);
>> figure()
>> surf(X, Y, Z)
>> figure()
>> mesh(X, Y, Z)
>> figure()
>> contour(X,Y,Z)
```

3D Plots IV

Optionen von surf und mesh

- **surf** ohne Gitterlinien: **shading**
- Perspektive ändern: **view()**
- Farblegende hinzufügen: **colorbar**

```
>> figure()
>> surf(X, Y, Z)
>> view(2)
>> shading flat
>> shading interp
>> colorbar
>> caxis([-1, 1])
```

.m-Dateien I

- Einen Befehlsablauf als MATLAB-Script speichern:
  Endung **.m**
- Aufruf vom Kommandofenster ohne Endung.
- Vereinfacht Code-Gestaltung.
- Längere Codes: Kommentare und Formatierung
- Zwei Kommentarzeichen: Der Editor markiert den Bereich zwischen zwei %% als einen Block.

Funktionen I

- Funktion: Nimmt Inputs, produziert Outputs.

```
function y = Quadrat(x)
    y = x.^2;
return
```

- Funktionen müssen als .m-Dateien abgespeichert werden.
- Der Dateiname muss derselbe wie der Funktionenname sein! (Hier: **Quadrat.m**)
- MATLAB sucht nach Funktionen innerhalb des aktuellen Ordners oder des PATH.
- `addpath` fügt Ordner zum PATH hinzu.

Funktionen II

- Aufrufbar von Kommandozeile oder Scripts, **falls Script und Funktion in demselben Ordner liegen.**
- Call-files: Zu einer Funktion ein Script, das die Funktion mit Parametern aufruft.
- Variablen, die kein Output sind: Verloren.

Funktionen III

Beispiel: Leibniz-Reihe

- **Madhava, 14. Jhd.:** $4 \sum_{k \geq 0} \frac{(-1)^k}{(2k+1)} = \pi$
- Liefert Möglichkeit, π durch endliche Summen zu approximieren. Wie schnell?
- Zuerst: Schreibe Approximationsfunktion

```
function piApprox = piLeibniz1(N)
   piApprox = 1; % Erster Summenterm
   for k = 1:N
      piApprox = piApprox + (-1)^k/(2*k+1);
   end
   piApprox = 4*piApprox;
return
```

Funktionen IV

Call-File

```
%Call-File fuer Pi-Approximation
clear all

%% INPUT
N = ...;

%% Berechnung von Werten
piApprox = piLeibniz2(N);
fehler = abs(piApprox-pi);

format long
disp([pi 0; piApprox fehler])
```

Funktionen V

- Konvergiert sehr langsam gegen π!
- Für hohe `N` verlangsamt die `for`-Schleife den Code.
- Stichwort: **Vektorisieren!**

```
function piApprox = piLeibniz2(N)
    ind = 0:N;
    piApprox = 4*sum((-1).^ind./(2*ind+1));
return
```

Funktionen VI

```
clear all
N = 1000:1000:3e5;
t = 0*N;

for k = 1:length(N)
    tic;  piLeibniz1(N(k));
    t1(k) = toc;
    tic;  piLeibniz2(N(k));
    t2(k) = toc;
end
plot(N, t1, 'k-', N, t2, 'b-')
legend('Slow', 'Fast')
```

Funktionen VII

Fehler plotten:

```
clear all
N = 1000:1000:3e5;
fehler = 0*N;

for k = 1:length(N)
    fehler(k) = abs(pi- piLeibniz2(N(k)));
end
plot(N, fehler, 'ko--')
xlabel('N'); ylabel('Fehler')
```

Funktionen VIII

- Wie verhält sich der Fehler $\epsilon_N = |\pi - \pi_N|$?
- Vermutung: $\epsilon_N \simeq N^{-\alpha}$ mit einem $\alpha > 0$.
- Nicht klar sichtbar in einem Plot!
- **Idee:** $\log(\epsilon_N) \simeq (-\alpha)\log(N) + c$.

  $\Rightarrow$ Mit $y := \log(\epsilon_N)$ und $x := \log(N)$ suchen wir die Steigung einer Geraden $y = -Nx + c$!
- MATLAB: **loglog** anstatt plot:
  `loglog(N, fehler, 'ko--')`

Rekursionen I

- Rekursion: Aufruf desselben Programms im Programm
- Meistens ist Iteration weniger aufwändig.
- Oftmals ist Rekursion nötig.

Rekursionen II

```
function nF = fakultaet(n)
if n==0
    nF = 1;
else
    nF = n*fakultaet(n-1);
end
return
```

Rekursionen III

```
function nF = fakultaet2(n)
nF = 1;
for k=1:n
    nF=k*nF;
end
end
```

Noch einfacher: `nF = prod(1:n);`

Beispiel: Sphären zeichnen I

- Einheitssphäre: Für $p \in \mathbb{N}$ ist

$$B_1(p) := \{(x, y) \in \mathbb{R}^2 \,:\, |x|^2 + |y|^2 = 1\} \,.$$

Menge der Vektoren mit "p-Länge"$= 1$.