v1.2

Matlab-Übungen zu Deterministische Signale und Systeme Musterlösung zur 1. Übung

1. Aufgabe

Abbildung 1 zeigt eine typische Ansicht der Matlab-Oberfläche. Die rechte Hälfte des Fensters wird durch die Kommandozeile belegt, in der Sie interaktiv arbeiten können. Links unten ist eine Liste der zuletzt benutzten Befehle zu finden. Weiter oben ist eine Auflistung der Dateien im aktuellen Verzeichnis zu finden, die Sie durch Klicken auf »Workspace« in eine Ansicht der im Moment von Matlab verwendeten Variablen umschalten können. Die Dateiliste stellt auch zahlreiche Funktionen eines Dateimanagers bereit und über die Variablenliste können Sie Variablen anzeigen lassen und manipulieren.

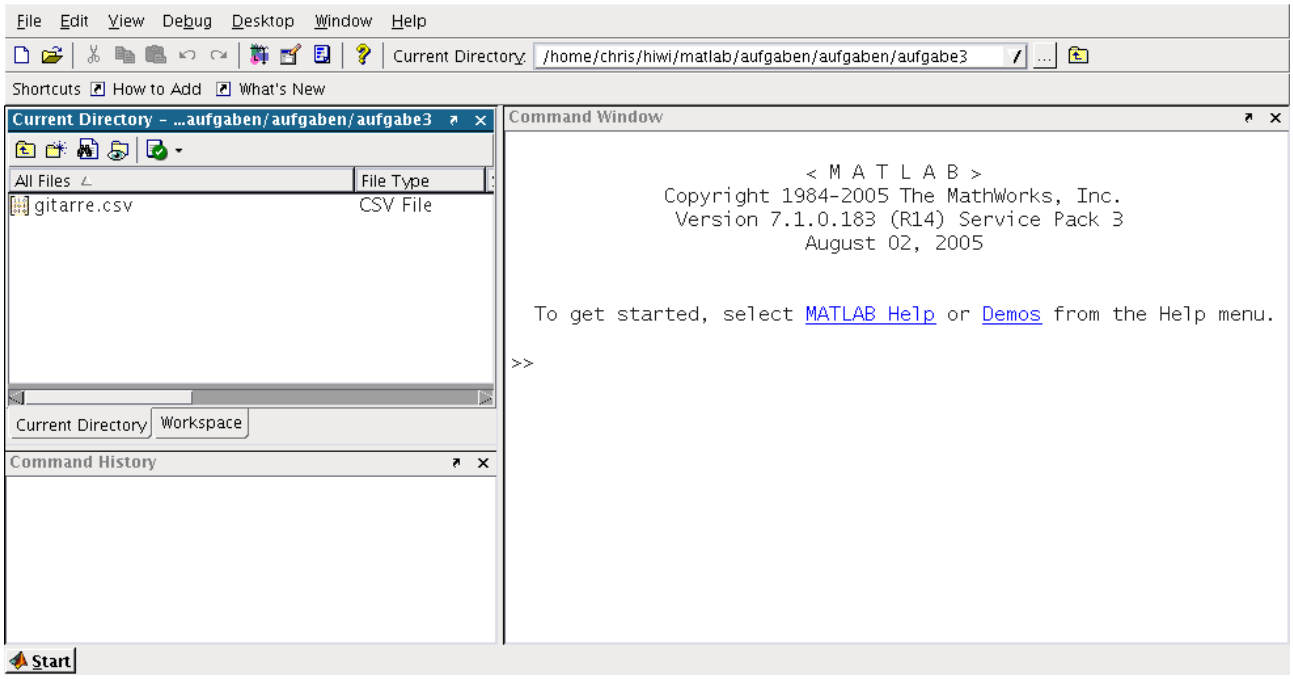


Abbildung 1: Die grafische Oberfläche von Matlab.

