WiMa-Praktikum 1

Universität Ulm, Sommersemester 2017

Woche 4

Lernziele

In diesem Praktikum sollen Sie üben und lernen:

- Einlesen von Dateien
- Schreiben von Dateien

Am Anfang geben wir Ihnen einen kurzen Überblick über verschiedene Aus- und Eingabefunktionen in MATLAB.

Beantworten Sie danach bitte erst einige Fragen, bevor Sie sich an den Rechner setzen!

Überblick

Häufig schreibt man Funktionen, die Daten benötigen oder verarbeiten, welche sich außerhalb des aktuellen Arbeitsspeichers von MATLAB befinden. Diese Dateien wurden möglicherweise von einem anderen MATLAB-Programm erzeugt oder gar von einer anderen Software (wie z. B. Excel) und können daher die unterschiedlichsten Formate besitzen. Andererseits müssen häufig am Ende des Programmablaufs oder auch während der Laufzeit des Programms erzeugte Daten in Dateien mit verschiedensten Formaten abgespeichert werden.
Zu diesem Zweck verfügt MATLAB über mehrere Aus- und Eingabefunktionen. Die Wahl der richtigen Funktion hängt von der Aufgabe und vom verlangten Format ab. In bestimmten Fällen kann man das gleiche Ergebnis mit verschiedenen Funktionen erreichen. Die folgende Tabelle soll einen Überblick über die wichtigsten Funktionen geben, die wir besprechen werden. An dieser Stelle sei erwähnt, dass MATLAB über viele andere Funktionen verfügt, die in dieser Einführung nicht behandelt werden. Darunter sind Funktionen zum Einlesen und Bearbeiten von Bildern und Videos. Interessierte Leser wenden sich bitte an die MATLAB Dokumentation.

`fopen, fclose`	**Files öffnen und schließen**
`fgetl, fprintf, fscanf`	**Lesen und Schreiben formatierter Files**
`sprintf, sscanf, textscan`	**Stringfunktionen**
`save, load`	**Lesen und Schreiben .mat/ASCII Files**

`save` und `load`

Beim Arbeiten in MATLAB kann man mit dem Befehl `save` die Variablen aus dem aktuellen Arbeitsspeicher in einer .mat Datei abspeichern. Optional kann auch ASCII Format verwendet werden (*American Standard Code for Information Interchange*). Mit dem Befehl `load` kann man den Inhalt einer Datei in den Arbeitsspeicher wieder laden. Die Funktionsweise wird am folgenden Beispielskript `ex04x01.m` erläutert.

```
1   % ex04x01.m
2   % Erzeuge Variablen und speichere diese in einer
3   % Datei ab
4   foo = 10;
5   bar = 5;
6   A   = zeros(foo, bar);
7   for i=1:min(foo, bar)
8       A(i,i) = 1;
9   end
10
11  whos; % Zeige Speicherinhalt
```

Der folgende Block beinhaltet die Ausführung des Skriptes und den Umgang mit `save` und `load`. Versuchen Sie die einzelnen Schritte nachzuvollziehen.

```
>> ex04x01
  Name        Size              Bytes  Class     Attributes

  A          10x5                400  double
  bar         1x1                  8  double
  foo         1x1                  8  double
  i           1x1                  8  double
>> save store.mat; % Speichere .mat Datei ab
>> clear all % loescht Arbeitsspeicher
>> whos; % Ausgabe wird leer sein
>>
>> load store.mat; % Lade Speicher
>> whos
  Name        Size              Bytes  Class     Attributes

  A          10x5                400  double
  bar         1x1                  8  double
  foo         1x1                  8  double
  i           1x1                  8  double
>>
>>
>> % Speichere nur Matrix A ab, verwende ASCII Format
>> % Wenn store.dat bereits existiert,
>> % so wird diese \"uberschrieben!
>> save A.dat A -ascii
>> clear % loescht Arbeitsspeicher
>> load A.dat
>> whos
  Name        Size              Bytes  Class     Attributes

  A          10x5                400  double
```

