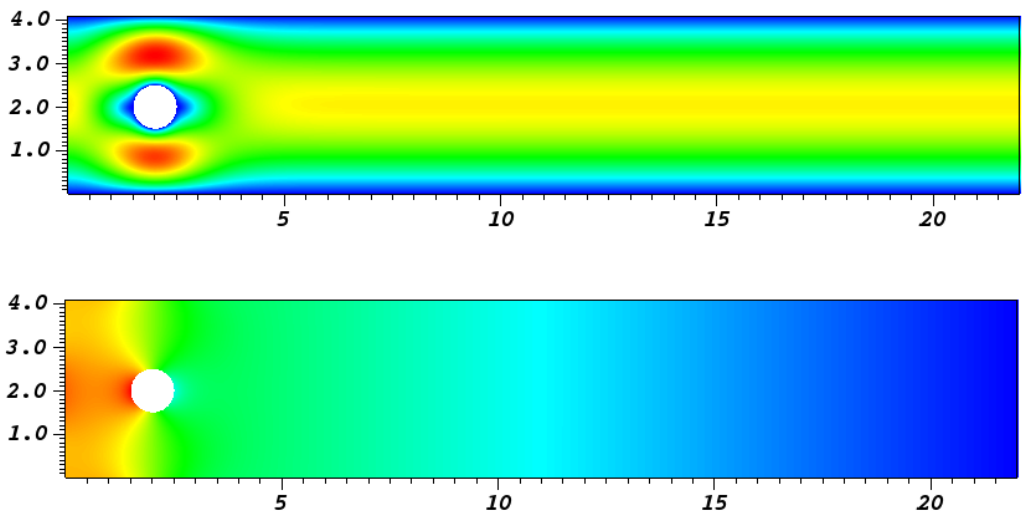
Einführung in MATLAB / GNU Octave

Philipp Siehr

Heidelberg

01. Oktober 2014



Inhalt

- Organisatorisches
- Erste Schritte mit Linux.
- Was ist MATLAB bzw. Octave ?
- Erste Schritte mit MATLAB.
- Zuweisung von Werten an Variablen.
- Anlegen und Bearbeiteten von Matrizen.
- Operatoren auf Matrizen.
- Anlegen von Funktionen.
  - Einfache Funktionen
  - Schleifen und Verzweigungen

Dieser Workshop basiert zu Teilen auf der Numerik 0 Einführung von Klinger, Vihharev.

Organisatorisches

Organisatorisches

- Accounts: vkmp0XX mit XX=Rechnernummer
  Bitte nicht die Passwörter ändern.
- Material: www.mathsim.eu/~psiehr/links.html

Erste Schritte mit Linux

- Wechsel in euer Home-Verzeichnis (~)
  `cd ~/Dokumente/`, benutzt autofill: `cd ~/Dok +Tab`

- Wechsel in euer Home-Verzeichnis (~)
  `cd ~/Dokumente/`, benutzt autofill: `cd ~/Dok +Tab`
- Dateien und Verzeichnisse anzeigen lassen:
  `ls`, bzw. optional: `ls -l`

- Wechsel in euer Home-Verzeichnis (~)
  `cd ~/Dokumente/`, benutzt autofill: `cd ~/Dok +Tab`
- Dateien und Verzeichnisse anzeigen lassen:
  `ls`, bzw. optional: `ls -l`
- Ein Verzeichnis „TestOrdner“ anlegen:
  `mkdir TestOrdner`
- Benutzt abermals `ls`. Was hat sich geändert?

- Wechsel in euer Home-Verzeichnis (~)
  `cd ~/Dokumente/`, benutzt autofill: `cd ~/Dok +Tab`
- Dateien und Verzeichnisse anzeigen lassen:
  `ls`, bzw. optional: `ls -l`
- Ein Verzeichnis „TestOrdner“ anlegen:
  `mkdir TestOrdner`
- Benutzt abermals `ls`. Was hat sich geändert?
- In das Verzeichnis wechseln:
  `cd TestOrdner`
- Wieder `ls` verwenden. Beobachtung?

- Wechsel in euer Home-Verzeichnis (~)
  `cd ~/Dokumente/`, benutzt autofill: `cd ~/Dok +Tab`
- Dateien und Verzeichnisse anzeigen lassen:
  `ls`, bzw. optional: `ls -l`
- Ein Verzeichnis „TestOrdner“ anlegen:
  `mkdir TestOrdner`
- Benutzt abermals `ls`. Was hat sich geändert?
- In das Verzeichnis wechseln:
  `cd TestOrdner`
- Wieder `ls` verwenden. Beobachtung?
- In das übergeordnete Verzeichnis zurück wechseln:
  `cd ..`
- `ls`. Und jetzt?