Dem »aktuellen Verzeichnis« oder »Arbeitsverzeichnis« (engl. working directory) kommt in Matlab eine besondere Bedeutung zu. Wenn Matlab Funktionen oder Skripte sucht (etwa weil

sie auf der Kommandozeile aufgerufen wurden – siehe dazu auch später in dieser Übung), schaut es zuallererst in diesem Verzeichnis, danach durchläuft es einen Suchpfad (mehr dazu in einer späteren Übung). Sie sollten immer darauf achten, daß das Arbeitsverzeichnis das Verzeichnis ist, in dem Ihre Daten und Matlab-Funktionen liegen, die Sie gerade verwenden (die »eingebauten« Funktionen findet Matlab natürlich auch so).

Sie können sich das Arbeitsverzeichnis auf der Kommandozeile anzeigen lassen, indem Sie **`pwd`** (»*p*rint *w*orking *d*irectory«) eingeben. Mit **`ls`** (»*l*i*s*t«) geben Sie den Inhalt des Arbeitsverzeichnisses auf der Kommandozeile aus. Wenn Sie das Arbeitsverzeichnis wechseln wollen, können Sie das Kommando **`cd <pfad>`** (»*c*hange *d*irectory«) verwenden, welches in den angegebenen Pfad wechselt. `<pfad>` kann dabei eine relative Angabe (etwa `verzeichnis/unterverzeichnis`) oder eine absolute Angabe (etwa `/verzeichnis/unterverzeichnis` – unter Windows bezieht sich diese absolute Angabe immer auf das aktuelle Laufwerk, `/Dokumente` entsprich also etwa `c:\Dokumente`) sein. Die Angabe `..` hat dabei eine Sonderbedeutung, sie wechselt eine Verzeichnisebene tiefer. Ist man etwa im Verzeichnis `/home/user/matlab-uebung`, wechselt **`cd ../..`** nach `/home`. Mit dieser Syntax folgt Matlab den Konventionen aus der UNIX-Welt, wer z. B. mit Linux arbeitet, kennt diese Befehle vielleicht schon.

Tipp: Lassen Sie sich auch unter Windows nicht von den Schrägstrichen als Pfadtrenner ins Bockshorn jagen, die können zwar auch DOS-typische Pfadangaben verwenden (und etwa den Befehl **`dir`** statt **`ls`**), in Matlab steckt jedoch einiges der UNIX-Philosophie und Sie tun sich später leichter, wenn Sie sich das gleich angewöhnen. (Matlab wurde erst relativ spät auch auf Windows portiert, daher diese UNIX-Affinität.)

Nun haben sie schon ein paar »Werkzeuge« kennengelernt und es kann mit den Aufgaben losgehen.

1.1. Matlab ist eine auf den Umgang mit Matrizen und Vektroren optimierte Programmiersprache und das ist auch einer der Haupt-Gründe, warum sich dieses Tool in so vielen Wissenschaftsdisziplinen als Standardwerkzeug etabliert hat.

Erzeugen Sie mehrere Variablen, die unterschiedliche Datentypen enthalten, nach dem folgenden Beispiel (verwenden Sie dazu den Zuweisungsoperator `"="`. Bei Zuweisungen steht immer links vom Operator der Variablenname und rechts der zugewiesene Wert):

```
a = 3
b = 5;
c = [1 2 a 4 b]
d = c.'
e = [a b; b a]
f = [b, a; 1, 2];
g = 'Hallo'
h = ' Welt!';
i = [g h]
j = 0:.1:1
% k = j
```

Sie können sich den Inhalt der entstandenen Variablen ansehen, indem Sie einfach den Variablennamen auf der Kommandozeile eingeben (und dann die Eingabetaste drücken).

Was enthalten die Variablen `a` bis `k`? Nennen Sie den Matlab-Datentyp und die mathematische Bezeichnung (falls möglich).

Unterscheidet Matlab zwischen Skalaren, Vektoren und Matrizen?

Finden Sie heraus, was die Funktionen **`who`** und **`whos`** machen und erklären Sie, wofür das Semikolon am Ende einer Zeile und das Prozentzeichen stehen.

