MATLAB: Kapitel 1 – Erste Schritte

Einleitung

MATLAB ist ein Software-Programm, welches in erster Linie für numerische und ingenieurwissenschaftliche Anwendungen konzipiert wurde. Das System entstand ursprünglich aus dem Wunsch, für numerische Berechnungen mit Matrizen sehr genaue Resultate zu erzielen. MATLAB ist die Abkürzung für *MATrix LABoratory*. Seit her wurde das System auf viele neue Anwendungsgebiete erweitert. MATLAB hat sich auch zu einer Programmiersprache entwickelt.

Die Stärke von MATLAB liegt im effizienten und genauen numerischen Berechnen und im einfachen Programmieren. Im Gegensatz dazu liegt die Stärke von MAPLE im symbolischen Rechnen, d.h. im Aufsuchen von exakten und algebraischen Resultaten.

Die elementaren MATLAB-Operationen lassen sich grob in fünf Klassen einteilen:

- Arithmetische Operationen
- Logische Operationen
- Mathematische Funktionen
- Grafikfunktionen
- I / O-Operationen (Datenaustausch, Kommunikation)

Grundsätzlich handelt es sich dabei um *Operationen auf Matrizen und Vektoren*. Sogar eine Zahl ist eine einfache Matrix (1x1 - Matrix)

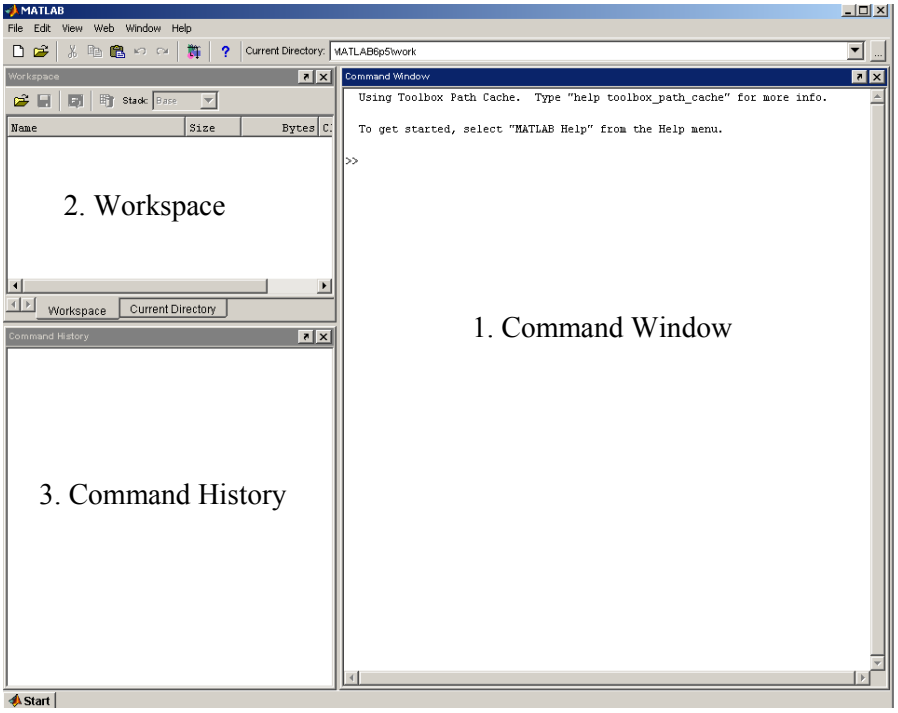
1.1 Die MATLAB-Oberfläche

Oberfläche

Starten Sie das Programm MATLAB. Über die Menüfolge

[*Desktop*] → [*Desktop Layout*] → [*Default*]

können Sie die Standart-Oberfläche einstellen.



Die wesentlichen Elemente sind:

1. *Command Window*: Eingabe von Befehlen,
2. *Workspace*: Auflistung der definierten Variablen
3. *Command History*: „Geschichte" der Eingaben (zur Rückverfolgung)

Der Benutzer gibt in MATLAB seine Kommando im *Command Window* nach dem Eingabepromt >> ein. Die Eingaben werden sofort (interaktiv) ausgeführt, ausser bei der Programmierung. Das untere linke Fenster *Command History* enthält die Geschichte der Eingaben.