- Wechsel in euer Home-Verzeichnis (~)
  `cd ~/Dokumente/`, benutzt autofill: `cd ~/Dok +Tab`
- Dateien und Verzeichnisse anzeigen lassen:
  `ls`, bzw. optional: `ls -l`
- Ein Verzeichnis „TestOrdner“ anlegen:
  `mkdir TestOrdner`
- Benutzt abermals `ls`. Was hat sich geändert?
- In das Verzeichnis wechseln:
  `cd TestOrdner`
- Wieder `ls` verwenden. Beobachtung?
- In das übergeordnete Verzeichnis zurück wechseln:
  `cd ..`
- `ls`. Und jetzt?
- Löschen von Dateien und Verzeichnissen:
  `rm -r TestOrdner`
- `ls`. Der Ordner sollte verschwunden sein!

Aufgabe

Erstellt den Ordner `~/Dokumente/vorkurs`, und öffnet darin MATLAB mit:

```
matlab &
```

Was ist MATLAB / GNU Octave?

MATLAB

- Kommerzielle Software zur Lösung numerischer Probleme:
  - Numerik 0: Lineare Gleichungssysteme
  - Numerik 1: Gewöhnliche Differentialgleichungen
  - Numerik 2: Partielle Differentialgleichungen

MATLAB

- Kommerzielle Software zur Lösung numerischer Probleme:
  - Numerik 0: Lineare Gleichungssysteme
  - Numerik 1: Gewöhnliche Differentialgleichungen
  - Numerik 2: Partielle Differentialgleichungen

GNU Octave

- Kostenfreies Pendant zu MATLAB.
- Besitzt keine eigene Oberfläche (GUI).
- Verwendet gnuplot
- Start mit

```
octave &
```

Erste Schritte mit MATLAB

Newtonverfahren

Mit dem Newtonverfahren ist die approximative Bestimmung der Nullstellen einer Funktion möglich. Was genau dahinter steckt kommt später.
$\rightarrow$ Tafel

Variablen

- Variablen werden durch die Zuordnung von Werten angelegt.
- Variablennamen müssen mit einem Buchstaben anfangen, ansonsten dürfen Buchstaben, Zahlen und Unterstriche benutzt werden.
- Es wird zwischen Groß- und Kleinschreibung unterschieden ($A \neq a$).
- Variablen sollten sinnvolle Namen haben.

Ab jetzt steht nur auf den Folien ein „>“ vor MATLAB Code

- Variable anlegen

```
> a=42.42;
```

- Variable ausgeben

```
> a
```

  ergibt

```
a=42.42
```

- Das Semikolon ; unterdrückt die Ausgabe.

Bis auf wenige Ausnahmen sind die Werte, die wir Variablen in MATLAB zuordnen können, $n \times m$ Matrizen. Ein Skalar ist dann eine 1×1 Matrix. Zeilenvektoren sind $1 \times m$ Matrizen und Spaltenvektoren $n \times 1$ Matrizen.

Matrizen

Beispiel

Finde den Schnittpunkt der Geraden:

$$f(x) = 3x + 5,$$
$$g(x) = 6x + 7.$$

Beispiel

Finde den Schnittpunkt der Geraden:

$$f(x) = 3x + 5,$$
$$g(x) = 6x + 7.$$

Tafel:

$$\begin{pmatrix} 1 & -3 \\ 1 & -6 \end{pmatrix} \begin{pmatrix} y \\ x \end{pmatrix} = \begin{pmatrix} 5 \\ 7 \end{pmatrix}$$

Schnittpunkt:

$$(x, y) = (-\tfrac{2}{3}, 3)$$

Vektoren anlegen

a) Komponentenweise eingeben

b) Mit (vorgegebenen) Funktionen sukzessive füllen.

Vektoren anlegen

a) Komponentenweise eingeben

b) Mit (vorgegebenen) Funktionen sukzessive füllen.

- `> a=[1,2,3]` erzeugt `a=`$(1\ 2\ 3)$.
- `> b=[1;2]` erzeugt `b=`$\begin{pmatrix}1\\2\end{pmatrix}$.

Vektoren anlegen

a) Komponentenweise eingeben

b) Mit (vorgegebenen) Funktionen sukzessive füllen.

- `> a=[1,2,3]` erzeugt `a=`$(1\ 2\ 3)$.
- `> b=[1;2]` erzeugt `b=`$\begin{pmatrix}1\\2\end{pmatrix}$.
- `> c=1:5` erzeugt `c=`$(1\ 2\ 3\ 4\ 5)$.
- `> d=1:2:5` erzeugt `d=`$(1\ 3\ 5)$.

