Matlabskripte und Funktionen

Stand: 22.8.2016

A) MATLABSKRIPTE:

Für aufwändige Berechnungen wäre es viel zu viel Arbeit, jedes Mal alle Arbeitsschritte einzeln in das Command Window zu tippen. Dafür gibt es die Möglichkeit, Skripte zu schreiben - Abfolgen von Befehlen, die in immer gleicher Weise ausgeführt werden. (Einige von Ihnen haben diese Möglichkeit ja gestern schon genutzt.) Bei einem Skript handelt es sich um die implementierte Version eines Algorithmus.

- Wenn man das Skript unter einem bestimmten Dateinamen mit der Endung `.m` abgespeichert hat (der Matlab-Editor ergänzt diese Endung automatisch), hat man damit einen neuen Matlab-Befehl erzeugt.
- Ruft man diesen Befehl auf, indem man ihn (ohne `.m`) in das Command Window eintippt, werden nacheinander alle Zeilen der Datei abgearbeitet. Erst wenn die gesamte Abarbeitung erfolgt ist, kann der nächste Befehl in das Command Window eingegeben werden.
- Natürlich kann der neue Befehl auch innerhalb eines anderen Skripts (oder einer Funktion) verwendet werden.

Zur Strukturierung größerer Programme dienen:

- Unterprogramme, bei denen Programmteile in eigene Dateien ausgelagert werden, die dann aus dem Hauptprogramm aufgerufen werden. Das ist insbesondere sinnvoll, wenn gewisse Teilaufgaben mehrfach erledigt werden müssen oder auch in einem anderen Zusammenhang interessant sein könnten.
- Innerhalb eines Programms können sogenannte "cells" - Programmteile - dazu dienen, den Ablauf weiter zu untergliedern. Diese lassen sich leicht im Programm wiederfinden und eventuell an anderer Stelle noch einmal verwenden.

Falls Sie das noch nicht getan haben, ist jetzt der richtige Zeitpunkt sich die Texte Wichtige Konzepte in Matlab und Wichtige Schritte beim Programmieren anzusehen, deren Inhalt hier nicht noch einmal wiederholt werden soll.

Damit Sie selber und auch andere Leute Ihre geschriebenen Matlabprogramme auch nach einiger Zeit noch nachvollziehen können, sollten Sie sich als "guten Stil" angewöhnen:

- der Datei einen aussagekräftigen Namen zu geben
- aussagekräftige Variablennamen zu verwenden
- großzügig zu kommentieren (am besten jede Zeile, die nicht total offensichtlich ist)
- das Programm mit einem Hilfetext zu versehen.
- Matlab kann nicht gut mit Umlauten, Leerzeichen und Rechenzeichen in

Dateinamen umgehen. Verwenden Sie deshalb nur Dateinamen (auch Verzeichnisnamen) nur die normalen Buchstaben des Alphabets (Groß- / Kleinschreibung wird unterschieden) und Unterstriche.

Der Matlab-Editor ist mehr als ein Fenster zum Eintippen und Speichern von Befehlen. Er beinhaltet eine Reihe von Werkzeugen, die einem das Programmieren und insbesondere das Finden von Fehlern in Programmen ("Debugging") erleichtern. Später im Kurs werden wir auch den Profiler verwenden. Dieses Hilfsmittel gibt Hinweise, wie man sein Programm Rechenzeit- und Speicherplatz-effizienter gestalten kann.

Eine Bitte: Bitte denken Sie daran, für ihre abgegebenen Lösungen der rot markierten Aufgaben eindeutige Dateinamen zu verwenden, die die jeweilige Aufgabennummer und Ihren eigenen Namen enthalten. Speichern Sie die Aufgaben dann im Cloudsystem in ihrem persönlichen Ordner.

Aufgaben:

T2A1) Falls Sie gestern auf Ihrem Laptop oder unter Ihrem Uni-Account auf den Uni-Rechnern noch kein Verzeichnis für die Daten aus diesem Kurs angelegt haben, tun sie das jetzt. Es bietet sich an, darin weitere Unterordner für die einzelnen Tage anzulegen. Achten Sie in Zukunft jedes Mal, wenn Sie Matlab neu starten, darauf, dass Sie sich im richtigen Verzeichnis befinden.

T2A2) Öffnen Sie den Matlab-Editor, indem Sie auf das Symbol für ein neues Dokument klicken. Und tippen Sie dort ein (bzw kopieren Sie):

```
clear all A1=ones(2,5); A2=2*ones(2,5);


Asum=A1+A2;
```

-
- Speichern Sie dieses Skript als *skriptchen.m* ab (unter `File->save as`). Achten Sie darauf, dass sie im richtigen Verzeichnis sind.
- Rufen Sie im Command Window das Programm auf indem Sie *skriptchen* eintippen.
- Was passiert im Workspace?
- Löschen Sie jetzt in jeder Zeile Ihres Skripts das Semikolon, speichern sie es ab und lassen es noch einmal laufen. Was ist der Unterschied?

