MATLAB Programme erstellen

1. Einfache Skripte

Legen Sie in Ihrem Homeverzeichnis z.B. unter **`Z:\NUMA\Uebung-2`** eine lokale Kopie der Dateien **`sphere.m`** und **`sphere_test.m`** an. Versuchen Sie als erstes den Inhalt der Dateien zu verstehen!

Starten Sie MATLAB und geben Sie den folgenden Befehl ein:

`addpath('<Pfadangabe>')` **`z. B. „Z:\NUMA\Uebung-2“`**

Geben Sie interaktiv die folgenden Befehle ein

```
>>  help sphere
>>  [u1, f1, v1]=sphere(1.0)
>>  u2=sphere(3.3)
>>  sphere_test
```

Das Beispiel zeigt, dass es neben der interaktiven Eingabe von Befehlen, *zwei weitere Möglichkeiten* gibt mit MATLAB zu arbeiten:

(a) **Skriptdateien** (e.g. `sphere_test.m`)

Sie enthalten eine Reihe von MATLAB-Anweisung, die nacheinander abgearbeitet werden sollen und tragen stets die Endung <name>**.m.** Gibt man den Dateinamen <name> als MATLAB-Befehl ein, wird die Anweisungsfolge ausgeführt, genauso als ob man diese interaktiv eingegeben hätte.

(b) **Funktionsdateien** (e.g. `sphere.m`)

Sie enthalten eine Funktionsdefinition und haben ebenfalls die Endung <fct_name>**.m**. Der Name der definierten Funktion **<fct_name>** und der *Name der Datei* ***müssen*** übereinstimmen !

Allgemeine Syntax für die Definition einer MATLAB-Funktion

```
function [r_1, ..., r_M] = <fct_name> ( p_1, ..., p_N)

    < Funktionsrumpf >  ...

return;
```

Die Eingabeparameter der Funktion **`p_1, ..., p_N`** werden in runden

Klammern nach dem Funktionsnamen angegeben. Hat die Funktion Rückgabewerte **`r_1, ..., r_M`**, so werden diese in eckigen Klammern und durch Kommata getrennt spezifiziert.
Ein Aufruf der Funktion kann an unterschiedlichen Stellen erfolgen:

- interaktiv durch die Angabe des Funktionsnamens und entsprechender Übergabeparameter
- aus Skriptdateien
- aus anderen Funktionsdateien

Weitere ***nützliche Hinweise*** zum Umgang mit Skript- und Funktionsdateien:

(a) MATLAB-Dateien können nur ausgeführt werden, wenn sie sich entweder im **aktuellen Verzeichnis** oder im **Suchpfad** befinden.

(b) Das aktuelle Verzeichnis erhält man durch die Eingabe des Befehls **`pwd.`** Mit dem Befehl **`cd`** kann man das Verzeichnis wechseln.

(c) Der Suchpfade kann mittels **`addpath`** ergänzt werden. Der Befehl **`path`** zeigt eine Liste der aktuellen Suchpfade an. Mit dem Kommando **`pathtool`** erhalten Sie ein komfortables GUI zum Editieren und Anpassen der Suchpfade**.**

(d) Jede von Ihnen erstellte MATLAB-Datei sollte einen Anfangskommentar mit einer Kurzbeschreibung der Funktion enthalten. Kommentare beginnen mit **%.**

```
%  Eine kurze aussagekräftige Beschreibung der Funktion
%  sinnvoll sind ca 3-10 Zeilen

%  Erstellt von   .....                 am ....
```

Der erste zusammenhängende Kommentarblock am Anfang einer MATLAB-Datei wird durch das **`help`**-Kommando ausgegeben.

(e) Da in Funktionen Ausgaben i. d. R. unerwünschte sind, sollten Sie durch Strichpunkte unterdrückt werden.

(f) Mit der Anweisung **`disp('text')`** können Texte ausgegeben werden; formatierte Ausgaben erhält man mittels **`sprintf`** (vgl. C-Syntax).

(g) Eine Funktion kann jederzeit mit der Anweisung **`return`** verlassen werden. Der Befehl **`error('meldung')`** bricht eine Funktion mit einer Fehlermeldung ab.

(h) Mit der Tastenkombination **`<CTRL> c`** kann die Ausführung einer Funktion manuell abgebrochen werden!

(i) Der Befehl **`diary`** protokolliert eine interaktive MATLAB-Sitzung in Form einer Skriptdatei (Defaultdatei: **diary**).

2. Kontrollstrukturen (Schleifen, bedingte Anweisungen)

Als Beispiel soll hier die **Folge der Fibonacci-Zahlen** dienen, die folgendermaßen definiert ist:

fib(1) = 1;
fib(2) = 1;
fib(n+1)=fib(n) + fib(n-1) für n > 2.

Legen Sie nun in Ihrem Homeverzeichnis eine lokale Kopie der Dateien **`fibA.m`** und **`fibB.m`** an. Versuchen Sie den Inhalt der beiden Dateien zu verstehen und testen Sie die Funktionen mit verschiedenen Aufrufparametern!

Wichtige Kontrollstrukturen:

- Die **`for`**-Schleife

```
for Laufvariable = von:schrittweite:bis
    Anweisungen
end
```

  Die **`Anweisungen`** im Rumpf der Schleife werden **`von:bis`** mit der angegebene **`schrittweite`** ausgeführt. Anstelle des Zeilenvektors **`von:schrittweite:bis`** können auch beliebige andere Zeilen-vektoren angegeben werden.

- Die **`while`**-Schleife

```
while Bedingung
    Anweisungen
end
```

  Die **`Anweisungen`** im Rumpf der Schleife werden ausgeführt, solange die Auswertung der **`Bedingung`** den Wert **`true`** liefert.

  Zur Formulierung von Bedingung stehen u.a. die folgenden Vergleichsoperatoren
  == (equal), **~=** (not equal)
  <, >, <=, >=
  und logischen Verknüpfungen
  and &, or | sowie **not ~**
  zur Verfügung!

- Die **if**-Verzweigung

```
if  Bedingung
     Anweisungen_1
[else                              % optional
     Anweisungen_2 ]
end
```

Ist die **Bedingung** erfüllt, dann werden die **Anweisungen_1** ausgeführt, ansonsten (sofern vorhanden!) werden die **Anweisungen_2** im **else**-Zweig durchlaufen.

Hinweis zur Praxis:

MATLAB bietet zwar alle gängigen ***Schleifenkonstrukte*** an, diese sollten aber aus Effizienzgründen ***sparsam verwendet*** werden und ggf. durch geeignete Matrixoperationen ersetzt werden!

Advanced Programming:

Vergleichoperatoren können auch auf ***Vektoren und Matrizen*** komponentenweise angewendet werden. Als Ergebnis erhält man wiederum einen Vektor/Matrix, deren Einträge **0** bzw. **1** den logischen Werten **false** and **true** entsprechen. So können zum Beispiel Laufschleifen vermieden werden.

Beispiel:

```
v = randn(100,1);

g = v < 0;      % Vektor g enthält eine 1 für diejenigen
                % Komponenten von v, die negative sind

k= sum(g);      % Anzahl der negativen Komponenten von v

v (v < 0) = 0; % Verwendet einen logischen 0/1 Vektor
               % zum indizieren;
               % Resultat: alle negativen Einträge in v
               % werden 0 gesetzt!
```