1.2. Im letzten Aufgabenteil haben sie mehrere Vektoren und Matrizen in Variablen gespeichert. Finden Sie heraus, wie man Spaltenvektoren direkt anlegt (ohne Transponierung).

1.3. Speichern sie das aktuelle Arbeitsverzeichnis (also die Ausgabe von `pwd`) in der Variablen `pfad` und lassen Sie Sich auf der Kommandozeile sowohl `pfad` als auch `pfad.'` ausgeben. Was für ein Datentyp ist die Ausgabe von **`pwd`** und in welcher Form speichert Matlab diesen?

Lösung

1.1. Die Tabelle zeigt die Antworten zu den jeweiligen Datentypen. »double array« bedeutet in etwa »Matrix von Fließkommazahlen doppelter Genauigkeit« und »char array« »Matrix von Zeichen«.

Variable	Datentyp	Mathematische Bezeichnung
a	double array	Skalar
b	double array	Skalar
c	double array	Zeilenvektor
d	double array	Spaltenvektor
e	double array	2×2-Matrix
f	double array	2×2-Matrix
g	char array	—
h	char array	—
i	char array	—
j	double array	Zeilenvektor
k	Variable wird nicht angelegt	

Matlab unterscheidet bei der Speicherung von Werten nicht zwischen Skalaren, Vektoren und Matrizen. Skalare sind einfach 1×1-Matrizen, Vektoren sind $1 \times n$- oder $n \times 1$-Matrizen. Selbst Strings speichert Matlab in dieser Form, mit dem Unterschied, daß die Matrix nun Char(acter)-Werte, also einzelne Zeichen enthält. Dementsprechend kann man Strings auch aus einzelnen Zeichenketten zusammensetzen, als wären es Vektoren.

Die Funktionen **`who`** und **`whos`** listen beide die vorhandenen Variablen auf, mit dem Unterschied, daß **`whos`** genauere Angaben macht und **`who`** einfach nur deren Namen anzeigt.

Ein Semikolon unterdrückt die Ausgabe eines Ergebnisses (auch Variablenzuweisungen erzeugen also Ergebnisse resp. Rückgabewerte). Das Prozent-Zeichen leitet Kommentare ein.

1.2. Spaltenvektoren erzeugt man durch Verwenden des Semikolons als Zeilentrenner:

```
v = [1; 2; 3]
```

1.3. Der Rückgabewert von `path` ist ein String und folglich speichert Matlab ein »char array« in der Variablen `pfad`.

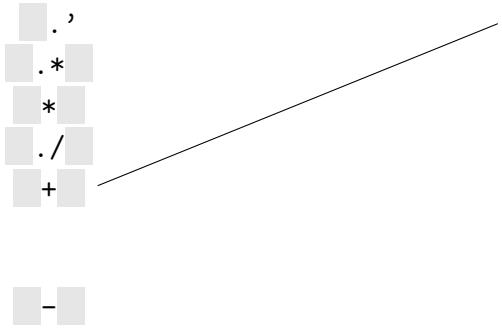
2. Aufgabe

Mit `clear all` können Sie den Workspace, also alle Variablen, die Matlab momentan verwendet, löschen. Nutzen Sie diesen Befehl zu Beginn jeder Aufgabe, um das System auf den »Ausgangszustand« zurückzusetzen.