T2A3) Ein Skript greift direkt auf die Variablen im Workspace zu und kann diese verändern. Es verhält sich also so, als ob man jeden Befehl einzeln im Command Window eingibt. Probieren sie aus, in das Command Window einzugeben:

```
clear all % loescht alle Variablen im Workspace  A1=[1 1; 2 2]
% definiert eine neue Variable  skriptchen    % ruft das oben
definierte Skript auf
```

Schauen Sie nach, welche Variablen jetzt im Workspace vorhanden sind. Welchen Wert hat die Variable A1?

T2A4) Es ist sehr wichtig, beim Programmieren Kommentare in das Programm zu schreiben, damit man auch nach ein paar Jahren noch weiss, was man sich dabei gedacht hat. Ergänzen Sie hinter jeder Programmzeile abgetrennt durch `%` als Kommentar, was gerade passiert, z.B.

```
clear all  % alle Variablen aus dem Workspace loeschen
```

T2A5) Außer der Kommentare zu den einzelnen Zeilen sollte jedes Programm einen Kommentar-Kopf haben. Schreiben Sie einen Hilfetext für `skriptchen.m`, indem Sie eine erste Zeile (mit `%` beginnend) ergänzen, die den Namen des Skripts nennt und seinen Inhalt erläutert, z.B.

```
% skriptchen.m ist ein Beispielskript, in dem zwei festgelegte
Matrizen addiert werden.
```

T2A6) Diese erste Zeile (bzw alle ersten Zeilen, die mit `%` beginnen) dienen als Hilfetext zum jeweiligen Skript: Tippen Sie in das Command Window `help skriptchen`.

T2A7) Bei unserem Skript geht alles zu schnell, um die Abarbeitung der einzelnen Schritte sehen zu können.

- Ergänzen Sie nach jeder Zeile des Skripts die Zeile `pause`- diese hält das Skript bei der Ausführung jeweils solange an, bis sie eine beliebige Taste drücken.
- Probieren Sie das aus und lassen Sie sich "unterwegs" ein paar Zeilen auf dem Bildschirm ausgeben. Dies können Sie mit dem Befehl `disp('hier steht Text')`.
- Der Befehl `pause` kann auch benutzt werden, um sich die Programmausführung sozusagen in automatischer Zeitlupe anzusehen. Wenn man ihm eine Zahl übergibt (z.B. mit `pause(2)`), wird die weitere Programmausführung für eine entsprechende Anzahl Sekunden unterbrochen.

T2A8) Statt jeweils `pause` einzutippen, kann man auch den Matlab-Editor für sich arbeiten lassen. Der Editor bietet die Möglichkeit, den sogenannten `cell mode` zu verwenden. Das bedeutet, dass man Programme aus Unterblöcken (cells) aufbaut.

- Einen solchen Block definiert man im Matlab-Editor durch eine Kommentarzeile, die mit `%%` anfängt. Diese sollte beschreiben, was im nächsten Unterabschnitt des Programms passiert.
- Der cell mode ist besonders hilfreich für große Programme, da sich anhand der Blöcke leicht die grobe Struktur des Gesamtprogramms nachvollziehen lässt und sich Blöcke gut finden und in andere Programme uebernehmen lassen.
- Wenn man "cells" benutzt, kann man diese auch einzeln laufen lassen. Dazu klickt man in den Programmcode oder die Kommentarzeile

eines "cells", dieses wird dann farbig markiert.
- Ganz oben links gibt es ein Symbol (zwei Kästchen, bei denen auf das obere ein Pfeil zeigt), das nur den betreffenden Programmteil ausführen lässt.
- Probieren Sie dies aus, indem Sie Ihr Skript in zwei Teile unterteilen, die jeweils mit einer `%%`-Kommentarzeile anfangen.