Vektoren anlegen

a) Komponentenweise eingeben

b) Mit (vorgegebenen) Funktionen sukzessive füllen.

- `> a=[1,2,3]` erzeugt `a=`$(1\ 2\ 3)$.
- `> b=[1;2]` erzeugt `b=`$\begin{pmatrix}1\\2\end{pmatrix}$.
- `> c=1:5` erzeugt `c=`$(1\ 2\ 3\ 4\ 5)$.
- `> d=1:2:5` erzeugt `d=`$(1\ 3\ 5)$.
- Länge von Vektoren

```
>length(a)
ans= 3
>length(c)
ans= 5
```

Vektoren bearbeiten

- Lese-/Schreibzugriff auf die i-te Komponente von a über $a(i)$:

```
> e=11:15     e=(11 12 13 14 15)
> val=e(3)    val=13
> e(3) = 42   e=(11 12 42 14 15)
```

Vektoren bearbeiten

- Lese-/Schreibzugriff auf die i-te Komponente von a über $a(i)$:

```
> e=11:15     e=(11 12 13 14 15)
> val=e(3)    val=13
> e(3) = 42   e=(11 12 42 14 15)
```

- Subvektoren:

```
> f=-5:-1             f=(-5 -4 -3 -2 -1)
> g=f(2:4)            g=(-4 -3 -2)
> f(2:4) = [5,5,5]    f=(-5 5 5 5 -1)
```

Vektoren bearbeiten

- Lese-/Schreibzugriff auf die i-te Komponente von a über $a(i)$:

```
> e=11:15     e=(11 12 13 14 15)
> val=e(3)    val=13
> e(3) = 42   e=(11 12 42 14 15)
```

- Subvektoren:

```
> f=-5:-1             f=(-5 -4 -3 -2 -1)
> g=f(2:4)            g=(-4 -3 -2)
> f(2:4) = [5,5,5]    f=(-5 5 5 5 -1)
```

- Mit $'$ werden Vektoren transponiert

> g$'$ liefert $\begin{pmatrix} -4 \\ -3 \\ -2 \end{pmatrix}$

Matrizen erstellen

Wie bereits erwähnt sind Vektoren spezielle Matrizen. Also lassen sich auch diese entsprechend befüllen.

Matrizen erstellen

Wie bereits erwähnt sind Vektoren spezielle Matrizen. Also lassen sich auch diese entsprechend befüllen.

- `> A=[1,2,3;4,4,4;0,0,0]` erzeugt `A=` $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 4 & 4 \\ 0 & 0 & 0 \end{pmatrix}$

Matrizen erstellen

Wie bereits erwähnt sind Vektoren spezielle Matrizen. Also lassen sich auch diese entsprechend befüllen.

- `> A=[1,2,3;4,4,4;0,0,0]` erzeugt `A=` $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 4 & 4 \\ 0 & 0 & 0 \end{pmatrix}$
- `> v=[1,2]`
  `B=[v; 5,-32]` erzeugt `B=` $\begin{pmatrix} 1 & 2 \\ 5 & -32 \end{pmatrix}$

Matrizen erstellen

- Weitere Befehle zum Befüllen von Matrizen.
  - Einheitsmatrix `> I=eye(3)` erzeugt `B=` $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$
  - Nullmatrix `> N=zeros(3)` erzeugt `N=` $\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$
  - Einsmatrix `> E=ones(3)` erzeugt `N=` $\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$
  - $n \times m$ Nullmatrix `> N=zeros(2,3)` erzeugt `N=` $\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$
  - $n \times m$ Einsmatrix `> N=zeros(1,3)` erzeugt `N=` $\begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$

Aufgabe

Berechnet den Schnittpunkt der beiden Geraden aus dem Beispiel.
Wenn A, b angelegt sind, dann kann MATLAB mit `> x=A\b` das LGS lösen.

Aufgabe

Berechnet den Schnittpunkt der beiden Geraden aus dem Beispiel. Wenn A, b angelegt sind, dann kann MATLAB mit `> x=A\b` das LGS lösen.

Lösung

```
> A=[1,-3;1,-6];
> b=[5;7];
> x=A\b
```

Matrizen bearbeiten

Sei nun $C = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$

Sei nun $C = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$

- Anzahl der Zeilen (n) und Spalten (m).

```
> [n,m]=size(C).
```

Matrizen bearbeiten

Sei nun $C = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$

- Anzahl der Zeilen (n) und Spalten (m).

  `> [n,m]=size(C)`.

- Lese-/Schreibzugriff auf Eintrag (i,j):

  `> val=C(i,j)` `> C(i,j)=5`.