Gegeben seien der Vektor $\vec{b} = (\ 1 \quad 2\ )$ und die Matrizen

$$A = \begin{pmatrix} 1 & 7 \\ 2 & 3 \end{pmatrix}, B = \begin{pmatrix} 5 & 3 \\ 4 & 2 \end{pmatrix} \text{ und } C = B^T.$$

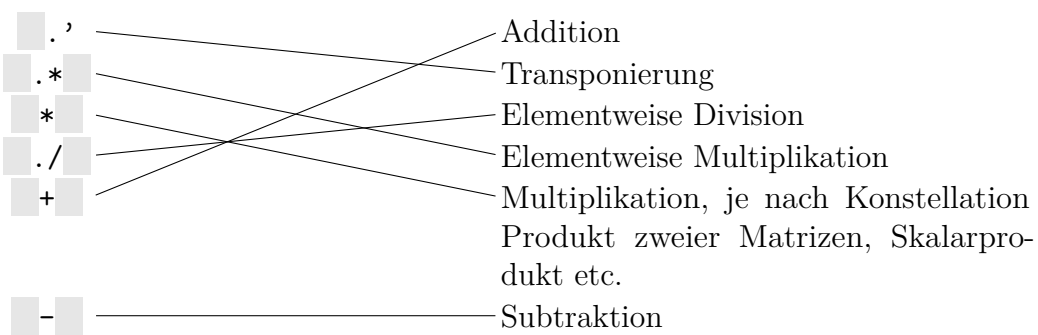
2.1. Ordnen Sie die Beschreibung den entsprechenden Rechenoperationen zu, indem Sie die oben angegebenen Vektoren und Matrizen in Variablen speichern und mit diesen dann ausprobieren, was die angegebenen Rechenoperationen bewirken (eine Zuordnung ist schon angegeben). Die grauen Kästchen dienen als Platzhalter für die einzusetzenden Variablen.



2.2. In Matlab sind die Operatoren \ und / definiert, die es, zumindest auf Vektoren und Matrizen angewendet, in der Mathematik so nicht gibt. Finden sie mit Hilfe der Matlab-Hilfe heraus, was diese Operatoren machen und beschreiben Sie es kurz.

Lösung

2.1. Die korrekte Zuordnung ist



2.2. Auf Skalare angewendet wirken die Operatoren \ und / wie eine gewöhnliche Division, wobei bei \ das rechte Element durch das linke Element geteilt wird. Beispiele (mitgeschriebene Matlab-Sitzung):

```
>> 1\2
ans =
     2
>> 1/2
ans =
    0.5000
```

Wendet man die Operatoren auf (quadratische) Matrizen an, wirken sie wie $\boldsymbol{A}\boldsymbol{C}^{-1}$ bzw. $\boldsymbol{A}^{-1}\boldsymbol{C}$ (werden intern aber anders berechnet):

```
>> A = [1 7; 2 3];
>> b = [1; 2];
>> C = [5 4; 3 2];
>> A/C
ans =
    9.5000   -15.5000
    2.5000    -3.5000
>> A*inv(C)
ans =
    9.5000   -15.5000
    2.5000    -3.5000
>> A\C
ans =
    0.5455    0.1818
    0.6364    0.5455
>> inv(A)*C
ans =
    0.5455    0.1818
    0.6364    0.5455
>> x = A\b
x =
     1
     0
>> A*x
ans =
     1
     2
```

Ist $\boldsymbol{A}$ eine $n \times n$-Matrix und $\vec{b}$ ein Spaltenvektor der Länge n, so gibt `A\b` die Lösung des Gleichungssystems $\boldsymbol{A}\vec{x} = \vec{b}$.

Für andere Eingabekonstellationen haben die beiden Operatoren weitere Bedeutungen, die aber für diese Übungen keine Bedeutung haben. Für eine Beschreibung sei der interessierte Leser auf die zahlreich vorhandene Literatur zu Matlab verwiesen.

3. Aufgabe

In dieser Aufgabe geht es um die Darstellung von mathematischen Funktionen, also dem Umgang mit etwas wie $f(x) = x^2$, nicht zu verwechseln mit Funktionen im Sinne von »Unterprogrammen« bei der Programmierung, welche wir später kennenlernen werden.