T2A9) Der Matlab-Editor kann auch zum "Debugging" benutzt werden, also um Fehler zu finden. Selbst wenn keine Fehler im Programm sind, bietet das Debugging die Möglichkeit, Matlab während der Arbeit "zuzusehen" und die Entwicklung von Variablenwerten Schritt für Schritt nachzuvollziehen. Probieren Sie diese Möglichkeiten aus:

- Wenn man oben in der Leiste des Editors auf das Symbol mit dem grünen Pfeil klickt, läuft das Skript los.
- Wenn nicht das ganze Programm durchlaufen soll, sondern man den Zustand des Workspace zwischendurch untersuchen will, kann man dafür sogenannte "Breakpoints" setzen, bei denen der Programmablauf unterbrochen wird. Das geht, indem man vor der Zeile, deren Effekt man sehen will, auf den kleinen Strich klickt. Dieser wird dann zum roten Punkt. Durch nochmaliges Klicken verschwindet er wieder.
- Probieren Sie aus, was passiert, wenn Sie einen Breakpoint setzen und das Skript laufen lassen. Testen Sie, was Sie während dieser Unterbrechung im Matlab Command Window tun können und was nicht. Was passiert, wenn man in diesem Zustand auf die verschiedenen Pfeilsymbole oben im Editor klickt? Wie kommt man insbesondere wieder aus dem Debugging-Modus heraus?
- Benutzen Sie statt des Befehls `pause` den Matlab-Editor, um sich die Bearbeitung jedes einzelnen Schritts des Programms anzusehen. Finden Sie heraus, wie das geht.

T2A10) Schreiben Sie ein Skript *`spaghettiskript`* zur Lösung der folgenden Aufgabe:

Gesucht ist die Gesamtlänge der Spaghetti auf Ihrem Mensateller, wenn diese in Längsrichtung geradegezogen aneinandergereiht werden. Sie wissen, dass die Spaghetti (ohne Soße) 250g schwer sind, in gekochtem Zustand einen Durchmesser von 2mm und ein spezifisches Gewicht von 1.05g/cm^3 haben. (Falls es Ihnen zu blöd ist, sich mit Spaghetti zu beschäftigen, stellen Sie sich vor, es handele sich um die Gesamtlänge der DNA in einem Zellkern, diese kann man im Prinzip entsprechend berechnen.)

- Überlegen Sie sich den Rechenweg (im Zweifelsfall mit Hilfe von Wikipedia).
- Setzen Sie diesen schrittweise in Zeilen Ihres Skripts um, verwenden Sie dabei aussagekräftige Variablennamen und achten Sie darauf, dass die Einheiten passen!
- Testen Sie Ihr Skript, indem Sie es laufen lassen.
- Falls das Skript nicht das erwartete Ergebnis liefert, können Sie die

Debugging-Funktionen des Editors verwenden, um den Fehler zu finden.

B) FUNKTIONEN

Häufig möchte man nicht, dass ein Skript beliebige Variablen des Workspace verändern kann, weil man diese weiterverwenden möchte. Deshalb gibt es Funktionen:

- Eine Funktion ist ein Programm, das während seiner Abarbeitung einen eigenen Workspace erzeugt, der vom Haupt-Workspace des Kommandofensters getrennt ist. Dieser Funktions-Workspace wird nach Abarbeitung aller Programmzeilen wieder gelöscht.
- Für die Kommunikation zwischen dem Workspace des Kommandofensters und dem Workspace der Funktion dienen Argumente:
- Eingabeargumente übergeben Variablen aus dem Workspace an die Funktion, so dass sie dort verwendet werden können.
- Ausgabeargumente übermitteln Ergebnisse der Funktion an den Workspace.
- Alle anderen innerhalb der Funktion verwendeten Variablen werden gelöscht, sobald die Funktion beendet ist.

Um eine Funktion zu erzeugen, muss in der ersten Zeile der Datei ("Kopf") das Schlüsselwort `function` stehen. Die Syntax fuer die Erzeugung einer Funktion ist die gleiche wie für den Aufruf einer Funktion, bis auf dieses Schlüsselwort:

- Erzeugung (erste Zeile in der Datei):
- *`function [Ausgabe1,Ausgabe2] = Funktionsname (Eingabe1, Eingabe2)`*
- Aufruf (im Command Window):
- *`[Ausgabe1,Ausgabe2] = Funktionsname (Eingabe1, Eingabe2)`*
- Hierbei sind *`Ausgabe1, Ausgabe2, Eingabe1`* und *`Eingabe2`* Variablennamen (sie könnten also auch `A`, `TTT` oder `Erna` heißen).
- Die Variablen können im Workspace der Funktion und im Workspace des Kommandofensters unterschiedlich heißen, aber die Werte der sich entsprechenden Parameter sind gleich.
- Wenn z.B. der Aufruf lautet: *`[a,b]=Funktionsname(27,M)`*
- Dann hat die Variable *`Eingabe1`* in der Funktion den Wert *`27`* und die Variable *`Eingabe2`* in der Funktion den gleichen Wert wie die Variable *`M`* im Kommandofenster.
- Die Funktion berechnet Werte für die Ausgabeargumente *`Ausgabe1`* und *`Ausgabe2`* und übergibt diese an die Variablen *`a`* und *`b`*, die nach Ende der Funktion im Workspace des Kommandofensters enthalten sind.

