Heidelberg, ab dem 22.10.2012

Eine kurze Einführung in Octave

Matthias Klinger

Arbeitsgruppe Numerik und Mathematische Methoden der Simulation

Universität Heidelberg

Übersicht

- Organisatorisches
- Was ist Octave???
- Block A: Octave als 'wiss. Taschenrechner' auf dem Computer
  - Zuweisung von Werten an Variablen.
  - Anlegen und Bearbeiten von Vektoren & Matrizen.
  - Operatoren auf Vektoren & Matrizen.
- Block B: Mit Octave programmieren
  - Anlegen von Funktionen mit Octave.
  - Simple Funktionsauswertungen.
  - Schleifen und Verzweigungen.
  - Realisierung von numerischen Algorithmen.

Was ist Octave???

- Frei erhältliche Software zur numerischen Auswertung von mathematischen Problemen:
  - Lösung von LGS, Integration, Nullstellenber., etc. (Numerik 0).
  - Lösung von Gewöhnlichen Differentialgleichungen (Numerik 1).
  - Lösung von Partiellen Differentialgleichungen (Numerik 2).
- **Octave** basiert auf einer Skriptsprache, die im wesentlichen kompatibel zur Syntax des kommerziellen Programms **Matlab** ist.
- **Matlab** ist ebenfalls auf den Rechnern im CIP-Pool installiert und kann später als grafische Benutzeroberfläche für die **Octave** Programme benutzt werden.
- Die grafische Ausgabe erfolgt über das Programm **Gnuplot**, welches in **Octave** enthalten ist.
- Die Befehlseingabe bei beiden Programmen erfolgt über die Kommandozeile.

Woher bekommt ihr Octave???

- Windows:
  - Installiert **GNU Octave**:
    1. `http://sourceforge.net/projects/octave/files/`
    2. Klick auf `Octave Windows binaries`!
    3. Ladet euch eine **Octave** Version herunter.
       Unser Tipp **Octave 3.2.4.**:
       Klick auf `Octave 3.2.4 for Windows MinGW32 Installer`
       Laden Sie die `*setup.exe` herunter und installieren Sie diese!
  - Installiert euch als Editor das Programm **notepad++**:
    Z. B. erhältlich auf `http://notepad-plus-plus.org/`
- Linux:
  - In der Regel findet sich **Octave** in den Paketverwaltungen der gängigen Linux-Distributionen (Suse, Ubuntu etc.).
  - Editor: **kate**, **vim**, etc.

BLOCK A

Starten von Octave

Windows

- **Octave** wird wie jede andere Executable-Datei unter Windows ausgeführt
- Es öffnet sich ein schwarzes Fenster, welches im folgenden als Shell, Terminal oder Kommandofenster bezeichnet wird
- In das Fenster könnt ihr analog wie im Linux-Fall Befehle in die Kommandozeile eingeben
- Ihr seit jetzt startbereit und könnt die folgenden 2 Folien überspringen

Linux (Rechner im CIP-Pool)

- Terminal öffnen!
- Wenn Ihr Erfahrungen mit Linux habt könnt ihr die nächste Folie überspringen und direkt Octave starten, so wie es auf der übernächsten Folie beschrieben ist

Linux-Terminal

- Dateien und Verzeichnisse anzeigen lassen:

  ```
  ls
  ```

- Ein Verzeichnis anlegen:

  ```
  mkdir TestOrdner
  ```

- Benutzt abermals `ls`. Was hat sich geändert?
- In das Verzeichnis wechseln:

  ```
  cd TestOrdner
  ```

- Wieder `ls` verwenden. Beobachtung?
- In das übergeordnete Verzeichnis zurück wechseln:

  ```
  cd ..
  ```

- `ls`. Und jetzt?
- Löschen von Dateien und Verzeichnissen:

  ```
  rm -r TestOrdner
  ```

- `ls`. Der Ordner sollte verschwunden sein!

Aufgabe

Erstellt einen Ordner der 'OctaveIntro' heißt und wechselt in diesen Ordner!

Octave oder Matlab öffnen

- Octave **öffnen**: Dazu braucht man die folgende Eingabe in die Konsole

  ```
  octave
  ```

- Octave **schließen**: Dazu tippt man:

  ```
  quit
  ```

  oder

  ```
  exit
  ```

- Matlab **öffnen**: Tippt:

  ```
  matlab &
  ```

- Matlab **schließen**: Ihr seht, dass Matlab ein eigenes Fenster geöffnet hat (eine sogenannte grafische Benutzeroberfläche). In der Mitte befindet sich wieder eine Kommandozeile (>>). Ihr schließt das Programm über 'File'→'Exit MATLAB'.

