Übung 3: Einführung Matlab

Einführung

Was ist MATLAB **?**

- MATLAB (abkürz. für MATrix LABoratory) ist ein Software-Paket für numerische Berechnung und Visualisierung. Es besteht aus einer großen Anzahl an vorgefertigten Funktionen (z.B. lineare Algebra, Lösen von Differentialgleichungen, ...)
- Bei MATLAB steht der Fokus mehr auf numerischer Berechnung, während die Funktionalität von MAPLE und MATHEMATICA beispielsweise mehr auf symbolische Algebera ausgerichtet ist.
- Bekannte MATLAB-ähnliche Open-Source Programme: SCILAB, OCTAVE

Grund-Features:

- Automatische Dimensionierung (d.h., keine Dimensionsangaben bei Deklaration von Vektoren und Arrays erforderlich)
- Case-Sensitive (d.h., `a` und `A` sind unterschiedliche Variablen)
- Eingebaute Funktionen basieren auf Vektor- und Matrixoperationen und sind dementsprechend auch für diese optimiert.
- Datei-Typen:
  - M-Dateien: Standard ASCII Textdateien mit `.m` -Dateiendung
  - Mat-Dateien: Binärdateien, mit `.mat` -Dateiendung
  - Mex-Dateien ("MATLAB-Executable"): C, C++ oder Fortran Programme, welche von MATLAB aufgerufen werden können
- Grundlegendes Daten-Objekt ist das Array, welches aus Unterelementen verschiedenster Typen (Integern, Doubles, Matrizen, Characters, Strings, Strukturen und Zellen) bestehen kann.
- Der Typ einzelner Daten-Objekte muss nicht explizit deklariert werden. (z.B. besteht auch keine Notwendigkeit Variablen als Reel oder Komplex zu deklarieren)

Grundbefehle

Hilfe

`help <command>`	Hilfe zu `<command>`
`lookfor <string>`	listet Hilfe zu `<string>` auf

Verzeichnisbefehle

`pwd`	aktuelles Verzeichnis anzeigen
`cd`	Verzeichnis ändern
`dir` / `ls`	Inhalt des aktuellen Verzeichnisses anzeigen

Beispiel:

```
>> help cos
 COS    Cosine.
    COS(X) is the cosine of the elements of X.
 Overloaded methods
    help sym/cos.m
```

Variablen

Operatoren

`=`	einen Wert zuweisen
`+ - * / ^`	Rechenoperatoren
`,` (am Ende der Zeile)	Befehl mit Output
`;` (am Ende der Zeile)	kein Output
`...`	Zeilenumbruch innerhalb eines Befehls

Konstanten

`i,j`	Imaginärwert $\sqrt{-1}$
`pi`	$\pi = 3.14\ldots$
`inf`	Unendlich inf
`NaN`	not a Number, z. B. $\frac{0}{0}$
`eps`	kleinste positive Zahl, welche sich ausgeben lässt

Variablenmanagement

`who`	listet die Namen aller im Workspace befindlichen Variablen auf
`whos`	listet Namen und Größe der Variablen
`clear`	clear Workspace
`clear <var>`	clear Variable `<var>`
`clc` / `clf` / `cla`	clear command / figure / axes

Für mehr Informationen: Eingabe von `help +` in MATLAB-Kommandozeile.

Beispiele:

```
>> ( 47 + 1e+02 * 1.5 + 4^2 ) / 4
ans =
   53.2500
```

Die Variable `ans` ist das Ergebnis der letzten Rechenoperation.

```
>> a = ( 47 + 1e+02 * 1.5 + 4^2 ) / 4
a =
   53.2500
```

Das Ergebnis wird in Variable `a` gespeichert.

Mathematische Funktionen

`sqrt(x)`	Quadratwurzel
`xp(x)`	Exponentialfunktion
`log(x)` / `log10`	Logarithmusfunktionen
`sin(x)`	Sinus
`cos(x)`	Cosinus
`tan(x)`	Tangens
`atan(x)`	Arkustangens (Winkel $-90° \ldots +90°$)
`atan2(y,x)`	Arkustangens (Winkel $-180° \ldots +180°$)
`abs(x)`	Betrag von `x`
`sign(x)`	Signum (Vorzeichen von `x`)

Mehr Infos: Eingabe von `help elfun` oder `help datafun` in MATLAB-Kommandozeile.