3.1. Speichern Sie Werte von -2 bis 2 im Abstand von 0,1 in einem Zeilenvektor `x1` und die Werte -2, 0 und 2 in einem weiteren Zeilenvektor `x2`. Speichern Sie nun in einem Zeilenvektor `y1` das Quadrat von `x1` und in `y2` das Quadrat von `x2`. Stellen Sie die Funktionen $y_1 = x_1^2$ bzw. $y_2 = x_2^2$ mit

```
plot(x1, y1, x2, y2)
```

dar. Um die richtige Zuordnung der entstandenen Graphen zu erleichtern, können Sie mit

```
legend('y_1=f(x_1)', 'y_2=f(x_2)')
```

eine Legende einblenden.

(3.1.1) Warum unterscheiden sich die beiden Darstellungen, obwohl beide die Funktion $f(x) = x^2$ darstellen?

(3.1.2) Welche Werte bekommt der `plot`-Befehl von Matlab übergeben?

(3.1.3) Wie arbeitet der Befehl `legend`?

(3.1.4) Schätzen Sie durch Ablesen aus dem Graphen ab, wie groß der Fehler gegenüber dem genauen Wert der Funktion $f(x) = x^2$ bei der Darstellung von $y_2 = f(x_2)$ bei $x = 1$ ist (Sie können die Rechengenauigkeit von Matlab in dieser Aufgabe vernachlässigen!). Der befehl `grid` könnte Ihnen dabei helfen.

3.2. Haben Sie eine Idee, wie Sie den tatsächlichen Fehler bei $x = 1$ bestimmen können? Wenn ja, setzen Sie diese Idee um und bestimmen Sie damit den Fehler.

3.3. Was passiert, wenn Sie beim Plotten einer einzelnen Funktion den Vektor für die x-Werte weglassen. Was passiert, wenn Sie beim Plotten beider Funktionen die x-Werte weglassen?

3.4. Mit den Befehlen `xlabel`, `ylabel` und `title` können Sie Diagramme in Matlab beschriften. Finden Sie mit Hilfe der Hilfe heraus, wie die Befehle aufgerufen werden und geben Sie dem Diagramm aus den vorangegangenen Aufgabenteilen eine sinnvolle Beschriftung.

3.5. Wenn Sie

```
x = [ 1  0 -1  0  1 ];
y = [ 0  1  0 -1  0 ];

figure;
plot(x, y);
```

in die Matlab-Eingabeaufforderung eingeben, erzeugt das ein auf einer Ecke stehendes Quadrat. Erzeugen Sie die Darstellung des Einheitskreises.

3.6. Der Umgang und die Darstellung von Signalen geschieht in Matlab auf die gleiche Weise wie bei Funktionen. Stellen Sie das Signal $y(t) = \sin(2 \cdot \pi \cdot f_0 t)$ für $0 \leq t \leq 1\,\mathrm{s}$ mit $f_0 = 5\,\mathrm{Hz}$ dar. Wählen Sie dazu zuerst eine geeignete Zeitachse.

Lösung

3.1. Mit dem Matlab-Code

```
x1 = -2 : .1 : 2;
x2 = [ -2 0 2 ];

y1 = x1 .^2;
y2 = x2 .^2;

plot (x1 ,y1, x2 ,y2 );
legend ('y_1=f(x_1)', 'y_2=f(x_2)');
grid;
```

erzeugt man die Darstellung der Funktion $f(x) = x^2$.

(3.1.1) Matlab benutzt zum Plotten in der Regel[1] Wertepaare (bei 2D-Plots) und Wertetripel (3D-Darstellungen). Zwischenwerte, die dabei angezeigt werden, werden aus den Wertepaaren linear interpoliert. Da bei der Darstellung von y_2 nur 3 Wertepaare vorhanden sind, stellt Matlab auch nur zwei Geraden dar.

(3.1.2) Der `plot`-Befehl bekommt zuerst einen Vektor für die Werte auf der x-Achse, dann einen Vektor für die Werte auf der y-Achse. Dabei ist natürlich sicherzustellen, daß beide Vektoren die gleiche Länge haben und beides Zeilen- bzw. Spaltenvektoren sind. Sind diese Voraussetzungen nicht erfüllt, kann Matlab keine Wertepaare zuordnen und gibt eine Fehlermeldung aus.