Handwerkszeug

1) Variablen
2) Vektoren und Matrizen
3) Grundlegende Operatoren
4) Ein paar integrierte Funktionen und Hilfen
5) Grafische Ausgabe von Funktionen

Variablen

- Variablen werden durch die Zuordnung von Werten angelegt.
- Variablennamen müssen mit einem Buchstaben anfangen, ansonsten dürfen Buchstaben, Zahlen und Unterstriche benutzt werden.
- Es wird zwischen Groß- und Kleinschreibung unterschieden ($A \neq a$).
- Variable anlegen und Wert zuordnen:

  `octave:1 >` $a = 3.14159;$
- Variable ausgeben:

  `octave:2 >` a

  ergibt:

  $a \;=\; 3.14159$

Variablen

- Optisch ansprechendere Ausgabe:

  ```
  octave:1> disp([' Variable a =' , num2str (a)])
  ```

  bewirkt

  ```
  Variable a = 3.14159
  ```

- Das Semikolon hinter einer Wertzuweisung unterdrückt die Ausgabe. So weist

  ```
  octave:2 > a = 3.14159
  ```

  der Variable *a* den Wert 3.14159 zu und erzeugt die Ausgabe

  ```
  a  =   3.14159.
  ```

Bis auf Ausnahmen sind die Werte, die wir Variablen in **Octave** zuordnen können, $n \times m$ Matrizen. Ein Skalar ist dann eine 1×1 Matrix. Zeilenvektoren sind $1 \times m$ Matrizen und Spaltenvektoren $n \times 1$ Matrizen.

Vektoren erstellen

Vektoren anlegen

a) Komponentenweise eingeben

b) Mit (vorgegebenen) Funktionen sukzessive füllen.

- `octave:1>` $a = [1, 2, 3]$ erzeugt die Ausgabe: $a = \begin{pmatrix} 1 & 2 & 3 \end{pmatrix}$
- `octave:2>` $b = [11; 23]$ erzeugt die Ausgabe: $b = \begin{pmatrix} 11 \\ 23 \end{pmatrix}$
- Beispiele für spezielle Funktionen zum Füllen eines Vektors:
  - `octave:3>` $c = 1 : 5$ liefert den Vektor $c = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \end{pmatrix}$.
  - `octave:4>` $d = 1 : 2 : 5$ liefert den Vektor $d = \begin{pmatrix} 1 & 3 & 5 \end{pmatrix}$.
- Die Länge eines Vektors a wird mit $length(a)$ abgefragt.

```
octave:5> length(a)
ans =  3
octave:6> length(c)
ans =  5
```

Vektoren bearbeiten

- Lese-/Schreibzugriff auf die i-te Komponente von a über $a(i)$:
  - Sei $e = 11:15 = \begin{pmatrix} 11 & 12 & 13 & 14 & 15 \end{pmatrix}$

    `octave:1>` $val = e(3)$

    $val = 13$

    `octave:2>` $e(3) = 24$

    $e = \begin{pmatrix} 11 & 12 & 24 & 14 & 15 \end{pmatrix}$

- Analoger Zugriff auf Subvektoren:
  - Sei $f = -5:-1 = \begin{pmatrix} -5 & -4 & -3 & -2 & -1 \end{pmatrix}$

    `octave:3>` $g = f(2:4)$

    $g = \begin{pmatrix} -4 & -3 & -2 \end{pmatrix}$

    `octave:4>` $f(2:4) = [5\ \ 5\ \ 5]$

    $f = \begin{pmatrix} -5 & 5 & 5 & 5 & -1 \end{pmatrix}$

- Mittels $'$ werden Vektoren (und später auch Matrizen) transponiert
  - `octave:5>` g' liefert den Vektor $\begin{pmatrix} -4 \\ -3 \\ -2 \end{pmatrix}$ als Ausgabe

Matrizen erstellen

Wie bereits erwähnt, sind Vektoren spezielle Matrizen in **Octave**. Ergo lassen sich auch Matrizen sowohl explizit als auch mithilfe von Funktionen befüllen.