Matrizen bearbeiten

Sei nun $C = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$

- Anzahl der Zeilen (n) und Spalten (m).
  `> [n,m]=size(C)`.
- Lese-/Schreibzugriff auf Eintrag (i,j):
  `> val=C(i,j)` `> C(i,j)=5`.
- Zugriff auf i-te Zeile, j-te Spalte mit „`:`“
  `> w=(1,:)` liefert: $\begin{pmatrix} 1 & 2 & 3 & 4 \end{pmatrix}$
  `> w=(1,3:4)` liefert: $\begin{pmatrix} 3 & 4 \end{pmatrix}$
  `> C(:,4)=[]` liefert: $C = \begin{pmatrix} 1 & 2 & 3 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

Aufgabe

Erzeugt mit der Matrix C die Matrix $C2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$.

Aufgabe

Erzeugt mit der Matrix C die Matrix $C2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$.

Lösung

```
> C2=C(2:3,2:3)
```

MATLAB Funktionen

octave:1> <EINGABE>	Kurze Erläuterung
`abs(·)`	Der Absolutbetrag (Matrizen: Komponentenweise)
`sqrt(·)`	Die Wurzel
`exp(·)`	Exponentialfunktion
`sin(·)`, `cos(·)`, etc.	Trigonometrische Funktionen
`diag(·)`	Erzeugt eine Diagonalmatrix (z.B. `diag(1:5)`)
`fliplr(·)`, `flipud(·)`	Spiegelt die Matrix (vertikal, horizontal)
`sum(·)`	Summiert einen Vektor oder alle Spalten einer Matrix
`who`	Variablenbelegung (Was ist alles schon erzeugt?)
`help <Function>`	Informationen über die Funktion (z.B. `help sum`)
`help`	Übersicht über sämtliche Funktionen von Octave

Aufgabe

Erzeugt mit dem Befehl `diag(1:5)` diese 10×10 Matrix F

$$F = \begin{pmatrix} 4 & -1 & & & \\ -1 & 4 & \ddots & & \\ & \ddots & \ddots & -1 & \\ & & -1 & 4 & \end{pmatrix}.$$

Bestimmt die Summe aller Spalten und der gesammten Matrix.

Aufgabe

Erzeugt mit dem Befehl `diag(1:5)` diese 10×10 Matrix F

$$F = \begin{pmatrix} 4 & -1 & & \\ -1 & 4 & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 4 \end{pmatrix}.$$

Bestimmt die Summe aller Spalten und der gesammten Matrix.

Lösung

```
> F=diag(4*ones(10,1)) + diag(-1*ones(9,1),1) +
diag(-1*ones(9,1),-1);
> sum(F)
> sum(sum(F))
```

Grafische Ausgabe

Grafische Ausgabe

- Zu einem gegebenen Vektor mit Stützstellen (x) und Funktionswerten $(f(x) = y)$ kann man mit `> plot(x,y)` ein buntes Bild erzeugen.

Grafische Ausgabe

- Zu einem gegebenen Vektor mit Stützstellen (x) und Funktionswerten $(f(x) = y)$ kann man mit `> plot(x,y)` ein buntes Bild erzeugen.
  1) `> x=0:0.01:10; > y=x.^2; > plot(x,y)`
  2) `> x=0:0.01:10; > Y=[x', (x.^2)',(x.^3)'] ; > plot(x,Y)`

Grafische Ausgabe

- Zu einem gegebenen Vektor mit Stützstellen (x) und Funktionswerten $(f(x) = y)$ kann man mit `> plot(x,y)` ein buntes Bild erzeugen.
  1) `> x=0:0.01:10; > y=x.^2; > plot(x,y)`
  2) `> x=0:0.01:10; > Y=[x', (x.^2)',(x.^3)'] ; > plot(x,Y)`
- Beschriftung

Titel	`title('titletext');`
Achsen	`xlabel('x-Achsentext');`
	`ylabel('y-Achsentext');`
Legende	`legend('plot1','plot2',...);`

Grafische Ausgabe

- Zu einem gegebenen Vektor mit Stützstellen (x) und Funktionswerten $(f(x) = y)$ kann man mit `> plot(x,y)` ein buntes Bild erzeugen.
  1) `> x=0:0.01:10; > y=x.^2; > plot(x,y)`
  2) `> x=0:0.01:10; > Y=[x', (x.^2)',(x.^3)'] ; > plot(x,Y)`
- Beschriftung

  Titel `title('titletext');`

  Achsen `xlabel('x-Achsentext');`
  `ylabel('y-Achsentext');`

  Legende `legend('plot1','plot2',...);`
- Achsen können mit `semilogy`, `semilogx` und `loglog` logarithmisch skaliert werden. Z.B. erzeugt `semilogy(x,y)` einen plot mit logarithmisch skalierter y-Achse.