(3.1.3) Der Befehl `legend` bekommt in der Reihenfolge der geplotteten Kurven Strings übergeben, die im Plot als Legende angezeigt werden.

(3.1.4) Die Differenz beträgt ca. 1.

3.2. Man erkennt natürlich sofort, daß die Gerade zwischen $(0, 0)$ und $(2, 4)$ der Funktion $g(x) = 2 \cdot x$ entspricht, welche bei $x = 1$ den Wert 2 annimmt. Da y_1 eine Stützstelle bei $x = 1$ hat, enthält y_1 auch den genauen Wert 1^2 (also 1). Folglich ist die Differenz, also der Fehler, genau 1, was auch der folgende Matlab-Code zeigt:

```
x = 1;
e = 2*x - x^2
```

3.3. Vergißt man den Vektor mit den x-Werten, nimmt Matlab die Werte $1, 2, \ldots, N$ an, wobei N der Anzahl an Werten im y-Vektor entspricht. Vergißt man beide x-Vektoren, verwendet Matlab `y1` als zu `y2` gehörigen x-Wert. Sind beide gleich lang, ist das Resultat ein »falscher« Plot.

[1] bei manchen 3D-Plots gibt es Ausnahmen

3.4. Eine mögliche Beschriftung erzeugt der folgende Code:

```
figure;
plot (x1 ,y1 ,x2 ,y2 );
legend ('y_1=f(x_1)', 'y_2=f(x_2)');
xlabel ('x');
ylabel ('y');
title ('Diagramm␣einer␣quadratischen␣Funktion␣.');
```

3.5. Das einfachste Vorgehen ist wohl das Erzeugen eines parametrischen Plots. Dabei muß der Anwender selbst dafür sorgen, daß die parametrische Darstellung korrekt in kartesische Koordinaten umgesetzt wird, denn der Plot-Befehl von Matlab erwartet – wie schon besprochen – Wertepaare als Eingabe.

Ein geeigneter Parameter ist z. B. der Winkel ϕ und um eine gute Annäherungen an einen Kreis zu erreichen wählen wir eine Teilung im Abstand von 1°:

```
phi = 0 : 360;
```

Mit dem Parameter berechnet man die Punkte in kartesischen koordinaten:

```
x = cos(pi/180 * phi);
y = sin(pi/180 * phi);
```

Schließlich erfolgt der Plot:

```
figure;
plot (x, y);
```

3.6. Der Code

```
t1 = 0 : .25 : 1;
t2 = 0 : .01 : 1;

s1 = sin (2*pi * 5 * t1 );
s2 = sin (2*pi * 5 * t2 );

figure;
plot (t1, s1, '-v', t2, s2, '-o');
```

erzeugt zwei Plots des Signals $\sin(2\pi \cdot 5\,\mathrm{Hz} \cdot t)$, einmal mit 10 und einmal mit 100 Samples pro Sekunde. Bei der ersten Darstellung sieht man, daß pro Periode nur zwei Abtastwerte zur Verfügung stehen und diese auch noch so ungünstig liegen, daß eine Gerade auf der x-Achse entsteht. Mit der Zehnfachen Anzahl an Abtastwerten wird die Kurve schon einigermaßen glatt dargestellt.

4. Aufgabe

Matlab kann von Hause aus mit komplexen Zahlen umgehen. Um diese eingeben zu können, kennt Matlab die imaginäre Einheit i bzw. j.

Um die Lösung eines Polynoms zu bestimmen, kennt Matlab die Funktion `roots`, welche als Eingabe einen Vektor erwartet, der die Koeffizienten des Polynoms in absteigender Reihenfolge enthält. Die Gleichung $5x^3 + 7x^2 - 2x + 1 = 0$ läßt sich also mit

```
roots([ 5 7 -2 1 ])
```

numerisch lösen.

Die Aufgaben sind teilweise aus **mtut2** entnommen.