Stringfunktionen

MATLAB verfügt über Funktionen, die Informationen in einen *String* einsetzen oder Informationen aus einem String extrahieren können. Unter einem String verstehen wir eine Abfolge von

Characters. Grob gesagt handelt es sich bei einem Character um einen Buchstaben, eine Ziffer (0-9) oder ein Sonderzeichen. Genauer gesagt, ist ein Character ein elementarer Datentyp, der (i.d.R.) ein Byte groß ist, wobei dieses Byte in der 7 Bit langen ASCII-Kodierung oder der 8 Bit langen UTF-8-Kodierung (*Universal Character Set Transformation Format-8 Bit*) interpretiert wird.
Mit den Befehlen `sprintf`[1] und `sscanf` kann man *formatierte* Strings erzeugen und verarbeiten. *Formatiert* bedeutet in diesem Kontext, dass den Funktionen jeweils ein Format-String übergeben wird, der Formatierungs-Parameter enthält und damit den Aufbau des Strings beschreibt. Dadurch kann man z. B. Werte von numerischen Variablen in den String schreiben oder aus dem String extrahieren. Die meist verwendeten[2] Formatierungs-Parameter sind:

- `%c` steht für character;
- `%s` steht für string;
- `%d` steht für dezimal bzw. signed integer (Ganze Zahl);
- `%f` steht für floating-point number (Gleitkommazahl);
- die Angabe `%5.3f` bedeutet, dass das Feld für die Darstellung der Gleitkomma-Zahl mindestens 5 Stellen lang sein muss (falls nicht, füge Nullen bzw. Leerzeichen hinzu) und die Zahl mit genau 3 Nachkommastellen dargestellt werden muss.

Im folgenden Beispielskript `ex04x02.m` wird ein String mit Formatierungs-Parametern für ganze Zahlen erzeugt. Anschließend werden die Formatierungs-Parameter mit `sprintf` durch Variablen ersetzt.

```
1   % ex04x02.m
2   % Beispiel zu sprintf
3   woche = 4;
4   jahr  = 2017;
5   % %d bezeichnet das Format, signed int
6   % Variable satz ist ein String
7   satz  = 'Wir sind im besten WiMa Praktikum I aller
8           Zeiten im Jahr %d, Woche %d.';
9   % text ist auch ein String
10  text = sprintf(satz, jahr, woche);
11  disp(text)
```

Die Ausführung des Skriptes ergibt

[1] Funktioniert ähnlich wie die `printf` Funktion in C.
[2] Für eine ausführlichere Beschreibung siehe MATLAB Dokumentation.

```
>> ex04x02
Wir sind im besten WiMa Praktikum I aller Zeiten im Jahr 2017, Woche 4.
```

Man beachte hier, dass der String `satz` bereits die Format-Angaben enthält. Würde man den String `satz` direkt ausgeben lassen, so wäre die Ausgabe

```
>> satz

satz =
Wir sind im besten WiMa Praktikum I aller Zeiten im Jahr %d, Woche %d.
```

Nun wollen wir die Variablen `woche` und `jahr` löschen und den Wert dieser Variablen aus dem vorher erzeugten String `text` wiedergewinnen.

```
1   clear woche jahr
2   % String Scan. Lese formatierte Daten vom String.
3   % %*s bedeutet 'ueberspringe String'
4   A = sscanf(text,'%*s %*s %*s %*s %*s
5                    %*s %*s %*s %*s %*s %*s %d %*s %*s %d' );
6   satz = 'Ich sagte, wir sind im Jahr %d, Woche %d';
7   text = sprintf(satz, A(1), A(2));
8   disp(text)
```