1.2 Rechnen mit Zahlen (Skalare)

Skalare

Geben Sie im Kommandofenster die Rechenaufgabe 5 + 2 ein und schliessen Sie mit [Return] ab.

```
>> 5 + 2
```

Beobachten Sie die Reaktion in den verschiedenen Fenstern. Im *Kommandofenster* sehen wir das Resultat `ans = 7.`

Im *History-Fenster* erkennen wir die Eingabe und im *Workspace* die Variable `ans` (answer). MATLAB hat das Resultat der Variablen `ans` zugewiesen.

Eingaben korrigieren:
Um eine vorherige Eingabe zu korrigieren, muss die Eingabe nicht noch einmal eingegeben werden.

1. Variante: Mit der *CursorUp-Taste* ↑ können Sie früher eingegebene Befehle in die aktuelle Zeile holen und korrigieren. Mit den Tasten ↑ und ↓ bewegen Sie sich „virtuell" auf dem Kommandofenster.

2. Variante: Sie können im History-Fenster Kommandos markieren und mit dem Kontextmenü oder mit [Ctrl + C] usw. ins Kommandofenster kopieren.

Ausgaben unterdrücken:
Mit dem *Strichpunkt* können Sie Ausgaben unterdrücken. Das ist natürlich nur sinnvoll, wenn das Resultat einer Variablen zugewiesen wird. Beispiel: Tippen Sie folgende Zeile

```
>> 15 + 23;
```

Das Resultat wird nur in der Variablen ans gespeichert. Tippen Sie `>> ans.`

Aufgabe: Schreiben Sie `>> x = 0:0.05:2` einmal mit und einmal ohne Ausgabe. Interpretieren Sie!

Löschen der einzelnen Fenster (z.B. Command Window): Edit → *Clear*….

clear

Mathematische Funktionen:
Die wichtigsten mathematischen Funktionen sind vorprogrammiert:

`sin(..)`	Sinus	`asin(..)`	Arkus-Sinus
`cos(..)`	Kosinus	`acos(..)`	Arkus-Kosinus
`tan(..)`	Tangens	`atan(..)`	Arkus-Tangens
`sqrt(..)`	Quadratwurzel	`exp(..)`	natürliche Exponentialfunktion
`log(..)`	natürlicher Logarithmus	`log10(..)`	Zehner-Logarithmus
`abs(..)`	Betragsfunktion	`sign(..)`	Signumfunktion

Hinweis: Auch MATLAB rechnet bei den Winkelfunktionen immer in Bogenmass.

Alle elementaren Funktionen erhält man aufgelistet mit `>> help elfun`

arithmetische Operationen

Arithmetische Operationen:
MATLAB unterstützt die üblichen arithmetischen Operationen:

+	Addieren
-	Subtrahieren
*	Multiplizieren
^	Potenzieren

Division

Für die Division gibt es zwei verschiedene Operatoren:

/	Rechts-Division
\	Links-Division

Sind `a` und `b` zwei Skalare, so bedeutet `a/b` = $\frac{a}{b}$ und `a\b` = $\frac{b}{a}$.

Man erkennt, dass die Definition von zwei verschiedenen Divisionen bei Skalaren unnötig ist. Das macht erst dann Sinn, falls man mit Matrizen arbeitet, da in diesem Fall zwischen links und rechts zu unterscheiden ist.

Zahlenformate

Zahlenformat:
Die Ergebnisse werden im Allgemeinen mit 4 signifikanten Werten dargestellt. Beispiel

```
>> pi
ans =
      3.1416
```

Der im Rechner gespeicherte Wert ist jedoch viel exakter. Sie erhalten mehr Stellen mit der Anweisung

```
>> format long
>> pi
ans =
      3.14159265358979
```

Mit `>> format short` kehren Sie zur ursprünglichen Darstellung zurück.