Beispiel:

```
>> y1 = 2^2+log(pi)*sin(0.75*pi/2)+sqrt(exp(2*pi/3))

y1 =
    7.9072
```

$$y_1 = 2^2 + \ln \pi \sin(0.75\pi/2) + \sqrt{e^{2/3\pi}}$$

Vektoren und Matrizen

Vektor- und Matrixbefehle

`[ x1 x2 ...; y1,y2,...]`	Vektor oder Matrix (',' oder Leerzeichen zwischen Spalten, ';' oder Zeilenumbruch zwischen Reihen)
`start: <stepsize:> end`	Spaltenoperator (`stepsize` opt., sonst = 1)
`linspace(start,end,num_steps)`	linearer Zeilenvektor
`logspace(start,end,num_steps)`	logarithmischer Zeilenvektor
`eye(rows,columns)`	Einheitsmatrix [Zeilen x Spalten]
`ones(rows,columns)`	Matrix, bei der alle Einträge=1 sind [Zeilen x Spalten]
`zeros(rows,columns)`	Nullmatrix [Zeilen x Spalten]
`rand(rows,columns)`	Matrix mit Zufallswerten [Zeilen x Spalten]
`a(index)`	Element des Vektors `a` an Position `index`
`A(r,c)`	Element der Matrix `A` an Position `r` x `c`

Beispiele:

```
>> x=[1 2 3]
x =
     1     2     3
>> y=[4;5;6]
y =
     4
     5
     6
>> A=[1 2 3;4,5,6;7 8,9]
A =
     1     2     3
     4     5     6
     7     8     9
>> A(2,3)
ans =
     6
```

```
>> A(1,2)=8
A =
     1     8     3
     4     5     6
     7     8     9
>> B=A(2:3,1:3)
B =
     4     5     6
     7     8     9
>> v=0:2:8
v =
     0     2     4     6     8
>> v=[8:-2:0]
v =
     8     6     4     2     0
```

Operationen

`.* .\ ./ .^`	elementweise Berechnung
`\ /`	linke und rechte Division
`transpose(A)` oder `A.'`	Transponierte von `A`
`ctranspose(A)` or `A'`	Transponierte von `A` (komplex Konjugiert)
`inv(A)`	Inverse von `A`
`det(A)`	Determinante von `A`

Dimensionen

`[M,N] = SIZE(A)`	Dimension von Matrix und Vektor
`M = SIZE(A,DIM)`	Länge der Komponente `DIM` der Matrix `A`

Mathematische Funktionen von Vektoren und Matrizen

`sum(a)`	Summe der Vektorelemente
`prod(a)`	Produkt der Vektorelemente
`min(a)`	kleinstes Vektorelement
`max(a)`	größtes Vektorelement
`sort(a)`	Elemente in aufsteigender Reihenfolge
`find(a)`	nicht-Null Elemente

Beispiele:

```
>> A=[1 2 3; 4 5 6; 7 8 9];
>> B=[1 2 3; 2 4 5; 3 7 8];
>> b=[2 4 6 8 10]';
>> v=0:2:8;

>> v*b
ans =
   160
>> v'*b'
ans =
     0     0     0     0     0
     4     8    12    16    20
     8    16    24    32    40
    12    24    36    48    60
    16    32    48    64    80
```

```
>> c=v+b';
>> c=v-b';
>> C=A+B;
>> C=A-B;
>> C=A*B;
>> c=A*v(1:3)'
c =
    16
    34
    52
>> c=v.*b'
c =
     0     8    24    48    80
```

Beispiel:

```
>> A=[5 -3 2; -3 8 4; 2 4 -9];
>> b=[10;20;9];
>> x=A\b
x =
    3.4442
    3.1982
    1.1868
```

Lösen des linearen Gleichungssystems: $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$

$$\begin{aligned} 5x &= 3y - 2z + 10 \\ 8y + 4z &= 3x + 20 \\ 2x + 4y - 9z &= 9 \end{aligned}$$

Skripte und Funktionen

- Skript:
  Wenn ein Skript aufgerufen wird, führt MATLAB schlicht die in der Datei befindlichen Befehle aus. Skripte können im Workspace bereits vorhandene Daten bearbeiten oder auch neue Daten erzeugen. Auch wenn Skriptdateien keine Outputargumente aufweisen, werden alle erzeugten Variablen im Workspace gespeichert und können in darauffolgenden Rechenoperationen verwendet werden.