Die Ausgabe ergibt:

```
Ich sagte, wir sind im Jahr 2017, Woche 4
```

Eine weitere nützliche Funktion ist `textscan`. Mit dieser kann man formatierte Daten aus einem String oder File auslesen. Dabei wendet der Befehl die Format Angaben **wiederholt** im gesamten String/File an, solange das Format mit den Daten übereinstimmt. Auf diese Weise können aus einem langen String oder einer Datei, die ein bestimmtes Muster hat, numerische Informationen extrahiert werden. Das Ergebnis wird in einem sogenannten *cell array* abgespeichert. Ein cell array ist ein Daten Container, bei dem jede Zelle (cell) mit { } adressiert wird und einen beliebigen Datentyp enthalten kann. Die Funktionsweise wird am folgenden Beispiel erläutert.

```
1   str = 'Kapitel 4, Seite 7, Zeile 20';
2   % Lese formatierte Daten vom String oder File
3   C   = textscan(str, '%s %d,');
4   celldisp(C)
```

Die Ausgabe ergibt:

```
C{1}{1} = Kapitel
C{1}{2} = Seite
C{1}{3} = Zeile
C{2} =
     4
     7
     20
```

`fopen` und `fclose`

Die Lese- und Schreibefunktionen für Dateien, die wir bisher kennengelernt haben, laden den gesamten Inhalt der Datei in den Arbeitsspeicher oder überschreiben den Inhalt mit neuen Daten. Manchmal möchte man aber interaktiv Daten in Files schreiben und/oder auslesen, ohne den gesamten Inhalt in den Arbeitsspeicher zu laden. Vor allem wenn die Dateien sehr groß sind, wird das Lesen und Schreiben des gesamten Files entweder sehr langsam oder erst gar nicht möglich, da die Größe des zulässigen Arbeitsspeichers überschritten wird. Außerdem kann man mit Hilfe dieser Funktionen Inhalt zu einer Datei hinzufügen, anstatt diese komplett zu überschreiben.
Zu diesem Zweck kann man in MATLAB mit File-IDs arbeiten. Das Konzept ähnelt sehr den `stream`-Klassen in C++. Zuerst muss ein File mit dem Befehl `fopen` 'geöffnet' werden. Intern verwaltet MATLAB ein Objekt der `stream`-Klasse. Mit dem Befehl `fopen` wird dieses Objekt erstellt und an die eingegebene Datei gebunden. D. h., wenn Lese- oder Schreibzugriffe für das `stream`-Objekt erfolgen, liest und schreibt das Objekt in oder aus der angebundenen Datei. Mit dem Befehl `fclose` wird die Datei 'geschlossen', d. h. die File-ID und die verwendeten Ressourcen (beispielsweise Puffer) werden frei gegeben, die neue Datei ist nun auf der Festplatte abgespeichert.
Die Funktionsweise ist vergleichbar mit der eines einfachen Texteditors: Mit `fopen` wird die Text Datei geöffnet; mit `fscanf` und `fprintf`[3] wird gelesen und geschrieben; mit `fclose` wird die Datei gespeichert und geschlossen. Des Weiteren können beim Öffnen einer Datei Zugriffsrechte spezifiziert werden, standardmäßig wird nur der Lesezugriff erteilt. Im folgenden Beispiel erzeugen wir einen Vektor, speichern diesen in einer Textdatei ab und fügen dann noch einen weiteren Vektor zur Datei hinzu. Mit dem Befehl `type` kann man den Inhalt einer Datei anzeigen lassen.

```
1  x       = rand(8,1);
2  % 'w' fuer write permissions
3  fileID  = fopen('num.txt', 'w');
4  fprintf(fileID, '%6.4f\n', x);
5  fclose(fileID);
6  type num.txt
```

Der Inhalt der Datei `num.txt` lautet:

[3] `fprintf` kann auch für die Ausgabe in der MATLAB Konsole benutzt werden. Der Unterschied liegt lediglich in der Bestimmung des Output Mediums: standardmäßig wird hierfür die Konsole verwendet. Falls man aber der Funktion ein File-ID übergibt, wird der Output in die zugeordnete Datei geschrieben.