Grafische Ausgabe

- Zu einem gegebenen Vektor mit Stützstellen (x) und Funktionswerten $(f(x) = y)$ kann man mit `> plot(x,y)` ein buntes Bild erzeugen.
  1) `> x=0:0.01:10; > y=x.^2; > plot(x,y)`
  2) `> x=0:0.01:10; > Y=[x', (x.^2)',(x.^3)'] ; > plot(x,Y)`
- Beschriftung

  Titel `title('titletext');`
  Achsen `xlabel('x-Achsentext');`
  `ylabel('y-Achsentext');`
  Legende `legend('plot1','plot2',...);`
- Achsen können mit `semilogy`, `semilogx` und `loglog` logarithmisch skaliert werden. Z.B. erzeugt `semilogy(x,y)` einen plot mit logarithmisch skalierter y-Achse.
- Mit `figure(i)`, wobei $i \in \mathbb{N}$ können mehrere Ausgabefenster geöffnet werden. Mit `hold on` können mehrere Plots in einem Fenster ausgegeben werden. Mittels `hold off` deaktiviert man dieses Verhalten.

Aufgabe

Man plotte die Funktion $f(x) = \sin(x)$ im Intevall $[-\pi, \pi]$. Beschriftet die Ausgabe.

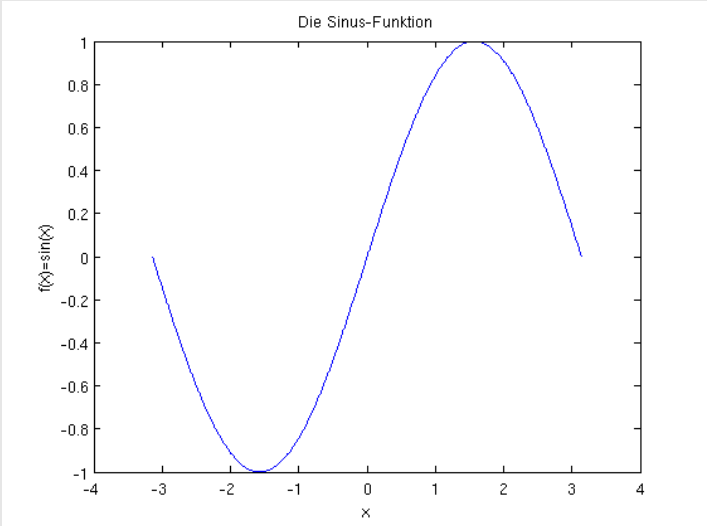
Aufgabe

Man plotte die Funktion $f(x) = \sin(x)$ im Intevall $[-\pi, \pi]$. Beschriftet die Ausgabe.

Lösung

```
> I = -pi:0.01:pi;
> I(1:10)
I =

   -3.1416  ...  -3.0516
> fx = sin(I);
> plot(I,fx);
> title('Die Sinus-Funktion');
> xlabel('x');
> ylabel('f(x)=sin(x)');
```



.m-Files und eigene Funktionen

.m-Files und eigene Funktionen

- Was ist ein .m-File?
- Wie lege ich ein .m-File an?
- Konzept einer Funktion.
- Wie beschreibt man Rekursionen/Iterationen mittels Schleifen und Verzweigungen ?
  - while-Schleife
  - for-Schleife
  - if-Verzweigung

Funktionen

- MATLAB Befehle können in Dateien abgespeichert werden.
- Diese werden als <Name>.m abgespeichert.
- Vorteile:
  - Wiederverwenbarkeit von Funktionen.
  - Zeitersparnis.
  - Übersichtlichkeit.

Funktionen

- MATLAB Befehle können in Dateien abgespeichert werden.
- Diese werden als <Name>.m abgespeichert.
- Vorteile:
  - Wiederverwenbarkeit von Funktionen.
  - Zeitersparnis.
  - Übersichtlichkeit.

- Vorgehen:
  - Öffnet mit einem beliebigen Editor eine neue Datei.
    Bspw. `strg+N` mit MATLAB.
  - Prinzipieller Funktionsaufbau

```
function [<out1>,...,<outN>] = <name>(<in1>,...,<inN>)

   <Funktionsbefehle>

end
```

- Vorgehen:
  - Öffnet mit einem beliebigen Editor eine neue Datei.
    Bspw. `strg+N` mit MATLAB.
  - Prinzipieller Funktionsaufbau

```
function [<out1>,...,<outN>] = <name>(<in1>,...,<inN>)

    <Funktionsbefehle>

end
```

- Vorgehen:
  - Öffnet mit einem beliebigen Editor eine neue Datei.
    Bspw. `strg+N` mit MATLAB.
  - Prinzipieller Funktionsaufbau

```
function [<out1>,...,<outN>] = <name>(<in1>,...,<inN>)

   <Funktionsbefehle>

end
```

  - Diese Datei wird unter dem Namen <name>.m gespeichert.