- Funktion:
  Funktionen sind `.m`-Dateien, welche Inputargumente aufnehmen und Outputargumente wiedergeben. Der Name der `.m`-Datei und der Funktion sollten übereinstimmen. Funktionen arbeiten mit lokalen Variablen in einem funktioneigenen Worskpace unabhängig des von der MATLAB-Kommandozeile bearbeiteten Workspaces.

Sowohl Skripte als auch Funktionen werden als .m-Datei gespeichert.

Struktur von Funktionen:

```
function [a_out,b_out] = name_of_function(a_in,b_in)
%         output list                     input list
%
% Description of function can be placed here
% by using the comment-operator '%'. This
% informations can be displyed by typing
% 'help name_of_function' at MATLAB-command-line.
.
.
a_out = a_in - a_out;
b_out = a_in + a_out;
.
.
```

Beispiel: Länge eines Vektors berechnen

```
function vlength = fvectorlength(vector)
%
% Computation of the length 'vlength' of
% a column vector 'vector'

vlength = sqrt(vector'*vector);
```

Aufrufen der Funktion:

```
>> fvectorlength([-1;3;5])
ans =
    5.9161
```

Logikoperatoren

`== , ~=`	gleich, nicht gleich		
`< , <=`	kleiner als, kleiner gleich		
`> , >=`	größer als, größer gleich		
`~`	logisches NOT	`&`	elementweise logiesches AND
`|`	elementweise logisches OR	`xor`	logisches EXCLUSIVE-OR

Mehr Infos: Eingabe von `help +` in MATLAB-Kommandozeile.

IF-/Fallunterscheidungen und Schleifen

`IF` expression (z.B. `a==1`)
  statements
`ELSEIF` expression
  statements
`ELSE`
  statements
`END`
	IF-Statement Bedingung
	
	ELSE und ELSEIF sind optional

`SWITCH` switch-expr (z.B. `id`)
  `CASE` case-expr, (z.B. `1`)
  statement, ..., statement
  `CASE {`case-expr1, case-expr2, case-expr3,...`}`
  statement, ..., statement
  `OTHERWISE,`
  statement, ..., statement
`END`
	SWITCH-Statement Fall

`FOR` variable = expr (z.B. `ii=1:10`) statement, ..., statement `END`	Wiederholung der Statements in einer spezifischen Anzahl.
`WHILE` expression (z.B. `ii<imax`) statements `END`	Wiederholung der Statements eine unbestimmte Anzahl bis Bedingnung erfüllt.

Plotten von Schaubildern

Bedienen des 'figure'-Fensters

`figure , figure(no)`	erstellen, aktivieren eines Schaubilds mit Nummer `no`
`subplot(num_rows,num_cols,index)`	erstellen eines Subplots z.B., `subplot(2,3,2)`: Subplot mit 2 Zeilen und 3 Spalten, aktivieren eines 2. Subplots
`gcf`	'get current figure'
`clf`	'clear active figure'
`delete(id)`	Objekte mit `id` löschen
`close(index)`	'figure'-Fenster Nr. `index` löschen
`close all`	alle 'figure'-Fenster löschen
`hold <on | off>`	aktuelles 'figure'-Fenster geöffnet lassen bei Generierung neuer Plots an/aus

2-D Plot Befehle

`plot(<x,>y<,plotstyle>,...)`	Plot, Linear (`<>` = opt.)
`fplot(func,range)`	Plot einer expliziten Funktion
`line(x,y)`	erstellen der Linie durch Koordinatenvektoren `x,y`
`text(x,y,string)`	platzieren von `string` an Position `x,y`

Für mehr Infos über 2-D Plots: aufrufen von `help plot` in Kommandozeile.