0.3804
0.5678
0.0759
0.0540
0.5308
0.7792
0.9340
0.1299

```
1   y = rand(3,1);
2   % 'a' fuer append
3   fileID = fopen('num.txt', 'a');
4   fprintf(fileID, '%6.4f\n', y);
5   fclose(fileID);
6   type num.txt
```

Der Inhalt der Datei `num.txt` lautet nun:

0.3804
0.5678
0.0759
0.0540
0.5308
0.7792
0.9340
0.1299
0.1656
0.6020
0.2630

```
1   % 'r' fuer read permissions (default)
2   fileID = fopen('num.txt', 'r');
3   A = fscanf(fileID, '%f');
4   fclose(fileID);
5   disp(A)
```

Die Daten werden nun mittels `fscanf` eingelesen und in der Matrix `A` gespeichert. Die Ausgabe des letzten Befehls lautet demnach:

Wenn man nur eine bestimmte Zeile aus einer sehr großen Datei auslesen möchte, kann man dafür den Befehl `fgetl` verwenden. Im Folgenden Beispiel wird nur die vierte Zeile aus der vorher

```
0.3804
0.5678
0.0759
0.0540
0.5308
0.7792
0.9340
0.1299
0.1656
0.6020
0.2630
```

erzeugten Datei `num.txt` gelesen. Der Befehl `fgetl` liest die aktuelle Zeile aus und versetzt den internen Zähler auf die nächste Zeile.

```
1   fileID = fopen('num.txt', 'r');
2   tline = fgetl(fileID);
3   count = 1;
4   % While Schleife bis zum Ende der Datei
5   while ischar(tline)
6       % Falls die vierte Zeile erreicht wurde,
7       % gebe Zeile aus und breche Schleife ab
8       if count == 4
9           disp(tline)
10          break
11      end
12      tline = fgetl(fileID);
13      count = count + 1;
14  end
```

Die Ausgabe von Zeile 9 ist demnach:

```
0.0540
```

Offline Aktivitäten

Übereinstimmen

Schreiben Sie vor jeden Begriff auf der linken Seite den passenden Buchstaben der Beschreibung, die am besten mit der aus der rechten Spalte übereinstimmt.

_____	1.	`%d`	a.	Erzeugen von formatierten Strings.
_____	2.	`sprintf`	b.	Steht für character.
_____	3.	`load`	c.	Steht für dezimal bzw. signed integer
_____	4.	`fopen`	d.	Formatierte Daten aus einem File oder einem String lesen.
_____	5.	`%c`	e.	Inhalt einer Datei in den Arbeitsspeicher laden.
_____	6.	`textscan`	f.	Daten formatiert in eine Datei schreiben.
_____	7.	`fprintf`	g.	File-ID wird erstellt und an eine Datei gebunden

Ihre Antwort:

Fragen und Antworten

Beantworten Sie die folgenden Fragen.

8. Die Syntax zum Öffnen von Files ist laut Einleitung

   ```
   [fid,message] = fopen(filename, permission, machineformat).
   ```

   Wofür stehen die Attribute `permission`, `machineformat` und `message`? Sind diese Attribute optional?

   Ihre Antwort:

9. Der `save`-Befehl speichert die Variablen aus dem aktuellen Arbeitsspeicher in einer `.mat`-Datei. Besteht auch die Möglichkeit, ein ASCII-Format zu verwenden? Falls ja, wie sieht dann die Umsetzung aus?

   Ihre Antwort:

10. Sie haben die folgende Datei `test.dat` vorliegen:

09/12/2005	Level1	12.34	45	1.23e10	inf	NaN	Yes	5.1+3i
10/12/2005	Level2	23.54	65	9e10	-inf	0.01	No	2.2-5i
11/12/2005	Level3	34.80	12	2e5	10	100	No	3.1+.1i

Schreiben Sie ein Skript, welches diese Datei öffnet und die Spalten auswertet. Verwenden Sie einen dafür geeigneten Befehl, welchert in der Einleitung erwähnt wurde.

Ihre Antwort:

Programmausgaben

Für jedes der folgenden Programmsegmente, lesen Sie zuerst die Zeilen und schreiben Sie die Ausgabe an die dafür vorgesehene Stelle.