- Vorgehen:
  - Öffnet mit einem beliebigen Editor eine neue Datei.
    Bspw. `strg+N` mit MATLAB.
  - Prinzipieller Funktionsaufbau

```
function [<out1>,...,<outN>] = <name>(<in1>,...,<inN>)

   <Funktionsbefehle>

end
```

  - Diese Datei wird unter dem Namen <name>.m gespeichert.
  - Wichtig: Die Datei muss genau diesen Namen haben und muss genau im verwendeten Verzeichnis liegen.

Funktionen

- Vorgehen:
  - Öffnet mit einem beliebigen Editor eine neue Datei.
    Bspw. `strg+N` mit MATLAB.
  - Prinzipieller Funktionsaufbau

```
function [<out1>,...,<outN>] = <name>(<in1>,...,<inN>)

    <Funktionsbefehle>

end
```

  - Diese Datei wird unter dem Namen <name>.m gespeichert.
  - Wichtig: Die Datei muss genau diesen Namen haben und muss genau im verwendeten Verzeichnis liegen.
  - Aufruf mit: `<name>(<in1>,...,<inN>)`

Aufgabe

Erstellt eine Funktion ExpPlot die einen Vektor x bekommt und einen Vektor $erg = \mathrm{e}^x$ zurückgibt. Die MATLAB-Funktion soll auch diese Funktion plotten. Verwendet diese Funktion.

Aufgabe

Erstellt eine Funktion ExpPlot die einen Vektor x bekommt und einen Vektor $erg = \mathrm{e}^x$ zurückgibt. Die MATLAB-Funktion soll auch diese Funktion plotten. Verwendet diese Funktion.

Lösung

Das m-File: ExpPlot.m

```
function erg = ExpPlot(x)
erg = exp(x);
plot(x,erg);
end
```

for-Schleife

- Syntax:

```
for <LAUFINDEX> = <START>:<ENDE>
<BEFEHL-1>
<BEFEHL-2>
⋮
<BEFEHL-M>
end
```

for-Schleife

- Syntax:

```
for <LAUFINDEX> = <START>:<ENDE>
<BEFEHL-1>
<BEFEHL-2>
⋮
<BEFEHL-M>
end
```

- Für eine bestimmte Anzahl von Wiederholungen wird dieselbe Gruppe von Befehlen immer wieder ausgeführt.
- Dabei wird ein Laufindex hochgezählt, der ebenfalls in der Befehlsgruppe verwendet werden kann.

for-Schleife

- Syntax:

```
for <LAUFINDEX> = <START>:<ENDE>
<BEFEHL-1>
<BEFEHL-2>
⋮
<BEFEHL-M>
end
```

- Für eine bestimmte Anzahl von Wiederholungen wird dieselbe Gruppe von Befehlen immer wieder ausgeführt.
- Dabei wird ein Laufindex hochgezählt, der ebenfalls in der Befehlsgruppe verwendet werden kann.
- Beispiel:

```
Sum=0;
   for i = 1:10
     Sum = Sum + i
   end
```

Aufgabe

Erstellt eine Funktion die alle geraden Zahlen bis zur Eingabevariablen summiert und zurückgibt.

Aufgabe

Erstellt eine Funktion die alle geraden Zahlen bis zur Eingabevariablen summiert und zurückgibt.

Lösung

Das m-File: MySum.m

```
erg=MySum(x)
Sum=0;
    for i = 0:2:x
      Sum = Sum + i;
    end
```

while-Schleife

- Syntax:

```
while <BEDINGUNG ERFÜLLT>
<BEFEHL-1>
⋮
<BEFEHL-M>
end
```

while-Schleife

- Syntax:

```
while <BEDINGUNG ERFÜLLT>
<BEFEHL-1>
⋮
<BEFEHL-M>
end
```

- Wiederholt eine Gruppe von Octave-Befehlen bis eine bestimmte logische Bedingung nicht mehr erfüllt ist.

while-Schleife

- Syntax:

```
while <BEDINGUNG ERFÜLLT>
<BEFEHL-1>
:
<BEFEHL-M>
end
```

- Wiederholt eine Gruppe von Octave-Befehlen bis eine bestimmte logische Bedingung nicht mehr erfüllt ist.
- Beispiel:

```
Sum=10;
   while Sum>0
     Sum = Sum - 1
   end
```

while-Schleife

- Syntax:

```
while <BEDINGUNG ERFÜLLT>
<BEFEHL-1>
:
<BEFEHL-M>
end
```

- Wiederholt eine Gruppe von Octave-Befehlen bis eine bestimmte logische Bedingung nicht mehr erfüllt ist.
- Beispiel:

```
Sum=10;
   while Sum>0
     Sum = Sum - 1
   end
```

- Hinweis: Ändern wir im obigen Beispiel das Minus durch ein Plus, wird das Abbruchkriterium `Sum>0` nie erfüllt $\rightsquigarrow$ Endlosschleife. Die Endlosschleife kann ebenfalls mittels `<Strg>-C` beendet werden!

