Informatik – Erste Schritte mit MATLAB

1. Der Bildschirmarbeitsplatz

Ihr Rechner ist über ein **Netzwerk** (Kabel + Programme für den Datenaustausch mit einem speziellen Rechner (im RZBT) verbunden, dem **Server.** Der Server verwaltet u.a., wer Rechner und Programme benutzen darf. Auch Programme, die man gemeinsam nutzen möchte, sind dort auf einer großen Festplatte gespeichert.

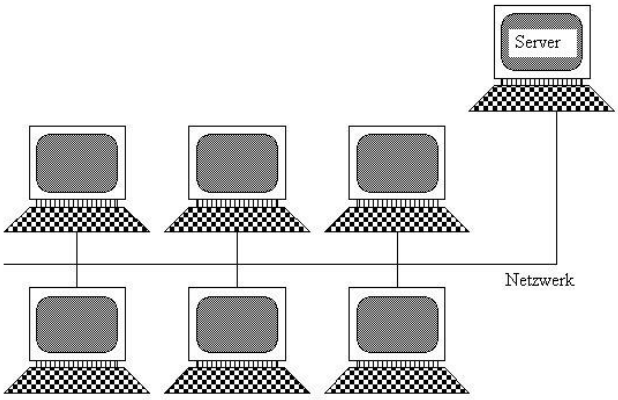


Abb.: Rechnernetz mit einzelnen Rechnern und einem Server

Auf den Festplatten des Servers steht Ihnen **Speicherplatz** zur Verfügung, in dem Sie Ihre Programme langfristig speichern können, und zwar in einem **persönlichen Verzeichnis** (Heimatverzeichnis, Home Directory) auf **Laufwerk H:**. Auf **Laufwerk P:** finden Sie den so genannten **Public-Bereich**, wo Dozenten allgemeine Daten ablegen.

Der Home-Bereich auf dem Server (auf Laufwerk H:\) bleibt Ihnen das gesamte Studium erhalten. Der Übersicht wegen sollten Sie dort deshalb so genannte **Ordner** (Verzeichnisse, Directories) erstellen, in denen Sie die Dateien zu den einzelnen Veranstaltungen sammeln. Klicken Sie hierzu z.B. auf das Icon **Arbeitsplatz** und erzeugen Sie z.B. einen **Ordner** namens **Informatik**, in dem Sie in Zukunft alle Ihre Programme zu Informatik speichern - und dann für unser erstes Labor auch gleich ein Unterverzeichnis **Labor1**:

Die Labor-Rechner werden regelmäßig „**geputzt**“, d.h. Daten auf den lokalen Festplatten, z.B. unter "C:\", werden dabei gelöscht. Wenn Sie also Ihre Arbeit aufbewahren wollen, dann müssen Sie Ihre Daten in Ihrem **Home-Verzeichnis H:** auf dem Linux-Server ablegen.

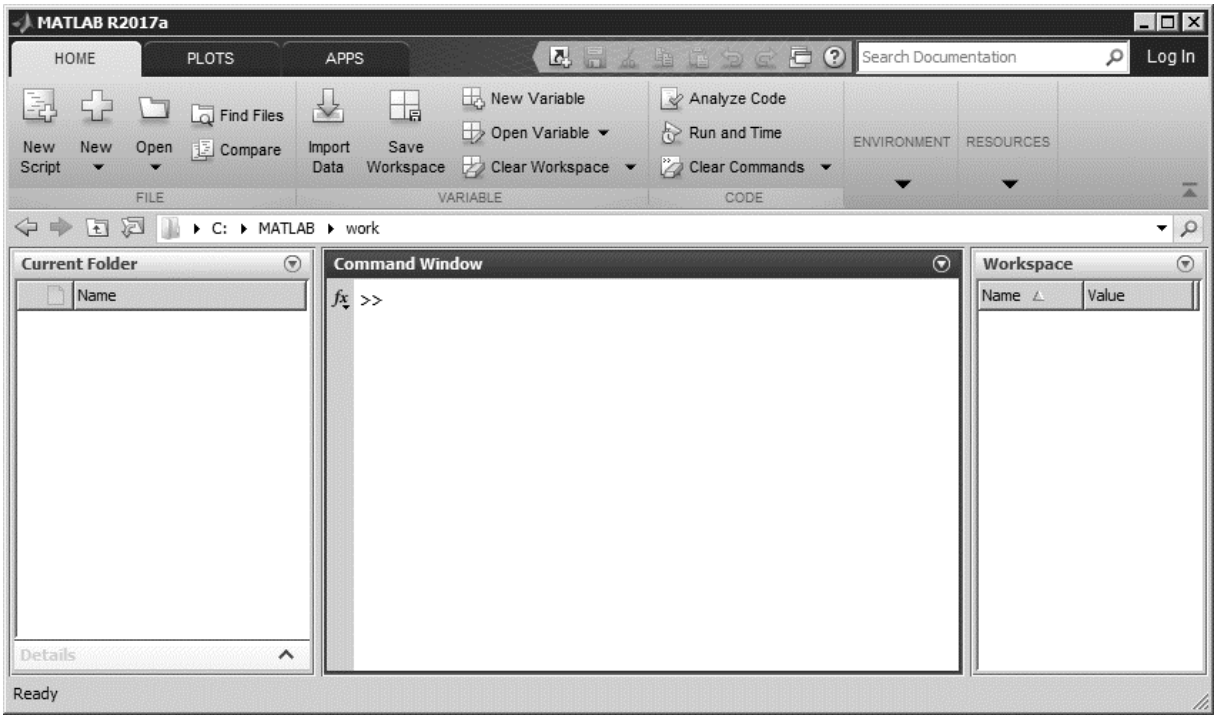
Arbeiten Sie später grundsätzlich in Ihrem persönlichen Verzeichnis!!!