---

11. Wie lautet die Ausgabe des folgenden Skripts?

```
1   str = '0.41 8.24 3.57 6.24 9.27';
2   C = textscan(str, '%f');
3   celldisp(C)
```

Ihre Antwort:

12. Die ASCII-Datei `niederschlaege.dat` enthält zwei Spalten. In der ersten Spalte sind die Monate von 1 bis 12 notiert. In der zweiten Spalte ist der durchschnittliche Niederschlag in der Einheit Zentimeter notiert. Beschreiben Sie den folgenden Skriptverlauf und interpretieren Sie diesen:

```
1   load niederschlaege.dat;
2   monate = niederschlaege(:,1);
3   regen  = niederschlaege(:,2);
4   plot(monate, regen, 'o');
```

Ihre Antwort:

13. Erläutern Sie kurz, was als Ergebnis entsteht, wenn man `type exp.txt` abruft:

```
1 x=0:0.25:1;
2 A=[x;exp(x)];
3 fid=fopen('exp.txt','w');
4 fprintf(fid,'%6.2f %12.8f\n',A);
5 fclose(fid);
```

Praktikumsaufgabe

Funktionen der Form $f(x) = e^{-x^2}$ nehmen in der angewandten Mathematik oftmals eine zentrale Rolle ein. So ist beispielsweise die Verteilungsfunktion ϕ der Standardnormalverteilung, die in der Formulierung des zentralen Grenzwertsatzes auftaucht, gegeben durch:

$$\phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{1}{2}t^2} dt.$$

Aus der Analysis I ist bereits bekannt, dass bei der Auswertung von solchen Integralen numerische Verfahren notwendig sind. Als Beispiel eines solchen Verfahrens wollen wir an dieser Stelle stellvertretend die *Gauß-Quadratur* betrachten und das folgende Integral lösen:

$$\int_{-\infty}^{\infty} f(x)\, dx \quad \text{mit} \quad f(x) := e^{-x^2}. \tag{1}$$

Die Idee der Gauß-Quadratur besteht darin, die zu integrierende Funktion f aufzuspalten in $f(x) = g(x) \cdot \omega(x)$, wobei ω eine stetige, positive Gewichtsfunktion bezeichnet, also $\omega(x) > 0$ für alle x. Die Gauß-Quadratur ist nun von folgender Form:

$$\int_{-\infty}^{\infty} f(x)\, dx = \int_{-\infty}^{\infty} g(x) \cdot \omega(x)\, dx \approx \sum_{i=1}^{n} w_i \cdot g(x_i).$$

Die in der Summe auftauchenden Knoten x_i, $1 \leq i \leq n$, und Gewichte w_i, $1 \leq i \leq n$, müssen (mit Hilfe eines Eigenwertproblems) in Abhängig von der Gewichtsfunktion ebenfalls numerisch bestimmt werden. Um (1) lösen zu können, werden wir die Gewichtsfunktion

$$\omega(x) = e^{-x^2}$$

verwenden. Somit gilt in unserem Beispiel

$$f(x) = g(x) \cdot \omega(x) = 1 \cdot \omega(x).$$

Diese spezielle Form mit dieser Gewichtsfunktion der Gauß-Quadratur nennt man auch *Hermite-Gauß-Quadratur*.

---

14. (a) Laden Sie sich die Dateien `n2.dat` bis `n12.dat` im Moodle herunter. Diese enthalten in der ersten Spalte die Knoten und in der zweiten Spalte die Gewichte zur Hermite-Gauß-Quadratur für $n = 2, ..., 12$.

    (b) Schreiben Sie nun eine Funktion

    ```
    val = gaussquad(n),
    ```

    welche für gegebenes $2 \leq n \leq 12$ das Integral (1) näherungsweise mit Hilfe der Hermite-Gauß-Quadratur bestimmt. Verwenden Sie dafür die in Aufgabe (a) heruntergeladenen Dateien.