Hilfe

Hilfe holen:
Sie möchten etwas über die *Sinusfunktion* in MatLab erfahren? Tippen Sie im Kommandofenster

```
>>help sin
```

Sie erhalten eine kurze Information. Wenn Sie genauere Informationen wollen, klicken Sie auf den Link doc sin oder auf das gelbe Fragezeichen in der Icon-Leiste. Vom Help-Fenster können Sie im Bedarfsfall Instruktionen ins Kommandofenster kopieren. Das Help-Fenster können Sie wie üblich mit einem Klick auf [×] schliessen. Alternativ gelangt man über die Menüfolge

[*Help*] → [*MATLAB Help*]

in die sehr umfangreiche Hilfsbibliothek. Dort können Sie nach der gewünschten Funktion suchen und Beispiel-Anwendungen kopieren.
Hinweis: Die Hilfsbibliothek muss bei der Installation von MATLAB nicht installiert werden (optional). Unter *www.mathworks.com* kann auf die komplette Bibliothek zugegriffen werden.

Musterbeispiele:

- Zur Berechnung von cos(10°):

```
>> cos(10/180*pi)
        0.98480775
```

- Zur Berechnung von $\sqrt{1+\ln(2)}$

```
>> sqrt(1 + log(2))
        1.3012
```

- Zur Berechnung von $\arctan\left(\frac{1}{3}\right)$

```
>> atan(1/3)
        0.3218
```

- um das Vorzeichen von $\ln(1+\sqrt{2}-\sqrt{5})$ zu erkennen

```
>> sign( log(1 + sqrt(2) - sqrt(5) ))
        -1
```

Aufgabe: Berechnen Sie im Command-Window:

a) $\sin(15°)$
b) $\sqrt[5]{1+\log_3(2)}$
c) $\ln(1+\sqrt{2}-\sqrt{5})$
d) $\arctan(\sqrt{3})$
e) $e^{\pi \cdot i}$

Konvertierung
Wenn Sie viele Rechnungen mit trigonometrischen Funktionen durchführen müssen, ist Ihnen vielleicht die „Umrechnerei“ ins Bogenmass verleidet. Sie können einen Umrechnungsfaktor definieren.

Beispiele:
Berechnen Sie sin(30°), cos(43°) und tan(–38°):

```
>> Grad = pi/180; % Konvertierungsfaktor
>> sin(30*Grad)
>> cos(43*Grad)
>> tan(-38*Grad)
```

Direkte Berechnung
Mit dem Zusatz „d“ rechnet Matlab direkt mit Grad-Winkelmass.

Beispiele:
Berechnen Sie sin(30°), cos(43°), tan(–38°) oder die inversen Funktionen wie $\sin^{-1}(0.5)$ usw.

```
>> sind(30)
>> cosd(43)
>> tand(-38)
>> asind(0.5)
```

1.3 Variablen

Variablen

Eine MATLAB-Variable ist ein Objekt eines bestimmten Datentyps: Reelle, komplexe Zahl, Matrix, String (Charakter-String), …

Mit folgender Zuweisung können Sie der Variablen x den Wert 2.65 zuweisen.

```
>> x = 2.65
```

Als Name für eine Variable müssen einige Punkte berücksichtigt werden:

- Gross- und Kleinbuchstaben werden unterschieden,
- jeder Name muss mit einem Buchstaben beginnen, gefolgt von Buchstaben oder Zahlen
- „_“ ist ebenfalls zugelassen
- MATLAB berücksichtigt nur die ersten 19 Zeichen eines Namens
- alle vordefinierten Funktionsnamen sind klein geschrieben.

Aufgabe: Fragen Sie den Wert von der Variablen x ab.

```
>> x
```

Aufgabe: Sie möchten wissen, welche Variablen Sie schon verwendet haben und wie viel Speicherplatz für jede Variable gebraucht wird.

who, whos

`>> who` : Liste der Variablennamen im *Workspace*
`>> whos` : Tabelle mit Variablennamen, Matrixgrösse und Speicherplatz

Aufgabe: Die definierten Variablen löschen

clear

`>> clear` Löscht den Arbeitsspeicher (Workspace). Gleiche Wirkung wie der Befehl `restart` bei MAPLE.