Sie können Ihre Daten aber auch auf einem **USB-Stick** abspeichern.

2. Starten von MATLAB

Zum Start von MATLAB klicken Sie auf das MATLAB-Icon auf dem **Desktop** bzw., falls es dort nicht vorhanden ist, gehen Sie links unten auf das Symbol **Start** und wählen dann unter Programme im Untermenü MATLAB, z. B. **MATLAB 2018a**.

Ein Klick startet die Programm-Umgebung von MATLAB. Mit dieser können Sie jetzt MATLAB-Programme ausführen, aber auch eigene Funktionen erzeugen.



Die Programm-Umgebung von MATLAB hat mehrere Fenster. Im **mittleren Fenster**, dem "**Command Window**" können Sie hinter dem **Prompt** ">>" MATLAB-Befehle eintippen.

2.1 MATLAB als Taschenrechner

Versuchen Sie es im **Command Window** zu Beginn mit folgendem Befehl:

```
>> 1+1
```

Nach dem Drücken der Eingabetaste antwortet MATLAB:

```
ans =
     2
>>
```

Versuchen Sie es im Command Window mit weiteren mathematischen Aufgaben, wie mit einem **Taschenrechner**. Verwenden Sie die mathematischen Symbole: +, -, * und /, z.B.

```
>> (20+4*4) / 9
```

MATLAB bietet auch weitere gewohnte **Funktionen**, z.B. für die Quadratwurzel:

```
>> sqrt(49)
ans = 7
```

oder für das Quadrieren ^ bzw. für die trigonometrischen Funktionen *sin*() und *cos*():

```
>> sin(1)^2 + cos(1)^2
ans = 1
```

Hilfe zu einer Funktion erhalten Sie über die *help*-Funktion, z.B.

```
>> help sin
SIN    Sine.
    SIN(X) is the sine of the elements of X.
```

Der Befehl *help*, ohne weitere Spezifikation, gibt Ihnen die Liste aller vorhandenen *help*-Bereiche. Beispiele finden Sie über den Befehl "`help demos`". Versuchen Sie es auch einmal mit dem Befehl "`doc sin`", wodurch die grafische Hilfe gestartet wird.

2.2 Zahlen- und Text-Darstellung

MATLAB kennt folgende Zahlen-Darstellungen:

- **ganze** Dezimalzahlen, mit oder ohne Vorzeichen: 1 -2 +30 -400
- **Dezimalbruch** mit Dezimalpunkt und mindestens einer Dezimalziffer vor und nach dem Punkt, mit oder ohne Vorzeichen: 1.5 -2.0 +.25 -425. 3.141592
- dezimale **Gleitkommazahlen**, wobei Mantisse und Exponent mittels des Buchstabens e (entspricht einem "10-hoch") verknüpft sind: 1.0e+3 ist also 1.0*10 hoch 3 = 1000, weitere Beispiele: .1e-5 = $0.1*10^{-5}$, 5.e3 = $5*10^{3}$, 2e3 = $2*10^{3}$

Solche Zahlendarstellungen werden als **Literale** bezeichnet. Daneben gibt es noch Konstante, die über ihren **Namen** angesprochen werden, z.B. die Zahl pi = 3.14...