4.1. Berechnen Sie j^n mittels einer `for`-Schleife für $n = 1, 2, \ldots, 10$, $n = 12$ und $n = 17$. Welche Variable sollten Sie hier besser nicht für die Zählschleife verwenden?

4.2. Ermitteln Sie die Lösung(en) der folgenden Gleichungen.

(4.2.1) $z^4 - 1 = 0$

(4.2.2) $z^4 + 1 = 0$

(4.2.3) $z^5 = 32j$

Stellen Sie die Lösungen in einem Diagramm, das die komplexe Ebene darstellt, dar.

4.3. Gegeben sei der Vektor $\vec{a} = (\ 1 \quad 2j \quad 3 \quad 4j\ )$. Benennen Sie den Unterschied zwischen den Operationen `a'` und `a.'` in Matlab.

4.4. Berechnen Sie mit den komplexen Zahlen $u = 3 + 2j$ und $v = 4 - 5j$ die Werte von

(4.4.1) $u + v$

(4.4.2) $2u - 3v$

(4.4.3) $u + v^*$

(4.4.4) $u^* + v^*$

(4.4.5) $(u \cdot v)^*$

4.5. Rechnen Sie die Zahlen $Z_1 = 3{,}247 + 1{,}254j$ und $Z_2 = -3{,}247 - 1{,}254j$ in die Darstellung $Z_n = a_n \cdot e^{j\phi_n}$ um. Tipp: Verwenden Sie die Funktionen `abs`, `atan`, `real` und `imag`. Überprüfen Sie die berechneten Werte zeichnerisch (mit Papier, Bleistift und Geo-Dreieck). Sind die Ergebnisse korrekt? Wenn nicht, welches Ergebnis ist falsch und warum?

4.6. Bringen Sie die Zahl $Z_3 = -3{,}247 - 6{,}125j$ in die Exponentialdarstellung. Verwenden Sie diesmal statt `atan` die Funktion `atan2`.

4.7. Mit `a == b` können Sie überprüfen, ob zwei Werte den gleichen Inhalt haben. Falls ja, ist das Ergebnis 1, wenn nein, dann 0. (Mit Strings funktioniert diese Überprüfung nicht, sollten Sie das benötigen, können Sie die Funktion `strcmp` bemühen.) Es sei $Z_4 = \pi + je$, wobei e die Eulersche Zahl darstellt. Überprüfen Sie die folgenden Aussagen auf die beschriebene Weise.

(4.7.1) $Z_4 \cdot Z_4^* = |Z_4^2|$

(4.7.2) $Z_4 \cdot Z_4^* = |Z_4|^2$

(4.7.3) $e^{j\pi} = -1$

Entsprechen die Ergebnisse dem, was Sie erwartet haben? Wenn nein, versuchen Sie herauszubekommen, warum das Ergebnis von dem von Ihnen erwarteten Ergebnis abweicht.

Lösung

4.1. Hier wäre es ungünstig, i oder j als Zähler der Schleife zu verwenden, da das die imaginäre Einheit überschreiben würde. Sollte das an anderer Stelle doch mal nötig sein, so kann man sie jederzeit mit der Zeile

```
i = sqrt(-1);
```

wieder herstellen. In neueren Versionen empfiehlt Matlab `1i` für die imaginäre Einheit zu verwenden und zeigt Warnungen an den entsprechenden Stellen im Code, um Fehlern vorzubeugen.