Guter Stil:

- Verwenden Sie immer den gleichen Namen für die Datei, den sie auch im Kopf der Funktion verwenden.

- Fangen Sie "unsinnige" Eingabeparameter durch Abfragen ab. (Wie das geht, besprechen wir morgen.)
- Überlegen Sie sich BEVOR Sie ein Programm schreiben, ob eine Funktion oder ein Skript die bessere Wahl ist und ob Sie bestimmte Teile des Programms in eine Unterfunktion oder ein Unterskript auslagern wollen.

Aufgaben:

T2B1) Beispiele für vordefinierte Funktionen mit Ein- und Ausgabeargumenten haben wir gestern schon gesehen, z.B. `y=sin(x)` oder `r=rem(a,b)`.

Es gibt auch Funktionen, die eine variable Anzahl von Ein- und Ausgabeargumenten haben können:

- Bereits gestern haben wir die Funktion *`size`* kennengelernt, die die Größe einer Matrix zurückgibt.
- *`size`* braucht mindestens eine Matrix als Eingabe (*`size(M)`*)
- zusätzlich kann die Dimension angegeben werden (*`size(M,d)`*).
- Abhängig von der Eingabe wird entweder die Anzahl der Zeilen und der Spalten der Matrix zurückgegeben (*`[zeilen,spalten]=size(M)`*) oder nur die Größe einer Dimension (*`z=size(M,1)`*).
- Probieren Sie aus, was size für die verschiedenen Arten des Aufrufs zurückliefert, wenn man es auf einen Skalar, einen Vektor und eine 3x3 Matrix anwendet.
- Für Vektoren gibt es den Befehle *`laenge=length(a)`*. Was liefert *`length(M)`* für eine Matrix *`M`*?

T2B2) Jetzt zu selbst geschriebenen Funktionen:

- Kopieren Sie den Inhalt von `spaghettiskript.m` mit dem Editor in eine neue Datei und ergänzen Sie *`function spaghettifunktion`* als erste Zeile, noch vor dem Hilfetext. Speichern Sie die Datei als `spaghettifunktion.m` ab (praktischerweise wird der Matlab-Editor Ihnen das sowieso als Standardoption anbieten).
- Lassen sie *`spaghettifunktion`* laufen. Was passiert?
- Löschen Sie alle Variablen des Workspace und lassen die Funktion wieder laufen. Was passiert?
- Vergleichen Sie das Verhalten der Funktion mit dem des Skripts. Ist es bei der Funktion auch so, als ob man die einzelnen Befehle im Command Window eingibt?
- Definieren Sie wie in T2B2) im Workspace des Kommandofensters eine Variable als *`[1 1; 2 2]`*, die den gleichen Namen hat wie eine Variable in Ihrer *`spaghettifunktion`*. Dann lassen Sie *`spaghettifunktion`* laufen. Was passiert? Welchen Wert hat die Variable im Workspace des Kommandofensters, wenn die Funktion durchgelaufen ist?
- Setzen Sie einen Breakpoint in der ersten Zeile ihrer Funktion und starten Sie die Funktion im Debugging Modus. Schauen Sie sich an,

wie sich der Workspace der Funktion von Schritt zu Schritt bei Abarbeitung des Programms verändert.

(Achtung: Wir werden die Funktion jetzt schrittweise verändern. Falls Sie alle Versionen aufheben wollen, müssen Sie jeweils einen neuen Funktionsnamen zum Speichern und Aufrufen verwenden, z.B. spaghettiskriptC2. Denken Sie daran, bei diesem und den nächsten Schritten jeweils den Hilfetext zu aktualisieren!)

T2B3) Häufig möchte man, dass eine Funktion ein Ergebnis liefert, das man auch weiter verwenden kann. Dies wird durch ein Ausgabeargument erreicht.

- Ergaenzen Sie in der ersten Zeile:
- *function laenge=spaghettifunktion*
- Löschen Sie alle Variablen des Workspace und rufen Sie *spaghettifunktion* auf. Was passiert?
- Vielleicht möchten Sie das Ergebnis Ihrer Funktion weiterverwenden. Weisen Sie dazu seinen Ausgabewert einer neuen Variablen zu, indem Sie z.B. *spaghettilaenge=spaghettifunktion* aufrufen.

