ETH zürich

# Matlab Tutorial

# Eingabe von Befehlen

- Operationsfenster: Eingabe von Befehlen möglich

```
1  >> A = rand(4,4)
2  >> v = rand(4,1)
3  >> vMin = min(v)
4  >> [vMin, i] = min(v)
5  >> help min
6  >> A/v
7  >> x = A\v
8  >> help \
9  >> help /
10 >> x*v
11 >> x.*v
12 >> x'*v
```

# Workspace

- Workspace: Alle gespeicherten Variablen sichtbar

```
1 >> vMin
2 >> clear vMin
3 >> vMin
4 >> A
5 >> clear all
6 >> A
```

# Matrizen erzeugen

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

# Matrizen manipulieren

- Werte extrahieren und einfügen

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

# Lineare Gleichungssysteme lösen I

- Backslash operator \

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

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

```
1  >> k = 20;
2  >> k>0
3  >> k<0
4  >> if k<100
5       disp('k<100 hat in MATLAB den Wert 1')
6     else
7       disp('k<100 hat in MATLAB den Wert 0')
8     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

- Verständlich, aber langsam.

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

# while-Schleifen

```
1  >> i = 1.001;
2  >> while i<50
3         i = i^2
4
5     end
6
7  >> clear all
8  >> count = 0; i=1.001;
9  >> while i<1e100
10        i = i^2;
11        count = count + 1;
12    end
13 >> count
14 >> i
```

# Timing I

```
1 >> A = rand(5000);
2 >> tic;
3 >> [L,U] = lu(A);
4 >> toc;
5 >> tic;
6 >> [L,U] = lu(A);
7 >> t=toc;
8 >> t
```

# Timing II

```
1  >> clear all
2  >> tic;
3  >> A = ones(1e4, 1e4);
4  >> toc;
5  >> B = zeros(1e4,1e4);
6  >> tic;
7  >> for i = 1:1e4       % Zeilen
8         for j = 1:1e4       % Spalten
9             B(i,j) = 1;
10        end
11    end
12 >> toc;
```

# 2D Plots I

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

```
1 >> N = 1000;
2 >> x = linspace(-7,7, N);
3 >> y = (abs(x)<1).*exp(-1./(1-x.^2));
4 >> figure(1)
5 >> plot(x,y)
6 >> 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 einem Fenster: subplot

```
>> 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

- Kurven in 3D

```
>> 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
```

# .m Dateien

- Einen Befehlsablauf als MATLAB-Script speichern

- Aufruf vom Kommandofenster ohne Endung
- Vereinfacht Code-Gestaltung

# Funktionen I

- Funktion: Input, Output

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

- Funktionen müssen als .m-Datei abgespeichert werden
- Dateiname gleich wie Funktionsname! (Quadrat.m)
- MATLAB sucht nach Funktionen innerhalb aktuellen Ordners

# Funktionen II

- Aufrufbar von Kommandozeile oder Script, falls Script und Funktion im gleichen Ordner
- Call-files: Script zu einer Funktion, welches die Funktion mit Parametern aufruft
- Variablen, die kein Output der Funktion sind gehen verloren.

# Funktionen III

- Beispiel Leibniz-Reihe
  - Approximiert PI durch endliche Summe

$$4 \sum_{k \geq 0} \frac{(-1)^k}{(2k+1)} = \pi$$

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

# Funktionen IV

- Call-File:

```
1  %Call-File fuer Pi-Approximation
2  clear all
3
4  %% INPUT
5  N = ...;
6
7  %% Berechnung von Werten
8  piApprox = piLeibniz2(N);
9  fehler = abs(piApprox-pi);
10
11 format long
12 disp([pi 0; piApprox fehler])
```

# Funktionen V

- 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')
```

# Rekursion I

- Rekursion: Aufruf desselben Programms im Programm

```
1 function nF = fakultaet(n)
2 if n==0
3     nF = 1;
4 else
5     nF = n*fakultaet(n-1);
6 end
7 return
```

# Rekursion II

- Gleiche Berechnung iterativ gelöst:

```
1  function nF = fakultaet2(n)
2  nF = 1;
3  for k=1:n
4      nF=k*nF;
5  end
6  end
```