Wima 1 - Praktikum (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. Anschließend bearbeiten Sie bitte die Übungsaufgaben.

Überblick

Häufig schreibt man Funktionen, die Daten benötigen, welche sich außerhalb des aktuellen Arbeitsspeichers von MATLAB befinden. Diese Dateien wurden von einem anderen MATLAB Programm erzeugt oder gar von einer anderen Software (wie z.B. Excel) und können verschiedene Formate besitzen. Am Ende oder während der Laufzeit des Programms müssen häufig wiederum erzeugte Daten in Dateien verschiedener Formate abgespeichert werden.
Zu diesem Zweck verfügt MATLAB über mehrere Aus- und Eingabefunktionen. Die Wahl der richtigen Funktion hängt vom Zweck und 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 ein 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 Beispiel erläutert.

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

Name	Size	Bytes	Class	Attributes
A	10x5	400	double	
bar	1x1	8	double	
foo	1x1	8	double	
i	1x1	8	double	

```
1   save store.mat; % Speichere .mat Datei ab
2   clear all % Arbeitsspeicher frei
3   whos; % Ausgabe wird leer sein
```

≫

```
1   load store.mat; % Lade Speicher
2   whos;
```

Name	Size	Bytes	Class	Attributes
A	10x5	400	double	
bar	1x1	8	double	
foo	1x1	8	double	
i	1x1	8	double	

```
1   % Speichere nur Matrix A ab, verwende ASCII Format
2   % Wenn store.dat bereits existiert,
3   % so wird diese überschrieben!
4   save store.dat A -ascii
5   clear all % Arbeitsspeicher frei
6
7   load store.dat
8   whos;
```

Name	Size	Bytes	Class	Attributes
A	10x5	400	double	

Stringfunktionen

MATLAB verfügt über Funktionen, die Informationen aus einem *String* einlesen können. Unter

einem String verstehen wir eine Abfolge von *Characters*. Grob gesagt handelt es sich bei einem Character um einen Buchstaben, einer Zahl (0-9) oder einem Sonderzeichen. Genauer gesagt, ist ein Character ein elementarer Datentyp, das (i.d.R.) ein Byte groß ist, wobei dieser Byte in der 7-Bit langen ASCII Kodierung oder der 8-Bit langen UTF-8 (*Universal Character Set Transformation Format-8 Bit*) Kodierung interpretiert wird.
Mit den Befehlen `sprintf`[1] und `sscanf` kann man *formatierte* Strings erzeugen und einlesen. *"Formatierte"* bedeutet in diesem Kontext, dass den Funktionen Format Parameter übergeben werden, die den Aufbau des Strings beschreiben. Auf diese Weise kann man, z.B., numerische Variablen in den String schreiben oder aus dem String auslesen. Die am häufigsten verwendeten[2] Format Konventionen sind:

- `%c` steht für character;
- `%s` steht für string;
- `%d` steht für decimal bzw. signed int (Ganzzahl);
- `%f` steht für floating-point number (Gleitkommazahl);
- die Angabe `%5.3f` bedeutet, dass die Länge des Feldes für die Darstellung der Zahl mindestens 5 Stellen lang sein muss (falls nicht, füge Nullen/Leerzeichen hinzu) und die Zahl genau mit 3 Nachkommastellen dargestellt werden muss.

Im folgenden Beispiel wird ein String mit Platzhalter für Ganzzahlen erzeugt. Anschließend werden die Platzhalter mit `sprintf` durch Variablen ersetzt.

```
1  woche = 4;
2  jahr  = 2015;
3  % %d bezeichnet das Format, signed int
4  % Variable satz ist ein String
5  satz  = 'Wir sind im besten Wima Praktikum aller
6          Zeiten im Jahr %d, Woche %d';
7  % text ist auch ein String
8  text = sprintf(satz, jahr, woche);
9  disp(text)
```

```
Wir sind im besten Wima Praktikum aller Zeiten im Jahr 2015, Woche 4
```