T2B4) Bekanntlich schwankt die Spaghetti-Menge auf Mensatellern erheblich, je nach Persönlichkeit und Laune des Mensapersonals. Wir tragen dem Rechnung, indem wir die erste Zeile um ein Eingabeargument ergänzen: *function laenge=spaghettifunktion(gewicht)* und die Variable *gewicht* für Ihre Berechnung benutzen. (Wahlweise können Sie auch gerne, wenn Sie in Ihrer Funktion bereits eine andere Bezeichnung für das Gewicht gewählt haben, diesen Variablennamen in die Klammer der ersten Zeile schreiben.)

- Rufen sie zum Test die Funktion auf mit *laenge=spaghettifunktion(250)*. Entspricht das Ergebnis Ihrer Erwartung? Berechnen Sie die Längen für ein paar andere Gewichte.
- Was passiert, wenn man die `spaghettifunktion` jetzt ohne Ein- oder ohne Ausgabeargument aufruft?
- Versuchen Sie Ihre `spaghettifunktion` auszutricksen, indem Sie ihr "unpassende" Eingabewerte geben, z.B. eine Matrix. Was funktioniert? Wann gibt es Fehlermeldungen?

T2B5) Zu unserem Leidwesen schwankt nicht nur die Menge der Spaghetti auf dem Teller, sondern auch ihre Konsistenz. Wenn sie mal wieder völlig verkocht sind, haben sie ein spezifisches Gewicht, das nicht wesentlich von Wasser abweicht, also z.B. bei 1.01g/cm^3 liegt.

- Um das spezifische Gewicht im Programm mit einzubeziehen, brauchen wir mehrere Eingabeargumente.
- Ergänzen Sie das spezifische Gewicht als weiteres Eingabeargument und benutzen es in der Berechung. Die Liste der Eingabeargumente steht (sowohl im Programmtext als auch beim Aufruf) durch Komma getrennt in der Klammer hinter dem Funktionsnamen.
- Testen Sie Ihre neue Funktion.

T2B6) Um gegen die Variabilität der Spaghettiqualität zu demonstrieren, setzen Sie sich in den Kopf, eine Spaghettischlange von der Mensa Wechloy bis zur Hauptmensa (0.9km) zu legen.

- Ergänzen Sie deshalb als weiteres Ausgabeargument die Anzahl der Spaghettiportionen, die sie für diese Strecke brauche. Die Ausgabeargumente stehen in eckigen Klammern vor dem Gleichheitszeichen und dem Funktionsnamen.
- Testen Sie Ihre neue Funktion.
- *) Da man in der Mensa keine Bruchteile von Spaghettiportionen kaufen kann, runden Sie bitte die Angabe der Portionen auf die nächste größere ganze Zahl. Schauen Sie dazu in der Hilfe nach, welche Funktion Sie verwenden können.

T2B7) Probieren Sie aus, wie sich der Debugging-Modus des Editors verhält, wenn man ihn auf eine Funktion mit Ein- und Ausgabeparametern anwendet. Welche Unterschiede gibt es zur Benutzung bei einem Skript? Was muss man beachten?

C) SPEICHERN UND LADEN VON DATEN:

Im Normalfall bedient man Matlab nicht durch eintippen aller Daten, die man verarbeiten möchte, sondern benutzt Dateien, aus denen Daten eingelesen und in die Daten abgespeichert werden.

- Beim Abspeichern einer Datei mit dem Befehl `save Dateiname` legt Matlab standardmäßig seinen Workspace (also alle aktuellen Variablen) auf dem Datenträger ab.
- Die dabei automatisch erzeugte Endung des Dateinamens ist `.mat`
- Wenn man nur einen Teil der aktuellen Variablen abspeichern möchte, muss man dies gezielt mit angeben (s. T2C3).

Beim Einlesen einer .mat Datei mit dem Befehl `load Dateiname` erzeugt Matlab die darin abgespeicherten Variablen in seinem Workspace.

- Achtung: Die eingelesenen Variablen haben die gleichen Namen, die sie vor dem Abspeichern hatten (und nicht etwa den Dateinamen). Wenn Variablen des gleichen Namens im Workspace vorhanden sind, werden diese überschrieben!
- Es ist egal, ob man beim Einlesen nur `Dateiname` oder `Dateiname.mat` angibt

Achtung: Matlab benutzt zum Speichern und Laden jeweils das oben in der Leiste `Current Directory` angegebene aktuelle Verzeichnis.

- Das Verzeichnis kann man wechseln, indem man sich im `Current Folder` Fenster durch Klicken an die richtige Stelle im Verzeichnisbaum bewegt.
- Eine häufig schnellere Möglichkeit ist es, den Pfad über die Kommandozeile anzugeben.
- Der Befehl für einen Wechsel des Verzeichnisses lautet *cd*. Z.B. bedeutet `cd Tag1` "wechsele in den Unterordner Tag1".