Aufgabe: Berechnen, aber überlegen Sie zuerst, was rauskommt

a)
```
>> 1+2+3*5;
>> ans*5 + 1
>> ans^(-1/2)
>> 3+1/ans
>> round(ans)
```

b)
```
>> format long
>> 1/1000
>> ans^4
>> 1/ans
>> log10(sqrt(ans))
```

c)
```
>> a = 20
>> b = 2
>> p = a*b
>> s = a + b
>> sq = (a + b)^2

>> a^2 + 2*a*b + b^2
>> ans - sq
```

d)
```
>> r = 5;
>> s = 3;
>> f = 0.1/r^s;
>> f
>> format short
```

1.4 Arbeiten mit M-Files

m-Files

Beim Eintippen der Befehle im *Command-Window* stösst man bald einmal auf Grenzen. Das *Command-Window* sollte man eigentlich nur dann einsetzen, wenn man kleinere Aufgaben durchzuführen hat, wie zum Beispiel einfache Rechenaufgaben. Das Hauptproblem ist, dass die eingegebenen Befehle nicht gespeichert werden können. Somit muss bei einem Neustart von MATLAB alles nochmals eingetippt werden! Zudem ist es schwierig umfangreiche Programme zu strukturieren und mit Kommentaren zu versehen.

Diese Probleme können mit einem MATLAB-Skript (sog. `M-file`) gelöst werden. Mit

[*File*] → [*New*] → [*M-file*]

definieren Sie ein neues MATLAB-Skript. Die Eingabe der Befehle und Variablen-Zuweisungen in einem `m-file` funktioniert genau gleich wie im *Command-Window*. Diese Skripte können nun gespeichert werden (Endung **.m**). Für die Wahl der File-Namen gelten dieselben Regeln, wie für die Wahl von Variablen-Namen (→ Kap 1.3)

Beispiel: Masse einer Kugel bestimmen (Filename: `MasseKugel.m`)

```
%----------------------------------------
%Masse einer Kugel bestimmen
%----------------------------------------
clear

%Eingaben
      %Durchmesser
      d=2;

      %Dichte
      rho=9500;

%Berechnung
      %Volumen
      V=4/3*pi*(d/2)^3;

      %Masse
      M=rho*V
```

Hinweis: Ein m-File kann unter [Debug] → [Run] oder direkt mit der Taste F5 ausgeführt werden.

Run→F5

Führen Sie dieses Programm aus und kontrollieren Sie den Workspace indem Sie im Command-Window `who` oder `whos` eingeben.

Aufgabe: Die Fakultät n! einer ganzen Zahl n, kann mit Hilfe der Stirling-Formel

$$n! \approx \sqrt{2 \cdot \pi \cdot n} \cdot \left(\frac{n}{e}\right)^n$$

angenähert werden. Schreiben Sie ein m-File `stirling_formel.m`, das nach Eingabe einer ganzen Zahl n die Fakultät nach der Stirling-Formel berechnet und den relativen Fehler in % zur exakten Fakultät ausgibt. *Hinweis:* n! lässt sich in MatLab mit `factorial(n)` exakt berechnen.

1.5 Matrizen und Vektoren

Der grundlegende Datentyp in MATLAB ist die Matrix. Diese kann aus reellen oder komplexen Zahlen oder aus Buchstaben und Zeichen (ASCII-Zeichen) bestehen.

1.5.1 Zeilenvektoren:

Die Vektor- und Matrixelemente werden bei der Eingabe in eckige Klammern gefasst. Die Elemente eines Zeilenvektors werden durch Leerzeichen oder Kommas getrennt.

Beispiel: Wir definieren einen Zeilenvektor und wollen die Vorzeichen der einzelnen Komponenten bestimmen.

```
>> a = [3 -5 0 7]

a =
     3 -5 0 7

>> sign(a)

ans =
     1 -1 0 1
```

Die Vorzeichen- Funktion sign wird auf jedes Element des Vektors einzeln angewandt.

Beispiel: Definiere eine Liste mit den ersten 6 Primzahlen. Multipliziere die dritte mit 8.

```
>> prim = [2 3 5 7 11 13]

prim =
     2 3 5 7 11 13

>> prim(3)*8

ans =
     40
```

Eine Vektorkomponente kann mit dem entsprechenden Index in runden Klammern ( ) angesprochen werden.