- `octave:1>` $A = [1,2,3;4,4,4;0,0,0]$ erzeugt $A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 4 & 4 \\ 0 & 0 & 0 \end{pmatrix}$.
- `octave:2>` $v = [1,2]$
  `octave:3>` $B = [v;5,-32]$ erzeugt $B = \begin{pmatrix} 1 & 2 \\ 5 & -32 \end{pmatrix}$.
- Beispiele für spezielle Befehle zum Füllen einer Matrix:
  - Einheitsmatrix: `octave:4>` $I = eye(3)$ erzeugt $I = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$.
  - Nullmatrix: `octave:5>` $N = zeros(3)$ erzeugt $N = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$.
  - $n \times m$ Nullmatrix: `octave:6>` $N = zeros(2,3)$ erzeugt $N = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$.
  - Eins-Matrix:`octave:7>` $N = ones(3)$ erzeugt $N = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$.
  - $n \times m$ Eins-M.: `octave:8>` $N = ones(1,3)$ ergibt $N = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$.

Matrizen bearbeiten

Sei nun $C = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$.

- $[n, m] = size(C)$ liefert den Zeilenvektor $\begin{pmatrix} n & m \end{pmatrix}$, wobei $n = \#$Zeilen und $m = \#$Spalten von C.

  Bsp.: `octave:1>` $[n, m] = size(C)$ gibt den Vektor $\begin{pmatrix} 3 & 4 \end{pmatrix}$ zurück.

- Lese-/Schreibzugriff auf den Eintrag i, j mittels $C(i, j)$

  Bsp.: `octave:2>` $val = C(1, 3)$
  $val = 3$.

- Zugriff auf n-te Zeile / m-te Spalte mittels $C(n, :)$, bzw. $C(:, m)$
  - `octave:3>` $w = C(1, :)$ liefert $w = \begin{pmatrix} 1 & 2 & 3 & 4 \end{pmatrix}$.
  - `octave:4>` $u = C(1, 3 : 4)$ hingegen ergibt $u = \begin{pmatrix} 3 & 4 \end{pmatrix}$.
  - `octave:5>` $C(:, 4) = [\,]$ löscht die 4. Spalte: $C = \begin{pmatrix} 1 & 2 & 3 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$.

Grundlegende Operatoren

- Octave kennt die elementaren Operatoren $+$, $-$, $*$, $/$ und $\hat{}$
- Operatoren werden als Matrizenoperatoren interpretiert.
- Das Voranstellen eines Punktes ($.+$, $.-$, $.*$, $./$ und $.\hat{}$) erzwingt die komponentenweise Interpretation des Operators.

Seien nun A und B $n \times n$ Matrizen, v und w Spaltenvektoren passender Größe, sowie c und d Skalare.

- Operatoren zwischen 'Matrizen'.
  - (Komponentenweise) Addition: $A + B$, $v + w$ und $c + d$
  - Multiplikation:
    - $A * B$ und $c * A$ liefern Matrizen, $c * d$ einen Skalar.
    - $A * v$, $w' * B$ und $c * v$ liefern Vektoren.
    - $v' * w = \sum_{i=1}^{n} v_i * w_i$ liefert einen Skalar.
    - $v * w'$ hingegen ergibt die Matrix $(v_i \; w_j)_{i,j=1}^{n}$.

Beispiele zu den Operatoren

Legt in Octave die folgenden Objekte an:

Sei $D = 2 * eye(2)$, $E = [4, 2; 0, 2]$, $p = [1; 0]$, $q = [1; 3]$

- `octave:1>` $D + E$ ergibt $\begin{pmatrix} 6 & 2 \\ 0 & 4 \end{pmatrix}$, `octave:2>` $p + q$ ergibt $\begin{pmatrix} 2 \\ 3 \end{pmatrix}$.
- `octave:3>` $D * E$ ergibt $\begin{pmatrix} 8 & 4 \\ 0 & 4 \end{pmatrix}$, `octave:4>` $0.5 * D \Rightarrow \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$.
- `octave:5>` $D * p$ ergibt $\begin{pmatrix} 2 \\ 0 \end{pmatrix}$, `octave:6>` $q' * E \Rightarrow \begin{pmatrix} 4 & 8 \end{pmatrix}$.
- `octave:7>` $p' * q$ ergibt 1, `octave:8>` $p * q' \Rightarrow \begin{pmatrix} 1 & 3 \\ 0 & 0 \end{pmatrix}$.