IF-VERZWEIGUNG

- Logische Vergleiche – Seien a und b zwei Skalare
  - a gleich b? $a == b$
  - a ungleich b? $a \sim= b$
  - a kleiner/größer gleich b? $a <= b$, bzw. $a >= b$
  - a kleiner/größer b? $a < b$, bzw. $a > b$
- `if`-Verzweigung
  - Syntax:

```
if <BEDINGUNG ERFÜLLT>
<BEFEHL-A>
else
<BEFEHL-B>
end
```

  - Anhand eines logischen Vergleichs wird entweder der eine oder der andere Octave-Befehl ausgeführt.
  - Beispiel:

```
x=10;y=5;
   if (x>y) z=1
   else   z=0
   end
```

Aufgabe

Erstellt eine Funktion die 3 Zahlen x, y, z bekommt und die größte zurückgibt.

Aufgabe

Erstellt eine Funktion die 3 Zahlen x, y, z bekommt und die größte zurückgibt.

Lösung

Das m-File: MyMax.m

```
function erg=MyMax(x,y,z)
    if x>y
      erg = x;
    else if y>z
          erg = y;
          end
      erg = z;
    end
end
```

Euklidischer Algorithmus

Aufgabe

Erstellt eine Funktion `erg=ggT(a,b)` die den größten gemeinsamen Teiler von $a, b \in \mathbb{N}$ bestimmt.

Euklidischer Algorithmus

Aufgabe

Erstellt eine Funktion `erg=ggT(a,b)` die den größten gemeinsamen Teiler von $a, b \in \mathbb{N}$ bestimmt.

Pseudo-Code

```
1    wenn a = 0
2        dann return b
3    sonst solange b ≠ 0
4        wenn a > b
5            dann a ← a − b
6        sonst b ← b − a
7    return a
```

Lösung

Das m-File: Euklid.m

```
function erg=Euklid(a,b)
    if (b==0)
      erg = a;
    else while (b~=0)
         if (a>0)
           a=a-b;
         else (a>0)
           b=b-a;
         end
         erg=a;
    end
end
```

Funktionen II

- An den Beispielen haben wir gesehen, das es sinnvoll ist Algorithmen auszulagern. So muss man den Euklidischen Algorithmus nicht für jedes Zahlenpaar neu schreiben.

Funktionen II

- An den Beispielen haben wir gesehen, das es sinnvoll ist Algorithmen auszulagern. So muss man den Euklidischen Algorithmus nicht für jedes Zahlenpaar neu schreiben.
- Das funktioniert auch mit Funktionen. So kann bspw. das Newtonverfahren für verschiedene Funktionen Nullstellen berechnen.

Funktionen II

- An den Beispielen haben wir gesehen, das es sinnvoll ist Algorithmen auszulagern. So muss man den Euklidischen Algorithmus nicht für jedes Zahlenpaar neu schreiben.
- Das funktioniert auch mit Funktionen. So kann bspw. das Newtonverfahren für verschiedene Funktionen Nullstellen berechnen.
- Sei foo.m eine Funktion.
  Problem: Unterscheidung Matrix und Funktion:

  ```
  erg=algorithm(f,a,b,...)
  ```

  main.m: `algorithm(foo,x,y,...)`

Funktionen II

- An den Beispielen haben wir gesehen, das es sinnvoll ist Algorithmen auszulagern. So muss man den Euklidischen Algorithmus nicht für jedes Zahlenpaar neu schreiben.
- Das funktioniert auch mit Funktionen. So kann bspw. das Newtonverfahren für verschiedene Funktionen Nullstellen berechnen.
- Sei foo.m eine Funktion.
  Problem: Unterscheidung Matrix und Funktion:

  ```
  erg=algorithm(f,a,b,...)
  ```
  main.m: `algorithm(foo,x,y,...)`
- Lösung: Function Handle

  ```
  erg=algorithm(f,a,b,...)
  ```
  main.m: `algorithm(@foo,x,y,...)`

Funktionen II

Aufgabe

Erstellt eine Funktion `[]=MyPlot(f,a,b)`, die eine Funktion im Intervall $[a, b]$ plottet.
Erstellt eine Funktion `[]=main()`.
Ruft dort mit einer beliebigen Funktion `MyPlot` auf.