Häufig arbeitet man in MATLAB mit sehr grossen Matrizen und Vektoren, deren Koeffizienten nach einer algebraischen Regel gebildet werden. Die entsprechenden Eingaben können automatisiert werden. In vielen Anwendungen sind arithmetische Folgen wichtig. Man definiert eine Anfangskomponente *a1*, die Schrittweite *d* und das letzte Komponente *an*. Diese Folge definieren wir mit

```
>> a = a1:d:an
```

Bei grossen Vektoren sollte die Ausgabe mit einem Semikolon (;) unterdrückt werden.

Aufgabe: Geben Sie folgende Ausdrücke ein und interpretieren Sie die Ergebnisse

```
>> a =  2.0:0.4:4.4
>> b =  100:-5:10
>> c =  0:0.001:1000
>> d =  0:100
```

Hinweis: Bei Vektor c könnte das Programm eine Ewigkeit rechnen oder abstürzen, falls Sie versuchen alle Komponenten darzustellen. Eine laufende Berechnung kann mit **CTRL+C** unterbrochen werden.

Anwendung: Wir teilen das Intervall von 0° bis 90° Grad in 10°-Schritte und berechnen die entsprechenden Sinuswerte. Der Variablen *Winkel* ordnen wir die Winkel in 10-er Schritten zu

```
>> Winkel = 0:10:90;
>> Winkel = pi*Winkel/180;
>> sin(Winkel)
```

Die Strichpunkte am Schluss bedeuten, dass die Ausgabe unterdrückt wird.

1.5.2 Spaltenvektoren:
Beim Definieren von Spaltenvektoren müssen die Elemente mit einem Semikolon getrennt sein.

Beispiel: Spaltenvektor mit den ersten vier Primzahlen

```
>> a = [2; 3; 5; 7]
a =
      2
      3
      5
      7
```

Zeilen- und Spaltenvektoren können durch sogenannte *Transposition (transponieren)* ineinander überführt werden, der Vektor wird umgeklappt. Zeilen gehen in Spalten über und umgekehrt.

So zum Beispiel $\begin{pmatrix}6 & 8 & 10\end{pmatrix}^T = \begin{pmatrix}6\\8\\10\end{pmatrix}$ (T für transponiert) in MATLAB schreibt man ’.

Aufgabe: Verwandeln Sie den Spaltenvektor a in einen Zeilenvektor b:

```
>> b = a'
>> b
>> b'
```

1.5.3 Matrizen:

Matrizen sind nicht anderes als eine Komposition aus Zeilen- oder Spaltenvektoren. Das heisst somit, dass bei der Eingabe von Matrizen die einzelnen Zeilen mit Semikolons getrennt werden müssen.

Beispiel: Eingabe der Matrix

$$A = \begin{pmatrix}1 & 2 & 0\\5 & -3 & 4\\2 & 0 & 11\end{pmatrix}$$

```
>> A = [1 2 0;5 -3 4;2 0 11]
```

```
A =

     1     2     0
     5    -3     4
     2     0    11
```

Zugreifen und ändern des Eintrages in der 2. Zeile und 3. Spalte:

```
>> A(2,3)
ans =
     4

>> A(3,2)=100
A =
     1     2     0
     5    -3     4
     2   100    11
```

Zugreifen auf die 3. Spalte:

```
>> A(:,3)

ans =

     0
     4
     11
```

Zugreifen auf die 1. Zeile:

```
>> A(1,:)

ans =

     1     2     0
```

Der Doppelpunkt „ : “ bei den beiden letzten Beispielen bedeutet, dass man alle Zeilen resp. alle Spalten betrachtet.

1.5.4 Zusammensetzen von Vektoren und Matrizen:

Man kann schon vorhandene Matrizen wie Bauklötze zu neuen, grösseren Matrizen

zusammenfügen.