Weitere grundlegende Operatoren

- Weitere Matrix-Operatoren (A, B seien $n \times n$ Matrizen. v ,w Vektoren. c Skalar.)
  - Division: A/B berechnet $A * B^{-1}$ (falls B invertierbar).
  - Potenzieren: $A\hat{}c$ berechnet A^c.
- Komponentenweise Operatoren
  - Multiplikation '.∗': $A.*B$ und $v.*w$
  - Division './': Analog zur Multiplikation.
  - Potenzieren '.^':
    - $v.\hat{}c$ und $A.\hat{}c$ bewirkt das komponentenweise Potenzieren.
    - $c.\hat{}v$ liefert den Vektor $(c^{v_i})_{i=1}^{n}$.

Hinweis

Alle Operatoren lassen sich sinngemäß auch auf Submatrizen und -vektoren übertragen.

Beispiele zu den Operatoren

- `octave:1>` E/D, `octave:2>` $E * D\hat{} - 1$ ergeben beide $\begin{pmatrix} 2 & 1 \\ 0 & 1 \end{pmatrix}$.
- `octave:3>` $E\hat{}2 \Rightarrow \begin{pmatrix} 16 & 12 \\ 0 & 4 \end{pmatrix}$,

  `octave:4>` $D\hat{}0.5 \Rightarrow \begin{pmatrix} 1.4142 & 0 \\ 0 & 1.4142 \end{pmatrix}$.
- `octave:5>` $D.*E \Rightarrow \begin{pmatrix} 8 & 0 \\ 0 & 4 \end{pmatrix}$, `octave:6>` $p./q \Rightarrow \begin{pmatrix} 1 \\ 0 \end{pmatrix}$.

Aufgabe

a) Man erzeuge mit dem Vektor $r = (1:5)$ einen Vektor mit den ersten fünf Quadratzahlen. Verwende dazu '.^'!

b) Was erzeugt der Ausdruck `octave:6>` $2.\hat{}(1:5)$?

Funktionen und Hilfen

`octave:1>` <EINGABE>	Kurze Erläuterung
`abs(·)`	Der Absolutbetrag (Matrizen: Komponentenweise)
`sqrt(·)`	Die Wurzel
`exp(·)`	Exponentialfunktion
`sin(·)`, `cos(·)`, etc.	Trigonometrische Funktionen
`diag(·)`	Erzeugt eine Diagonalmatrix (z.B. `diag(1:5)`)
`fliplr(·)`, `flipud(·)`	Spiegelt die Matrix (vertikal, horizontal)
`sum(·)`	Summiert einen Vektor oder alle Spalten einer Matrix
`who`	Variablenbelegung (Was ist alles schon erzeugt?)
`help <Function>`	Informationen über die Funktion (z.B. `help sum`)
`help`	Übersicht über sämtliche Funktionen von Octave

Aufgabe

Man erzeuge eine 10×10 Matrix F mit dem Befehl `diag`. Anschließend bestimme man die Summe aller Spalten und der gesamten Matrix.

$$F = \begin{pmatrix} 4 & -1 & & \\ -1 & \ddots & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 4 \end{pmatrix}$$

Lösung der Aufgabe

```
octave:1> dummy1 = 4 * ones(1,10)
dummy1 =

   4   4   4   4   4   4   4   4   4   4
octave:2> dummy2 = ones(1,9)
dummy2 =

   1   1   1   1   1   1   1   1   1
octave:3> F = diag(dummy1)-diag(dummy2,-1)-diag(dummy2,1)
F =

   4  -1   0   0   0   0   0   0   0   0
  -1   4  -1   0   0   0   0   0   0   0
   :
   0   0   0   0   0   0   0  -1   4  -1
   0   0   0   0   0   0   0   0  -1   4
octave:4> sum(F)
ans =

   3   2   2   2   2   2   2   2   2   3
octave:5> sum(sum(F))
ans =  22
```

Grafische Ausgabe

- Zu einem gegebenen Vektor von Stützstellen x wird der Vektor der zugehörigen Funktionswerte $y = f(x)$ mittels `plot(x,y)` visualisiert.
  - Bsp.: $x = 0 : 0.01 : 10$; und $y = x.\hat{}2$. `octave:1> plot(x,y)`
  - Anstelle eines Vektors y kann auch eine Matrix Y übergeben werden. Dann wird der x-Vektor gegen jede Spalte in einer anderen Farbe ausgegeben (Bsp.: $Y = [x', (x.\hat{}2)', (x.\hat{}3)']$).
- Beschriftung des Plots

Titel	`title('titletext');`
Achsen	`xlabel('x-Achsentext');`
	`ylabel('y-Achsentext');`
Legende	`legend('plot1','plot2',...);`

- Achsen können mit `semilogy`, `semilogx` und `loglog` logarithmisch skaliert werden.
- Mit `figure(i)`, wobei $i \in \mathbb{N}$ können mehrere Ausgabefenster geöffnet werden. Mit `hold on` können mehrere Plots in einem Fenster ausgegeben werden.