- Um ein Verzeichnis nach oben zu wechseln, benutzt man die Bezeichnung .. so kann man mit dem Befehl `cd ../Tag2` z.B. von einem Unterordner `Tag1` direkt in einen anderen Unterordner `Tag2` wechseln.
- Die direkte Angabe des Verzeichnisses ist insbesondere dann praktisch, wenn man eine Datei in einem anderen als dem gerade aktuellen laden oder speichern möchte. Z.B. bedeutet `save Tag1/testdatei` dass der aktuelle Workspace im Unterverzeichnis `Tag1` unter dem Namen `testdatei.mat` abgespeichert wird.

Matlab-Abbildungen werden auf etwas andere Weise abgespeichert und erneut in Matlab geöffnet als Daten.

- Die einfachste Möglichkeit zum Speichern von Abbildungen ist, den Menüpunkt `>File >Save as` zu benutzen.
- Der Standard ist dann, dass die Abbildung mit der Endung `.fig` abgespeichert wird - dabei handelt es sich um ein Format, was nur von Matlab vernünftig verarbeitet werden kann, dort aber beim erneuten Einladen alle Eigenschaften der Abbildung beibehält.
- Wenn man die Abbildungen mit einem anderen Programm benutzen will, z.B. um sie in ein mit Word geschriebenes Protokoll einzubinden, muss man ein anderes Dateiformat benutzen als `.fig`. Dazu kann man wie oben beschrieben abspeichern, muss jedoch das gewünschte Dateiformat, z.B. `jpg` oder `pdf` auswählen und die Endung der Datei entsprechend anpassen.

Es können auch Dateien eingelesen werden, die nicht von Matlab erzeugt wurden (z.B. Text-, Bild-, Excell-Dateien etc). Die wichtigsten Befehle dafür werden heute vorkommen, weitere Möglichkeiten kann man in der Hilfe finden.

Aufgaben:

T2C1) Erzeugen Sie eine Variable `A=[1 2 3; 4 5 6]`. Speichern Sie diese mit `save MatrixA` ab.

- Schauen Sie im Fenster "Current Folder" nach, ob dort die neue Datei auftaucht.
- Um sich zu überzeugen, dass die Matrix wirklich gespeichert ist, löschen Sie sie aus dem Workspace mit `clear A`.
- Laden Sie die Matrix wieder ein mit `load MatrixA`. Überprüfen Sie, dass die Matrix `A` wieder im Workspace vorhanden ist.

T2C2) Definieren Sie zusätzlich eine Variable Matrix `B=[0 0 1; 1 0 0]` und berechnen Sie `C=A+B` und `D=A.*B`.

- Speichern sie den gesamten Workspace mit `save AufgabeT2C2`.
- Löschen Sie alle Variabelen mit `clear all` aus dem Workspace.
- Laden Sie `AufgabeT2C2` wieder ein und überprüfen Sie, dass alle Variablen wieder da sind.

T2C3) Häufig will man nicht alle Variablen speichern, die gerade zufällig im Workspace sind sondern nur eine Auswahl.

- Probieren Sie folgende Abfolge aus:
  - `save MatrixCD C D`
  - `clear all`
  - `load MatrixCD`
- Was ist jetzt im Workspace vorhanden?
- Und was, wenn Sie noch einmal `load MatrixA` dazutippen?

T2C4) Statt `load FILENAME` einzutippen, kann man sich auch im Fenster links das Current Folder anzeigen lassen und die gewünschte Datei per Doppelklick auswählen. Probieren Sie das mit `AufgabeT2C2.mat` aus.

T2C5) Achtung! Bei Einladen von Variablen werden die Werte von Variablen gleichen Namens überschrieben. Definieren Sie ausgehend vom aktuellen Workspace eine neue Variable `A=1`und laden Sie dann `MatrixA.mat` ein. Was passiert jeweils im Workspace?

T2C6) Kopieren Sie folgende Datei in Ihr Verzeichnis und laden Sie sie in Matlab ein: grosse_Matrix.mat.

- Wie viele Zeilen und Spalten hat diese Matrix?
- Schauen Sie sich die Matrix mit dem Array Editor an.
- Greifen Sie die 4. Spalte und die 10. Zeile jeweils als Vektor heraus.

T2C7) Sehen Sie sich `MatrixA.mat` in einem beliebigen Texteditor an.

- Was sehen Sie?
- Falls Sie Werte aus Matlab abspeichern und mit einem anderen Programm weiterverwenden wollen, speichern Sie das file besser mit der Option `-ascii` ab. Z.B.:
  - `clear;`
  - `E=[89 76 0.1 9.7];`
  - `save -ascii`
  - `MatrixE.txt`
- Sehen Sie sich das Resultat im Texteditor und in Matlab an.
- Probieren Sie aus, was passiert, wenn man mehrere Variable in ein Textfile schreibt.