Beispiel:

```
>> a = [1 7 ; 2 3]

a =
      1 7
      2 3

>> b = [-3 5 ; 11 6]

b =
     -3 5
     11 6
```

• Wir wollen a und b zu einer (2 × 4) - Matrix zusammenfügen:

```
>> m = [a b]

m =
      1 7 -3 5
      2 3 11 6
```

• oder zu einer (4 × 2) – Matrix

```
>> m = [ a ; b ]

m =
      1 7
      2 3
     -3 5
     11 6
```

• oder zu einer (4 × 4) – Matrix

```
>> n = [ a b ; a' b']

n =
      1 7 -3 5
      2 3 11 6
      1 2 -3 11
                    7 3 5  6
```

1.5.5 Funktionen auf Vektoren und Matrizen

Die meisten MATLAB-Funktionen mit einem Argument können direkt elementweise auf Matrizen angewendet werden. Die Funktionsvorschrift wird dann auf jeden Koeffizienten angewandt.

Aufgabe: Beobachten und notieren Sie.

```
>> A = [1 0 -3; -2 -2 2]

>> B = [0 5 10; 3 -2 2]

>> C = [0 5; 10 3; -2 2]

>> A+B

>> A*C

>> 5*A

>> sign(A)

>> abs(A)

>> size(A)

>> w = [1 2 3 4 5 6 7 8 9]

>> sqrt(w)

>> exp(w)

>> C = [0 pi/4 pi/2 3*pi/4 pi]

>> sin(C)

>> u = [1 10 100 1000 10000]

>> log10(u)
```

Aufgabe: Im Folgenden sind noch einige wichtige Funktionen für Vektoren und Matrizen angegeben. Beobachten und notieren Sie.

```
>> data = [4.5 3.1 5.0 4.3]

>> max(data)

>> min(data)

>> sum(data)

>> mean(data)

>> size(data)

>> length(data)

>> sum(data)/length(data)
```

Einige wichtige Funktionen für Matrizen sind:

`inv(A)`	Matrixinverse (oder auch `A^(-1)`)
`det(A)`	Determinante
`EW = eig(A)`	Eigenwerte werden herausgegeben
`[EV,EW]= eig(A)`	Eigenvektoren als Spalten und die Eigenwerte in Dia-

	gonalmatrix. Die Eigenvektoren werden normiert, d.h. auf die Länge 1 gebracht.
`[EV,EW]= eig(A,'nobalance')`	Eigenvektoren als Spalten und die Eigenwerte in Diagonalmatrix. Die Eigenvektoren werden nicht normiert.
`poly(A)`	Charakteristisches Polynom von A. Es werden aber nur die Koeffizienten des Polynoms herausgegeben. Der höchste Koeffizient zuerst.

Beispiel: Wir berechnen die Eigenvektoren und Eigenwerte der Matrix $A = \begin{pmatrix} 1 & 3 \\ 0 & 2 \end{pmatrix}$ und überprüfen auf Richtigkeit. Dazu schreiben wir ein m-File `Eigen.m`

```
%----------------------------------------
%Eigenvektoren und Eigenwerte
%----------------------------------------
clear

%Matrix definieren
A=[1 3; 0 2];

%Eigenwerte & -vektoren
[EV,EW]=eig(A,'nobalance')   → Schauen Sie sich die Ausgabe im Com-
mand-Window an.

%Die Eigenvektoren sind die Spalten von EV
  x1=EV(:,1);   %1. Eigenvektor = 1. Spalte
  x2=EV(:,2);   %2. Eigenvektor = 2. Spalte

%Die Eigenwerte stehen in der Diagonalen der Matrix EW
  Lambda_1=EW(1,1);
  Lambda_2=EW(2,2);

%Test
  A*x1-Lambda_1*x1            → beide Ergebnisse sollten gleich Null
sein
  A*x2-Lambda_2*x2
```

1.5.6 Punktoperationen mit Vektoren und Matrizen:
Wir definieren einen Zeilenvektor `a = [1 2 4 8 12 20]` und möchten jedes Element quadrieren. Wir können nicht einfach

```
>> a*a
```

schreiben, da diese Operation aufgrund der Multiplikationsregel von Matrizen nicht definiert ist. Testen Sie das mit MATLAB aus.

Als Abhilfe wurde die sogenannte Punktoperation definiert.