Lösung

Das m-File: MyPlot.m

```
function []=MyPlot(f,a,b)
  plot(a:0.01:b,f(a:0.01:b))
end
```

Das m-File: main.m

```
function []=main()
a=1; b=2;
MyPlot(@f,a,b)
end
```

Lösung

Das m-File: MyPlot.m

```
function []=MyPlot(f,a,b)
  plot(a:0.01:b,f(a:0.01:b))
end
```

Das m-File: main.m

```
function []=main()
a=1; b=2;
MyPlot(@f,a,b)
end
```

```
plot(x,sin(x))

foo=@(x) x.^2;
plot(x,foo(x))
```

Newtonverfahren

Newtonverfahren (Tafel)

Gegeben: x_0 Startwert

Iteriere:

$$x^{k+1} = x^k - \frac{f(x^k)}{f'(x^k)}$$

Aufgabe I

Erstellt je eine Funktion für f und f' mit

$$f(x) = x^3 - 1.$$

Aufgabe I

Erstellt je eine Funktion für f und f' mit

$$f(x) = x^3 - 1.$$

Aufgabe II

Erstellt eine Funktion `[x]=newton(f,df,x0,max_step)`. Diese soll alle berechneten Werte x^k im Vektor x ausgeben.

Aufgabe I

Erstellt je eine Funktion für f und f' mit

$$f(x) = x^3 - 1.$$

Aufgabe II

Erstellt eine Funktion `[x]=newton(f,df,x0,max_step)`. Diese soll alle berechneten Werte x^k im Vektor x ausgeben.

Aufgabe III

Erstellt eine Funktion `[]=main()` und testet mit 10 Schritten und Startwert 6.

Aufgabe IV

Wie kann man feststellen ob man tatsächlich gegen die Nullstelle konvergiert?

Aufgabe IV

Wie kann man feststellen ob man tatsächlich gegen die Nullstelle konvergiert?

Aufgabe V

Plottet den Fehler mit der Anzahl der Schritte in x-Richtung.
Die y-Skala sollte logarithmisch sein.
Auch das kommt in die `main`.

Aufgabe IV

Wie kann man feststellen ob man tatsächlich gegen die Nullstelle konvergiert?

Aufgabe V

Plottet den Fehler mit der Anzahl der Schritte in x-Richtung.
Die y-Skala sollte logarithmisch sein.
Auch das kommt in die `main`.

Aufgabe VI*

Übergebt dem Newtonverfahren einen Toleranzwert (`ToL`).
Ändert die `for`-Schleife zu einer `while`-Schleife.
Führt eine Zählvariable für die maximale Schrittanzahl ein.

2D-Plot

Surface Plot

```
[X,Y]=meshgrid(-pi:0.1:pi,-pi:0.1:pi);
Z=sin(X+Y);
surf(X,Y,Z)
```

Gewöhnliche Differentialgleichungen

Gewöhnliche Differentialgleichungen (Tafel)

Löse im Intervall $I = [0, T]$:

$$u'(t) = f(t, u(t)),$$
$$u(0) = u_0.$$

Gewöhnliche Differentialgleichungen (Tafel)

Löse im Intervall $I = [0, T]$:

$$u'(t) = f(t, u(t)),$$
$$u(0) = u_0.$$

Explizites Eulerverfahren (Tafel)

Sei N die maximale Anzahl an Schritten. Dann ist $h = \frac{b-a}{N}$ Schrittgröße. Sei

$$t_0 = 0, y$$
$$t_k = t_0 + h \cdot k,$$
$$u_0 = u(0).$$

Iteriere:

$$u_{k+1} = u_k + h \cdot f(t_k, u_k)$$

Aufgabe

Löse im Intervall $I = [0, 5]$ mit $N \in \{5, 20, 50\}$:

$$u'(t) = \lambda u(t),$$
$$u(0) = 50.$$

Mit $\lambda = -2$.

Aufgabe I

Erstellt eine Funktion `rhs(t,u,lambda)`.

Aufgabe I

Erstellt eine Funktion `rhs(t,u,lambda)`.

Aufgabe II

Erstellt eine Funktion `[u]=ExpEuler(f,t0,T,u0,N)`.

Aufgabe I

Erstellt eine Funktion `rhs(t,u,lambda)`.

Aufgabe II

Erstellt eine Funktion `[u]=ExpEuler(f,t0,T,u0,N)`.

Aufgabe III

Löst mit allen Werten für N und plottet die Ergebnisse. Probiert den Befehl Subplot aus.