Experimentieren Sie mit diesen Zahlen-Darstellungen und rechnen Sie damit!

Beispielsweise:

```
>> 1.5 * 2e2
ans = 300

>> pi * 1e-1
ans = 0.3142
```

Achtung**:** MATLAB, als amerikanisches System, verwendet an Stelle des Dezimal-Kommas den **Dezimal-Punkt**!

Texte werden in MATLAB zwischen zwei (einfache) Hoch-Kommata gesetzt, z.B.

```
>> 'Das ist ein Text'
ans = Das ist ein Text
```

Die **Antwort** ans des Systems können Sie **unterdrücken**, wenn Sie hinter die Anweisung ein Semi-Kolon setzen, z.B.

```
>> 'Das ist ein Text';
```

2.3 Variablen und Datentypen

Man kann in MATLAB, wie in anderen Programmiersprache auch, **Variablen** definieren - d.h. man legt im Computer einen Speicherplatz mit selbst gewähltem **Namen** (Variablen-Name) an. In diesen Speicherplatz kann man Daten schreiben (**Wert-Zuweisung**) und später auf diese Daten über den Namen wieder zugreifen. Namen beginnen in MATLAB immer mit einem Buchstaben. Dann können Ziffern und "_"-Zeichen folgen - jedoch kein Leerzeichen! Zwischen Groß- und Kleinschreibung wird unterschieden. Bis zur Länge von 31 Zeichen sind die Namen eindeutig.

Variable werden in MATLAB erzeugt, indem man einen Namen wählt (z.B. *x*) und diesem Namen einen Datentyp und einen Wert zuweist, in der Form "***Name = Datentyp(Wert)***", z.B. dem Speicherplatz mit dem Variablen-Namen *x* den ganzzahligen Wert 5 :

```
>> x = int32(5)
x = 5
```

Typische **Datentypen** in MATLAB sind:

- ***int32*** : ganze (integer) Zahlen (Länge 4 Bytes = 32 bit)
- ***double*** : reelle Zahlen (Länge 8 Bytes, doppelte Genauigkeit)
- ***char*** : Text-Zeichen (character)

Der Befehl "`help datatypes`" listet alle in MATLAB vorhandenen Datentypen auf.

Den Datentyp einer speziellen Variablen kann man mit dem Befehl ***whos*** auslesen, z.B.

```
>> whos x
  Name        Size              Bytes  Class
  x           1x1                   4  int32 array
```

x ist in unserem Fall also vom Typ *int32*, genauer gesagt ein 1x1-Feld (Array) vom Typ *int32*.

MATLAB lässt auch zu, dass man den Datentyp nicht explizit angibt. In diesem Fall wählt MATLAB **automatisch** einen passenden Typ aus, z.B. für Zahlen den Typ *double*

```
>> y = 6
y = 6

>> whos y
  Name        Size             Bytes  Class
  y           1x1                  8  double array
```

Vorsicht: Die Namen der Variablen und auch die der Literale sind in MATLAB nicht geschützt und können überdefiniert werden!

3. Funktionen / M-Files

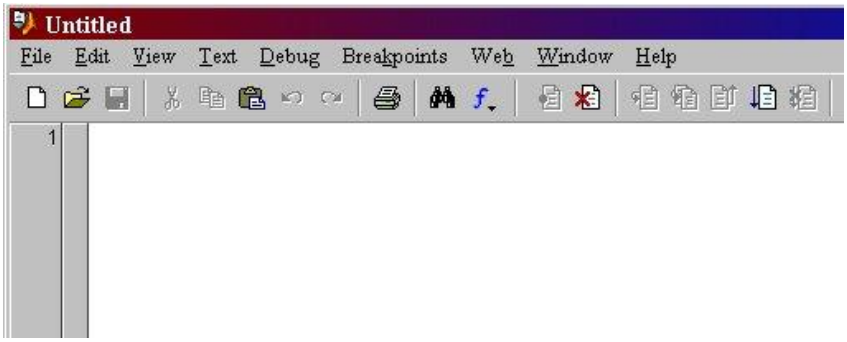
In diesem Abschnitt sollen die Schritte vom Schreiben des Funktionstextes bis zur lauffähigen MATLAB-Funktion geübt werden.