*Wenn wir wollen, dass eine Operation mit jedem Element einer Matrix ausgeführt werden soll, müssen wir **vor dem Operationszeichen einen Punkt** setzen.*

Für unser Beispiel heisst das

```
>> a.^2 oder >> a.*a
```

Wenn wir vom Vektor a noch den Kehrwert bilden wollen, müssen wir ebenfalls mit dem Punkt- Operator arbeiten:

```
>> 1./a
```

Aufgabe: Testen Sie das aus.

Testataufgaben Kapitel 1:

Diese Aufgaben gehören zu den Testatbedingungen und sind bis Mittwoch, 20. April 2016 auf Ilias in den Ordner

>Briefkasten > Abgabe Matlab > Serie1

als hochzuladen. Packen Sie dazu alle in Übungen in ein File (m-File, pdf, oder .doc-File). Die Lösungen müssen Sie nicht ausgeben, nur die Eingaben und Befehle die zur Lösung führen würden.

Testate die zu spät hochgeladen werden, werden nicht berücksichtigt.

Übung 1.1: Gegeben sind einige Vektoren (Zahlenfolgen), die Sie auf möglichst einfache und kompakte Weise definieren sollen. Bei gewissen Aufgaben sind mehrere Schritte notwendig.

a) u1 = [0,8 0,9 1,0 …. 2,7]

b) u2 = [1 4 9 16 …. 10000]

c) u3 = [1000 998 996 … 4 2]

d) u4=[1 -2 3 -4 5 -6 7 … 99 -100]

e) u5=[1 1 2 3 5 8 13 21 34 55 89 ... 6765] „schwierig & freiwillig“

Übung 1.2: Bestimmen Sie die Summe aller Quadratzahlen von 1 bis 1000 (1, 4, 9, 16, … , 1'000'000). Versuchen Sie das in einer Zeile zu lösen.

Übung 1.3: Gegeben ist der Vektor a=[0 0.1 0.2 0.3 … 1]. Berechnen Sie punktweise

a) a^3 b) $\sin\left(\sqrt{\frac{1}{a}}\right)$ c) $\exp\left(a^2\right)$

Übung 1.4: Gegeben sind die zwei Matrizen

$$A=\begin{pmatrix}2 & -1 & 0\\ 5 & 1 & -3\\ 8 & 10 & 1\end{pmatrix} \text{ und } B=\begin{pmatrix}0 & -1 & 9\\ -2 & 1 & 5\\ 0 & 1 & 1\end{pmatrix}$$

a) Berechnen Sie $A \cdot B$ und $B \cdot A$

b) Bilden Sie die Matrizen $C=\begin{pmatrix}A & B\\ B & A^T\end{pmatrix}$ und $D=\begin{pmatrix}A & N & N\\ N & A & N\\ N & N & A\end{pmatrix}$ (mit $N=\begin{pmatrix}0 & 0 & 0\\ 0 & 0 & 0\\ 0 & 0 & 0\end{pmatrix}$)

Übung 1.5: Gegeben ist das Gleichungssystem

$$\begin{aligned} x_1 + 3x_2 + 4x_3 + 6x_4 &= 2 \\ -x_1 + 4x_2 + x_4 &= 1 \\ x_2 - 10x_3 + 8x_4 &= 8 \\ 5x_4 &= 1 \end{aligned}$$

mit den vier Unbekannten x_1, x_2, x_3 und x_4.

a) Schreiben Sie das Gleichungssystem in Matrizenform um.

b) Lösen Sie das System in MATLAB in einem m-File

Übung 1.6: Gegeben ist die 3x3-Matrix

$$A = \begin{pmatrix} -2 & 0 & 3 \\ 2 & 4 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

Schreiben Sie ein m-File, dass der Reihe nach die folgenden Aufgaben löst:

a) Definieren der Matrix A

b) Berechnung der Determinanten und der Inversen Matrix.

c) Bestimmung der Eigenwerte und Eigenvektoren. Definieren Sie die Eigenvektoren als `x1`, `x2` und `x3` und die Eigenwerte als `p1`, `p2` und `p3`

d) Weisen Sie nach, dass das Produkt der Eigenwerte gleich der Determinante ist.