3-D Plot Befehle

`plot3(x,y,z <,plotstyle> ,...)`	dreidimensionaler Plot
`sphere`	plotten einer Kugel
`[x,y,z] = sphere`	Koordinaten einer Einheitskugel unter `x,y,z` speichern
`line(x,y,z)`	erstellen einer Linie durch Koordinatenvektoren `x,y,z`
`text(x,y,z,string)`	platzieren von `string` an Position `x,y,z`

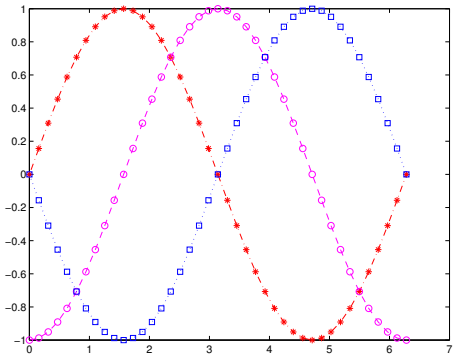
Plotstyles

Farben:			Linienstyles:		
`k` schwarz	`r` rot	`g` grün	`-` durchgängig	`o` Kreis	`.` Punkte
`b` blau	`m` magenta	`w` weiß	`--` gestrichelt	`*` sterne	`x` x
`c` cyan	`y` gelb		`:` gepunktet	`+` plut	`-.` Punkt-Strichlinie

Mehr Infos: `help plot`

Beispiele:

```
>> figure(1)
>> clf
>> t=0:pi/20:2*pi;
>> plot(t,sin(t),'-.r*')
>> hold on
>> plot(t,(sin(t-pi/2)),'linestyle','--',...
   'marker','o','color','m')
>> plot(t,sin(t-pi),':bs')
>> hold off
```

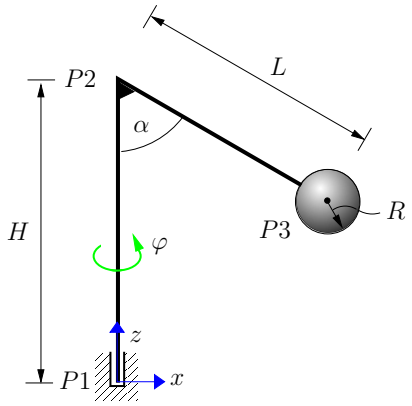


Labeln von Achsen und Figures

`axis([xmin,xmax,ymin,ymax<,zmin,zmax>)`	Skalierung (min=`-inf` oder max=`inf` → Auto Skalierung)
`axis <on | off | auto | equal | square>`	verschiedene Achsen-Befehle Mehr: `help axis`
`grid <on | off>`	Gitter an/aus
`gca`	Wiedergabe der aktuellen Achse
`cla`	löschen der aktuellen Achse
`xlabel(string)`	Label der x-Achse
`ylabel(string)`	Label der y-Achse
`zlabel(string)`	Label der z-Achse
`title(string)`	Titel der aktuellen Achse
`legend(string_1,string_2,...<,pos>)`	Legende platzieren `pos = 0,1,2,3,4,-1`

Aufgabe 3.1: Animation eines rotierenden Kragarms

Ein Kragarm der Länge L ist am oberen Ende (Punkt P2) an einer vertikalen Stange (höhe H) befestigt. Der Kragarm ist im Winkel α zur Stange geneigt. Die Stange ist am unteren Ende (Punkt P1) drehbar gelagert mit der z-Achse als Rotationsachse. Am Freien Ende des Kragarms (Punkt P3) ist eine Kugel mit dem Radius R befestigt.



Programmieren Sie ein MATLAB-Skript (`.m`-file), in welchem die Rotation φ um die z-Achse des dargestellten Systems in animierter Form dargestellt wird.

Hinweis: Nutzen sie die Rotationsmatrix

$$\boldsymbol{R} = \begin{bmatrix} \cos\varphi & -\sin\varphi & 0 \\ \sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

welche mit $\bar{\boldsymbol{v}} = \boldsymbol{R} \cdot \boldsymbol{v}$ die Rotation eines Vektors $\boldsymbol{v}$ um die z-Achse beschreibt.

Aufgabe 3.2: Animation mithilfe einer Subroutine

Das MATLAB-Skript aus der vorherigen Aufgabe soll nun modifiziert werden. Eine selbst programmierte Funktion soll die rotierten Koordinaten für die Animation berechnen. Input parameter sollen die die Ursprungskoordinaten und der Rotationswinkel φ sein. Der Output der Funktion sollten die Koordinaten des rotierten Systems sein.