Aufgabe zum `plot`-Befehl

Aufgabe

Man plotte die Funktion $f(x) = sin(x)$ auf $I = [-\pi, \pi]$ und beschrifte die Achsen auf sinnvolle Art und Weise. Danach gebe man dem Plot einen Titel.

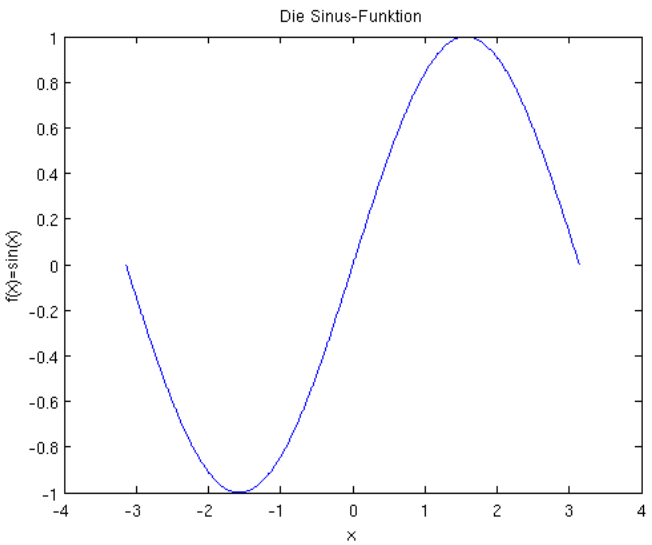
Aufgabe zum `plot`-Befehl

Aufgabe

Man plotte die Funktion $f(x) = sin(x)$ auf $I = [-\pi, \pi]$ und beschrifte die Achsen auf sinnvolle Art und Weise. Danach gebe man dem Plot einen Titel.

```
octave:1> I = -pi:0.01:pi;
octave:2> I(1:10)
I =

   -3.1416  ...  -3.0516
octave:3> fx = sin(I);
octave:4> plot(I,fx);
octave:5> title('Die Sinus-Funktion')
octave:6> xlabel('x');
octave:7> ylabel('f(x)=sin(x)');
```



BLOCK B

Programmieren mit Octave und Abgabe

- Ihr sollt mit Octave nicht nur einzelne Werte evaluieren sondern im späteren Verlauf der Vorlesung Algorithmen umsetzen.
- Dazu braucht ihr die Möglichkeit ganze Befehlsketten zu bearbeiten
- 2 Optionen: I) Ein 'Skript' schreiben oder II) Eine Funktion schreiben
- Ihr werdet immer Funktionen schreiben, die wiederum (falls notwendig) auf andere Funktionen zugreifen
- Ein Beispiel: Die Datei `main.m`

```
function erg = main()

   v=[ 15 5 -3 19 3];
   j=length(v);
   w=exp(linspace(-1,1,j));
   x=v.*w;
   k=sqrt(x*x');
   l=abs(x(3));
   y=3:l:30;
   m=length(y)/2;
   y=3:l:30;
   ans1 = (v*y(1:m)')*(w*y(m+1:2*m)');
   erg=v(floor(log(ans1*k)/l));

end
```

M-Files und eigene Funktionen

- Was ist ein m-File?
- Wie erstelle ich eigene Funktionen in einem m-File?
- Wie bekommt man **Octave** dazu, das zu tun, was man machen will?
- $\Rightarrow$ Konzipieren eines Funktionsaufbaus
- Wie beschreibt man Rekursionen/Iterationen mittels 'Schleifen' und 'Verzweigungen'?
  - for-Schleife
  - while-Schleife
  - if-Verzweigung
- Umsetzung anhand des Beispiels Horner-Schema!