T2C8) Erzeugen Sie sich in ihrem persönlichen Matlab-Verzeichnis ein Unterverzeichnis. Probieren Sie zwei Wege aus, eine Datei in dieses Unterverzeichnis zu schreiben:

- zunächst wechseln Sie in das Verzeichnis und speichern dort ab.
- Dann wechseln Sie wieder in das ursprüngliche Verzeichnis eine Ebene höher und speichern mit `save unterordner/dateiname2` (benutzen Sie dabei einen anderen Dateinamen als beim ersten Speichern, sonst wird die Datei überschrieben).
- Versuchen Sie beide Files sowohl vom Hauptverzeichnis als auch vom Unterverzeichnis aus einzuladen.

T2C9) Erzeugen Sie eine Abbildung mit zwei dargestellten Graphen, z.B. zwei Parabeln wie am ersten Kurstag. Speichern Sie die Abbildung im Standardformat `.fig` ab, indem Sie den Menüpunkt `>File >Save as`

benutzen.

- *) Zoomen Sie in die Abbildung hinein und speichern Sie die Abbildung unter einem anderen Namen wieder ab.
- Schließen Sie dann das Grafikfenster und laden die beiden gerade gespeicherten Abbildungen über `>File >open` im Hauptmenü ein.
- Probieren Sie, bei der Abbildung des Ausschnitts wieder den gesamten Graphen zu reproduzieren.

T2C10) Probieren Sie aus, Ihre in der letzten Aufgabe erzeugte Abbildung als jpg-Datei zu speichern, zu schließen und wieder mit Matlab zu öffnen.

- *) Was muss man tun, um sie sich wieder anzusehen?
- *) Kann man aus einem Ausschnitt der Kurve wieder zur gesamten Kurve zurückkommen?
- *) Probieren Sie aus, sowohl eine Abbildung im .fig Format als auch im .jpg Format in Word zu importieren.

T2C11) Da man Grafiken meistens sowieso ansehen möchte, bevor man sie abspeichert, bietet sich der Weg über die Menüleiste an. In manchen Fällen (z.B. wenn man für seine Bachelorarbeit viele gleich formatierte Abbildungen braucht) möchte man jedoch aus einem Skript oder einer Funktion heraus Abbildungen speichern. Hierfür gibt es den Befehl *saveas*. Damit dieser Befehl weiß, welche Grafik gespeichert werden soll, muss diese jedoch beim plotten einen Namen bekommen werden.

- Probieren Sie aus:
  - `x=1:1:100; y=sqrt(x);`
  - `h=plot(x,y);`
  - `saveas(h,'testbild.fig')`
- Schliessen Sie die Abbildung und laden Sie `testbild.fig`neu ein.
- Das Praktische an dieser Art des Speicherns ist, dass Matlab direkt verschiedene Formate speichern kann und das gewünschte Dateiformat an der Endung des Dateinamens erkennt. Unter anderem können *.jpg, .eps, .pdf* und *.bmp* benutzt werden. (Die gesamte Liste möglicher Formate und ihrer Endungen findet man in der Hilfe unter *saveas*.)
- *) Probieren Sie, die Abbildung z.B. als `bmp` abzuspeichern und in Word zu importieren.

T2C12) Eine häufig gebrauchte Anwendung ist das Einlesen von Text-Dateien, die mit einem anderen Programm abgespeichert wurden.

Wenn es sich bei der Datei ausschließlich um Zahlen handelt und in jeder Zeile der Datei gleich viele Zahlen enthalten sind, kann man in Matlab den normalen *load* Befehl verwenden.

- Erstellen Sie mit einem Textprogramm eine Testdatei *testdatei.txt* z.B. mit den Einträgen
  - 5 7 9 13.6
  - 0.5 100 1 17

- Speichern Sie sie als TEXT-Datei (NICHT word!) z.B. als `testdatei.txt` ab und laden Sie sie in Matlab mit `A=load('testdatei.txt')` ein.

*) In komplizierteren Fällen braucht man spezialisierte Befehle, deren Anwendung Sie bei Bedarf der Hilfe entnehmen können z.B.

- `dlmread` (delimited characters read) wenn die numerischen Einträge durch bestimmte Zeichen wie z.B. Semikolon getrennt sind.
- `textread` wird genutzt, um Dateien einzulesen, die gemischte numerische und ASCII Daten enthalten.
- Der `Matlab import wizard` hilft dabei, alle möglichen Dateiformate einzuladen.