- Schreiben (**Editieren**) des Funktionstextes
- Aktuelles MATLAB-Verzeichnis wählen
- **Ausführen** der Funktion

Um eine (wieder verwendbare) MATLAB-Funktion zu schreiben, muss man zuerst eine **Text-Datei** erzeugen und diese in einem Verzeichnis der Festplatte speichern. Jeder beliebige ASCII-Editor ist zum Schreiben der Datei mit dem Funktionstext geeignet. Wir werden hierfür den in MATLAB integrierten Editor verwenden.

3.1 Der interne MATLAB-Editor

Rufen Sie in der MATLAB-Programmumgebung **Home -> New Script** auf. Es erscheint dann ein neues Fenster, in das Sie den Funktionstext eintragen können.



Schreiben Sie folgende kurze MATLAB-Funktion in die Datei "Aufgabe1.m":

```
% Meine erste MATLAB-Funktion (Kommentar)
function Aufgabe1()
  disp( 'Hello World' );
```

Sie können (und sollten) Ihren Funktionstext mit Kommentaren versehen. **Kommentare** sind Textzeilen, die mit dem Zeichen "**%**" beginnen. Diese Zeilen werden von MATLAB bei der Ausführung nicht berücksichtigt, dienen deshalb nur zur Orientierung für den Programmierer.

Mit dem Befehl **Editor -> Save** speichern Sie Ihren Text auf der Festplatte in eine Datei, also z.B. in Ihrem Verzeichnis "H:\Informatik\Labor1\" als Datei mit dem Namen "Aufgabe1.m". Die **Endung** "**.m**" schlägt MATLAB automatisch vor. Nach dem Speichern müssen Sie den MATLAB-Editor aber nicht unbedingt schließen.

Zum **Ausführen** Ihrer MATLAB-Funktion müssen Sie jetzt nur den Namen der Datei (ohne die Endung ".m") im Command-Window eintippen, z.B.

```
>> Aufgabe1;
```

Die Funktion "Aufgabe1" zeigt anschließend den Text "Hello World" im Command Window an, durch Aufruf der Funktion ***disp*** (= Display, einfache Ausgabe von Text):

```
Hello World
```

3.2 Weitere Beispiele

Verwenden Sie für jede neue Funktion einen **separaten M-File**, der exakt denselben Namen hat wie Ihre Funktion. Achten Sie dabei auf die korrekte Groß- und Kleinschreibung des Namens.

Erstellen Sie die Funktion "Aufgabe1a", die untereinander mehrere Zeilen Text ausgibt. Verwenden Sie an Stelle von *disp* diesmal die Funktion ***fprintf***, also z.B. die Anweisung `fprintf('Hello World \n');`. Als Zeilenvorschub dient das Zeichen „`\n`".

```
function Aufgabe1a()
  fprintf( 'Hello World' )
  fprintf( 'how are You \n' )
```

Testen Sie die Funktion in Matlab!

Schreiben Sie als nächstes die Funktion *Quadrat*, abgespeichert im M-File "Quadrat.m":

```
function y = Quadrat( x )
  y = x*x;
```

Die Funktion hat diesmal einen ***Übergabewert***, z.B. die Zahl 3, und gibt den berechneten Wert, hier das Quadrat von 3, zurück. Testen Sie die Funktion im Command-Window, z.B. mit dem Aufruf:

```
>> Quadrat( 3 )
ans = 9
```

3.4 Graphik-Funktionen

Zum Abschluss eine etwas komplexere MATLAB-Funktion – ohne weiteren erklärenden Kommentar. Auf die verwendeten Befehle kommen wir in späteren Vorlesungen zu sprechen.

Schreiben Sie folgenden Funktions-Text in die Datei "Aufgabe2.m":

```
% Aufgabe2: Surface Plot
function Aufgabe2()
  [X,Y] = meshgrid(-8:.5:8);
  R = sqrt(X.^2 + Y.^2) + eps;
  Z = sin(R)./R;
  surf(X,Y,Z)
  colormap hsv
  colorbar
```

Rufen Sie die Funktion "Aufgabe2" anschließend im Command-Window auf:

```
>> Aufgabe2
```

Erweiterungen:

Experimentieren Sie bei der Graphik-Funktion an Stelle von `Z = sin(R)./R;` mit anderen Z-Werten, z.B. `Z = Y * sin(R)./R;` etc.

Verwenden Sie an Stelle von hsv eine andere colormap. Suchen Sie in der Matlab-Hilfe nach den *colormap*-Optionen.

Erzeugen Sie eine eigene Map, z.B. RGB-Farben (Rot-Grün-Blau, zwischen 0 und 1):

```
map = [ 1 1 0; 1 0 0; 0 0 0; 1 1 1 ];
colormap( map )
colorbar
```