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

```
≫ satz =
Wir sind im besten Wima Praktikum aller Zeiten im Jahr %d, Woche %d
```

Nun wollen wir die Variablen `woche` und `jahr` löschen und den Wert dieser Variablen aus dem

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

vorher erzeugten String `text` wiedergewinnen.

```
1   clear woche jahr
2   % String Scan. Lese formatierte Daten vom String.
3   % %*s bedeutet 'überspringe String'
4   A = sscanf(text,'%*s %*s %*s %*s %*s
5                    %*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)
```

```
Ich sagte, wir sind im Jahr 2015, 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, bis 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 sog. *cell array* abgespeichert. Ein cell array ist ein Daten Container, wo jede Zelle (cell) mit { } adressiert wird und 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)
```

```
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 wie 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 gar nicht möglich, da die Größe des zulässigen Arbeitsspeichers überschritten wird. Außerdem kann man mit Hilfe dieser Funktionen einen Inhalt zu einer Datei hinzufügen, statt diese 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 und aus der gebundenen Datei. Mit dem Befehl `fclose` wird die Datei 'geschlossen', d.h. die File ID und die verwendeten Ressourcen/-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 Textdatei 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, 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' für write permissions
3   fileID  = fopen('num.txt', 'w');
4   fprintf(fileID, '%6.4f\n', x);
5   fclose(fileID);
6   type num.txt
```

```
0.3804
0.5678
0.0759
0.0540
0.5308
0.7792
0.9340
0.1299
```

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

[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. Sollte man aber der Funktion ein File ID übergeben, wird der Output in die zugeordnete Datei geschrieben.

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

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

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

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 4.te Zeile aus der vorher 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 4.te 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
```

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.	`laod`	c.	Steht für decimal bzw. signed in.
_____	4.	`fopen`	d.	Formatierte Daten aus einem String oder File auslesen.
_____	5.	`%c`	e.	Inhalt einer Datei in den Arbeitsspeicher laden.
_____	6.	`textread`	f.	Objekt wird erstellt und an eine eingegebene Datei gebunden.

Antwort:

Fragen und Antworten

Beantworten Sie die folgenden Fragen.

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

Antwort:

8 Sie haben mit einem Text-Editor eine scan1.dat-Datei erstellt, welche die folgende Form besitzt:

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 die scan1.dat-Datei öffnet und die Spalten auswertet. Verwenden Sie einen dafür geeigneten Befehl, welcher in der Einleitung erwähnt wurde. Notieren Sie Ihre Antwort in die dafür vorgegebene Stelle.

Antwort:

Programmausgaben

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

1. Wie lautet die Ausgabe des folgenden Skriptes:

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

   Antwort:

2. Nehmen Sie an, sie besitzen eine ASCII-Datei mit dem Namen `PDXprecip.dat`, welche zwei Spalten enthält. In der ersten Spalte sind die Monate von Januar bis Februar notiert. In der zweiten Spalte ist der durchschnittliche Niederschlag in der Einheit "inch" notiert. Beschreiben Sie den folgenden Skriptverlauf und interpretieren Sie diesen:

```
1 load PDXprecip.dat;
2 month = PDXprecip(:,1);
3 precip = PDXprecip(:,2);
4 plot(month,precip,'o')
```

   Antwort:

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 s.g. *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. 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. Diese spezielle Form der Gauß-Quadratur nennt man auch die s.g *Hermite-Gauß-Quadratur*.

---

3. (a) Laden Sie sich von der Homepage der Veranstaltung die Dateien n2.dat bis n12.dat. Diese enthalten in der ersten Spalte die Knoten und in der zweiten Spalte die Gewichte zur Hermite-Gauß-Quadratur für $n = 2$, $n = 3$, usw..

   (b) Schreiben Sie nun eine Funktion

   ```
   val = gaußquad(n)
   ```

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