Funktionen

- Alle benötigten Variablen sowie Progammabläufe können in eine Datei (m-File <name>.m) geschrieben werden.
- Parameter können auch beim Aufruf "von außen" an die Funktion übergeben werden
- Vorteile:
  - Zeitersparnis
  - Wiederverwendbarkeit von Funktionen (bei Parameterübergabe)

- Vorgehen:
  - Wir öffnen eine Textdatei mit einem Texteditor (**kate**, **gvim**, **notepad++** oder **Matlab**-Editor)
  - Prinzipieller Funktionsaufbau

```
function [<out1>,...,<outN>] = <name>(<in1>,...,<inN>)

   <Funktionsbefehle>

end
```

  - m-File wird unter dem Funktionsnamen (<name>.m) gespeichert
  - Wichtig: Die Datei muss in das Verzeichnis gespeichert werden, in dem sie ausgeführt werden soll.

Funktionen

- Arbeitsweise:

  Die Funktion besteht aus drei Teilen

  1. Der Kopf:
     - Eingabevariablen `<in1>,...,<inN>`
     - Ausgabevariablen `<out1>,...,<outN>`
     - Funktionsname `<name>`
  2. Der Rumpf `<Funktionsbefehle>`:
     - Initialisierung von (Funktions-)internen Variablen
     - Manipulation der Variablen bei Octave- oder eigener Funktionen
     - Aufbereitung und Setzung der Ausgabevariablen
     - Hier wird die Funktion 'realisiert'!
  3. Das Ende via `end` oder `return`.

- Aufruf einer Funktion:
  - Funktion wird über den Dateinamen aufgerufen.
  - Dieser sollte mit dem Funktionsnamen übereinstimmen.
  - Fehlerquelle: Oft werden die Eingangsvariablen falsch übergeben!!! Bei Fehlern sollte man hier als erstes nachschauen!

Ein Beispiel

Das Plotten der Exponentialfunktion soll als Funktion geschrieben werden. Dabei wollen wir einen Vektor mit Stützstellen x an die Funktion übergeben.

Das m-File `ExpPlot.m`

```
function erg = ExpPlot(x)

    erg = exp(x);
    plot(x,erg);

end
```

Schleifen

- `for`-Schleifen
  - Syntax:

```
for <LAUFINDEX> = <START>:<ENDE>
<OCTAVEBEFEHL-1>
<OCTAVEBEFEHL-2>
:
<OCTAVEBEFEHL-M>
end
```

  - Für eine bestimmte Anzahl von Wiederholungen wird dieselbe Gruppe von Befehlen immer wieder ausgeführt.
  - Dabei wird ein Laufindex hochgezählt, der ebenfalls in der Befehlsgruppe verwendet werden kann.
  - Beispiel:

```
sum=0;
   for i = 1:10
     sum = sum + i
   end
```

Schleifen

- `while`-Schleifen
  - Syntax:

```
while <BEDINGUNG ERFÜLLT>
<OCTAVEBEFEHL-1>
⋮
<OCTAVEBEFEHL-M>
end
```

  - Wiederholt eine Gruppe von Octave-Befehlen bis eine bestimmte 'logische Verknüpfung' nicht mehr erfüllt ist.
  - Beispiel:

```
sum=10;
   while sum>0
     sum = sum - 1
   end
```

  - Hinweis: Ändern wir im obigen Beispiel das Minus durch ein Plus, wird das Abbruchkriterium `sum>0` nie erfüllt. Wir haben dann eine sogenannte Endlosschleife erzeugt. Ein laufendes Programm kann in **Octave** mittels `Strg-C` (`Ctrl-C`) beendet werden!

Logische Vergleiche / `if`-Verzweigung

- Logische Vergleiche
  - Seien a und b zwei Skalare
    - a gleich b? $a == b$
    - a ungleich b? $a \sim= b$
    - a kleiner/größer gleich b? $a <= b$, bzw. $a >= b$
    - a kleiner/größer b? $a < b$, bzw. $a > b$
- `if`-Verzweigung
- Syntax:

```
if <BEDINGUNG ERFÜLLT>
<OCTAVEBEFEHL-A>
else
<OCTAVEBEFEHL-B>
end
```

- Anhand eines logischen Vergleichs wird entweder der eine oder der andere Octave-Befehl ausgeführt.
- Beispiel:

```
x=10;y=5;
   if (x>y) z=1
   else   z=0
   end
```