* T2C13) Speziell für Nutzerinnen und Nutzer von Excel: Matlab kann excel-Dateien einlesen mit dem Befehl `variable=xlsread('dateinme.xls')`

- Probieren Sie das mit der Datei [entfernungen.xls] aus.
- Der Befehl `xlswrite('dateiname',variable)` soll eine Datei im xls-Format speichern (funktioniert aber auf meinem Apple nicht), probieren Sie ihn auf den Windows-Rechnern aus, indem Sie ein große Matrix erzeugen, abspeichern und in Excel öffnen.
- Man kann auch mit copy-paste direkt zwischen dem Array Editor in Matlab und Excel hin und her kopieren, allerdings muss man in Excel zunächst den Bereich markieren, in den eine Matrix kopiert werden soll. Probieren Sie auch das aus.

* T2C14) Speziell für Nutzerinnen und Nutzer von *R*: Leider gibt es keine vernünftige Möglichkeit, mit Matlab *R*-Dateien zu lesen oder zu schreiben. Umgekehrt geht das allerdings, man braucht dafür das *R*-package R.matlab. Probieren Sie das Konvertieren einmal aus:

- Starten Sie R, besorgen Sie sich das package R.matlab
- Laden Sie die beiden Files [WetterBremen2014.mat] und [WetterHamburg.rdata] in *R* ein. Sehen Sie sich die Daten an: Es sind die in beiden Städten gemessenen Tageshöchsttemperaturen für jeden Tag des Jahres 2014.
- Fügen Sie in *R* beide Vektoren zu einer Matrix zusammen und speichern Sie diese mit dem package als .mat file ab.
- Laden Sie das .mat file in Matlab ein und plotten Sie beide Vektoren gemeinsam in eine Abbildung.
- *) In welcher Stadt war es im Mittel wärmer?

* T2C15) Matlab kann im Prinzip sehr viele Formate importieren - dies per Hand zu tun ist aber oft anstrengend. Deshalb gibt es den "Matlab import wizard", um dabei zu helfen. Schauen Sie bei Interesse in der Hilfe nach, wie man ihn verwendet. Für eine Auflistung verschiedener Importfunktionen für verschiedene Fileformate (inklusive Sound und Filmen) suchen Sie bitte in der Hilfe nach dem Schlagwort "File Formats".

D) HAUSAUFGABEN

T2H1) Schreiben Sie eine Funktion, die aus einer beliebigen Matrix die vier "Ecken" (oben links, oben rechts, unten links, unten rechts) als Vektor zurückgibt.

T2H2) Laden Sie folgende Funktion herunter, speichern sie in Ihrem Verzeichnis ab und schauen Sie mit dem Matlab-Editor an: [kompliziert.m]

- Wie viele Ein- und Ausgabeargumente hat diese Funktion?
- Versuchen Sie im Editor die Abfolge der Programmschritte nachzuvollziehen. Was tut die Funktion?
- Testen Sie Ihre Hypothese durch geeignete Eingabeparameter. Wenn Sie die Semikolon löschen, können Sie die einzelnen Schritte im Command Window verfolgen. Alternativ verwenden Sie die Debugging-Funktion des Editors.
- Kommentieren Sie die Funktion.
- *) Was passiert, wenn man die Funktion nicht wie im Hilfetext angegeben mit zwei gleich großen Matrizen aufgerufen wird? Warum?

T2H3) Die Werte eines Vektor `v` lassen sich wie gestern gelernt mit *`plot(v)`* grafisch darstellen.

- Schreiben Sie eine Funktion, die eine Matrix und einen skalaren Wert `n` als Eingabe bekommt und die n-te Zeile dieser Matrix grafisch darstellt.
- Testen Sie die Funktion mit einigen verschieden großen Matrizen (am besten auch mit sehr großen).
- Was steckt in der Überraschungsmatrix [ueberraschung.mat]?

* T2H4) Wie bereits erwähnt, können Funktionen auch andere Funktionen aufrufen. Schreiben Sie eine Funktion, die die erste, die mittlere und die letzte Spalte einer Matrix jeweils in einem neuen Graphikfenster darstellt, und dazu die Funktion aus T2H3 benutzt. Ein neues Graphikfenster öffnet man mit dem Befehl *`figure`*.

* T2H5) Schreiben Sie eine Funktion, die die Elemente auf der Diagonalen einer quadratischen Matrix als Vektor zurückgibt.

* T2H6) Sie haben gestern auch schon einige Funktionen kennengelernt, z.B. *`sin`* und *`rem`*. Wie viele Ein- und Ausgabeargumente haben diese Funktionen jeweils? Schauen Sie in der Hilfe nach.