Die Lösungen werden der Übersicht wegen in einer Matrix gespeichert (aus `Aufgabe4.m`):

```
loes = [];  % Matrix zum Speichern der Loesungen anlegen
for nn = [1:10 12 17]
    loes = [loes; nn j^nn]; % Loesungen berechnen
end
loes    % Loesungen ausgeben
```

4.2. Die Lösungen lassen sich leicht mit der `roots`-Funktion berechnen:

```
p1 = [1 0 0 0 -1];
l1 = roots(p1)

p2 = [1 0 0 0 1];
l2 = roots(p2)

p3 = [1 0 0 0 0 -32*j];
l3 = roots(p3)

plot(real(l1), imag(l1), 'o', ...
     real(l2), imag(l2), 'o', ...
     real(l3), imag(l3), 'o') %... bedeutet: Setze naechste Zeile fort!

grid;
legend('Loesungen von z^4-1=0', ...
       'Loesungen von z^4+1=0', ...
       'Loesungen von z^5=32')
```

Beim Darstellen haben wir uns zunutze gemacht, daß man dem Plot-Befehl im Prinzip x-y-Wertepaare übergibt und daß man das Zeichnen von Linien zwischen den Punkten unterbinden kann.

Will man nur eine der Lösungen darstellen, so akzeptiert Matlab auch die Form

```
figure;
plot(l3 , 'o');
grid;
legend('Loesungen␣␣von␣z␣^5=32');
```

und zerlegt die komplexen Zahlen selbständig in Real- und Imaginärteil. Für mehrere plots funktioniert das allerdings nicht mehr, da Matlab dann nicht mehr unterscheiden kann, welchem Plot was zugeordnet werden soll, man kann aber auch `hold on/off` verwenden.

4.3. Mit den Zeilen

```
a = [1 2*j 3 4*j];

a
a.' % transponierte
a'  % hermitesche
```

läßt sich überprüfen, wie die beiden Operatoren wirken. `a.'` bildet den transponierten Vektor $\vec{a}^T$, `a'` bildet den transponierten und gleichzeitig elementweise komplex konjugierten Vektor. Man nennt diesen auch *hermitesch* (nach Charles Hermite) und schreibt ihn daher meist $\vec{a}^H$.

4.4. Die Lösungen lassen sich leicht mit

```
u = 3 + 2*j;
v = 4 - 5*j;

disp('u␣+␣v␣=')
u + v
pause

disp('2u␣-␣3v␣=')
2*u - 3*v
pause

disp('u␣+␣v*␣=')
u + conj(v)
pause

disp('u*␣+␣v*␣=')
u' + conj(v)
pause

disp('(u␣+␣v)*␣=')
conj(u + v)
```

berechnen. Dabei wurden zum Bilden der komplex Konjugierten die Schreibweise ` ' ` und auch die Schreibweise `conj( )` angewendet, welche für Skalare ein identisches Ergebnis liefern.

4.5. Die Berechnung

```
Z1 = 3.247+1.254*j;
Z2 = -3.247-1.254*j;

a1 = abs(Z1)
phi1 = atan(imag(Z1)/real(Z1))

a2 = abs(Z2)
phi2 = atan(imag(Z2)/real(Z2))
```

liefert für Z_2 einen falschen Winkel, da der Arkustangens nicht zwischen a/b und $(-a)/(-b)$ unterscheiden kann.

4.6. Die Unzulänglichkeit des Arkustangens läßt sich mit der Funktion **atan2** beheben, welche allerdings zwei Parameter übergeben bekommt:

```
Z3 = -3.247 -6.125* j;

a3 = abs(Z3)
phi3 = atan2 ( imag (Z3), real (Z3 ))
```

Noch einfacher ist allerdings die Verwendung von **angle(Z)**, was direkt den Winkel zurückliefert.

4.7. Die Zeilen

```
Z4 = pi + exp(1)*j;

Z4 * conj(Z4) == abs(Z4^2)
Z4 * conj(Z4) == abs(Z4)^2
exp(j*pi) == -1
```

müßten dreimal eine 1 zurückliefern. Die letzte Zeile liefert allerdings eine 0. Die folgenden Zeilen zeigen, warum die Überprüfung scheitert:

```
>> Z = exp(j*pi)
Z =
  -1.0000 + 0.0000i
>> real(Z)
ans =
    -1
>> imag(Z)
ans =
   1.2246e-16
```

Hier liegt also ein Problem mit der Genauigkeit der Zahlendarstellung in Matlab vor.