Horner-Schema

- Ziel ist es, das Horner-Schema zur Auswertung von Polynomen als Octave-Programm zu schreiben!
- Was ist das Horner-Schema?
  - Polynom n-ten Grades: $p(x) = a_{n+1}\, x^n + a_n\, x^{n-1} + \ldots + a_2\, x + a_1$
  - Andere Darst.: $p(x) = a_1 + x\,(a_2 + x\,(a_3 + \ldots + x\,(a_n + x\, a_{n+1})\ldots))$
  - Rekursion (Horner-Schema):

    1: $b_{n+1} = a_{n+1}$
    2: $b_k \quad = a_k + \hat{x}\, b_{k+1}$ für $k = n, \ldots, 1$

  $\Rightarrow$ b_1 liefert den Polynomwert an der Stelle $\hat{x}$, also $p(\hat{x}) = b_1$
- Vorteil: Statt $\sum_{i=0}^{n+1} i = \frac{(n+1)(n+2)}{2}$ Multiplikationen nur noch $n+1$ Multiplikationen bei gleichbleibender Anzahl von Additionen und leichte Stabilitätsvorteile des Horner-Schemas.

Aufbau für die Abgabe

Anhand des Horner-Schemas, soll präsentiert werden, in welcher Form ein Programm im Rahmen der praktischen Übungen abgegeben wird!

- Ihr schreibt dazu immer eine Hauptfunktion die entweder `main.m` oder `aufgabe<xy>.m` heißt!
- Der Dateiname stimmt mit dem Namen der Hauptfunktion überein! Die Hauptfunktion besitzt keine Eingangsvariablen!
- In der Hauptfunktion werden alle Variablen generiert und von dort aus weitere Unterfunktionen ausgeführt. Im vorliegenden Fall eine (noch zu erstellende) Funktion `horner(...)`!
- Aufgaben in den einzelnen Funktionsteilen der Funktion `horner(...)`:
  - Kopf:

    Eingabevariablen:
    1) Das Polynom $p(x)$ als Koeffizientenvektor $p = (a_1, \ldots, a_{n+1})$.
    2) Die Stelle $\hat{x}$ an der $p(x)$ ausgwertet werden soll.

    Ausgabevariablen:
    1) Das Ergebnis $p(\hat{x})$.
  - Rumpf:
    - Logischer Vergleich: Ist das Polynom überhaupt von x abhängig (`length(p)>1`)?
    - Falls 'Nein': Gib a_1 als Ergebnis aus.
    - Falls 'Ja': Beginne mit dem Horner-Algorithmus:

    a: Setze $b_{n+1} = a_{n+1}$
    b: Führe eine Schleife über $b_i = a_i + \hat{x}\, b_{i+1}$ solange aus, bis b_1 berechnet ist!

Lösung: Die Datei `main.m`

```
function erg = main()

    % Benoetigte Variablen
    % a ~ Koeffizientenvektor beginnend mit dem kleinsten Koeff.
    % x ~ Auswertungspunkt des Polynoms
    coeff = [ 5 0 -9 13 2 1 ];
    x = 1.345;

    % Funktionsaufruf
    erg = horner(coeff,x);

    end

    function erg = horner(a,x)

       n = length(a);
       b(n) = a(n);

       while (n>1)
           n=n-1;
           b(n) = b(n+1) * x + a(n);
       end

       erg = b(n);

    end
```

Hilfe! Ich weiß nicht mehr weiter!!!!

1. Neben dem Befehl `help` gibt es noch den Befehl `doc`, der einzelne Befehle ausführlich dokumentiert.
2. Des Weiteren nutzt die Weiten des Internets
   - `http://www.mathworks.de/help/techdoc/` liefert eine ausführliche **Matlab**-Dokumentation mit vielen nützlichen Beispielen.
   - **Vorsicht!!!** Es gibt einige kleine Unterschiede zwischen **Octave** und **Matlab**, deshalb wundert euch nicht, wenn mal etwas nicht funktioniert. In der Regel kann man aber mit einem kleinen Trick dennoch alle Probleme, die in der Vorlesung auftreten, mit **Octave** lösen!
   - `http://www.google.de` :) Holt euch Impressionen von anderen **Octave** und **Matlab** Usern und adaptiert diese. Keine Kopien! Übernehmt nichts was ihr nicht versteht!!!
3. Nur in dem Fall, dass weder 1. noch 2. zu Ergebnissen geführt haben, wendet ihr euch an uns!