Einführung in GNU Octave

Markus Herrich

13.10.2017

Inhaltsverzeichnis

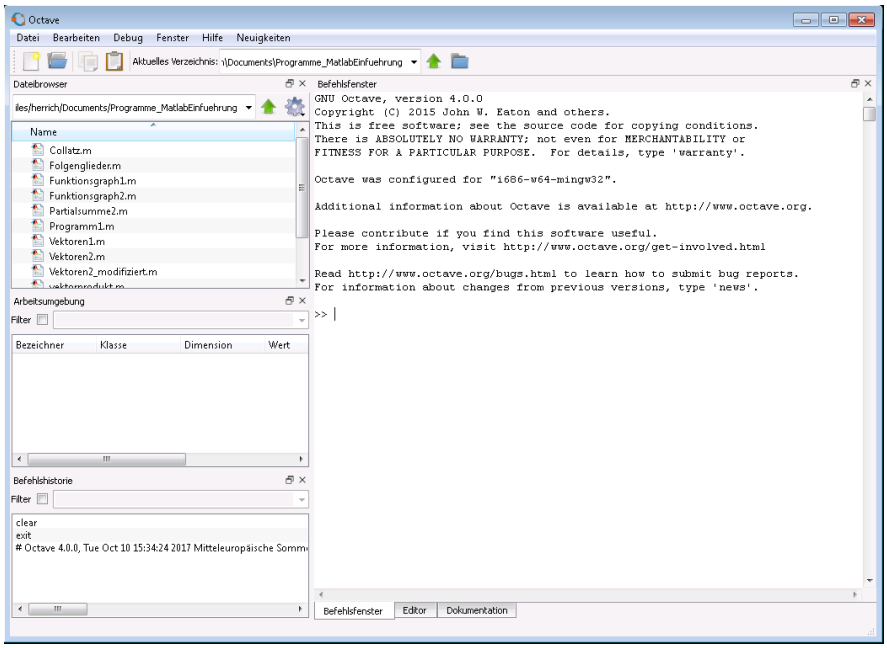
1 Start von `Octave`, allgemeine Bemerkungen

`GNU Octave` (im Folgenden nur als „`Octave`“ bezeichnet) ist eine freie Software, die zum Beispiel auf der folgenden Internetseite zum Download angeboten wird:

`https://www.gnu.org/software/octave/download.html`

Die meisten Befehle und Anweisungen in `Octave` decken sich mit denen in MATLAB (zumindest für die Befehle/Anweisungen, die wir benötigen, trifft das zu). Dementsprechend ist diese Einführung in `Octave` nahezu identisch zu meiner Einführung in MATLAB.

Wenn Sie `Octave` starten (man sagt auch: eine `Octave`-Session starten), erscheint das folgende Fenster, das in mehrere Felder unterteilt ist (wobei es je nach `Octave`-Version und Einstellungen Unterschiede im konkreten Aussehen geben kann).



Von den Feldern, die hier zu sehen sind, wird für uns später hauptsächlich das größte Feld, das Befehlsfenster, von Bedeutung sein. Trotzdem sollen die einzelnen Felder kurz beschrieben werden.

- **Befehlsfenster:** Im Befehlsfenster kann man bereits alle `Octave`-Anweisungen ausführen, die man auch innerhalb von Programmen oder `Octave`-Funktionen benutzt. Insbesondere kann man Variablen anlegen, ihnen Werte zuweisen und Rechnungen durchführen, wie wir es in den Abschnitten 2 und 3 auch tun werden. Vor allem können aber vom Befehlsfenster aus `Octave`-Programme und `Octave`-Funktionen aufgerufen werden.

  Neben dem Tab „Befehlsfenster“ können Sie hier auch zu den Tabs „Editor“ und „Dokumentation“ wechseln. Unter „Dokumentation“ finden Sie eine Beschreibung von `GNU Octave`. Unter dem Tab „Editor“ können Sie Programme und Funktionen schreiben, siehe Abschnitt 4 für Details.

- **Dateibrowser:** Hier werden die Dateien des aktuellen Ordners angezeigt. Wenn später `Octave`-Programme oder -Funktionen aufgerufen werden, sollte man in dem Ordner sein, in dem sich die entsprechende Datei des Programms oder der Funktion befindet.

- **Arbeitsumgebung:** In diesem Feld sehen Sie alle Variablen, die während der aktuellen `Octave`-Session angelegt wurden. Während der Rechnungen in den Abschnitten 2 und 3 werden Sie sehen, dass sich das Arbeitsumgebungs-Feld nach und nach füllt. Mit dem Befehl `clear` können alle Variablen aus der Arbeitsumgebung gelöscht werden.

- **Befehlshistorie:** Hier werden chronologisch die ausgeführten Befehle angezeigt, auch von vorangegangenen `Octave`-Sessions.

2 Rechenoperationen und Zuweisungen

`Octave` kann die folgenden grundlegenden Rechenoperationen ausführen: `+` (Addition), `-` (Subtraktion), `*` (Multiplikation), `/` (Division), `^` (Potenzieren). Dabei werden die bekannten Rechenregeln beachtet (zum Beispiel Punkt- vor Strichrechnung). Führen Sie als Beispiel folgende Rechnungen im Befehlsfenster aus (nachdem die Formel eingegeben ist, jeweils `Enter`

drücken): `7-3*6` und `5^2+0.75`. Das zweite Beispiel zeigt, dass das Dezimalkomma als Punkt eingegeben wird.

Neben diesen Grundrechenoperationen kennt `Octave` alle gängigen mathematischen Funktionen, zum Beispiel

- Quadratwurzel: `sqrt`
- absoluter Betrag: `abs`
- Exponentialfunktion (e hoch ...): `exp`
- Logarithmen: `log` (das ist der natürliche Logarithmus!), `log2, log10` (Logarithmen zur Basis 2 bzw. 10)
- Winkelfunktionen: `sin`, `cos`, `tan` (verlangen Argument im Bogenmaß!)
- Arcus-Funktionen: `asin`, `acos`, `atan` (liefern Ergebnis im Bogenmaß!)

Führen Sie als Beispiel folgende Rechnungen im Befehlsfenster aus: `sqrt(49)` und `sin(pi/6)`. Letzteres Beispiel zeigt schon, dass `Octave` die Konstante π kennt – sie wird mit `pi` aufgerufen. Die Konstante e (Eulersche Zahl) wiederum erhält man durch `exp(1)`.

Falls das Ergebnis einer Rechnung keine ganze Zahl ist, dann gibt `Octave` das Ergebnis üblicherweise auf fünf Stellen genau an. Man kann sich das Ergebnis auch mit einer höheren Genauigkeit anzeigen lassen, und zwar, indem man vorher den Befehl `format long` eingibt. Es gibt weitere Alternativen, zum Beispiel `format longG`, womit die Werte mit einer hohen Anzahl von Stellen ausgegeben werden, überflüssige Nullen am Ende aber weggelassen werden. Gerechnet wird in `Octave` jedoch standardmäßig immer mit höherer Genauigkeit (mit 64-Bit-Zahlen); die verwendete Genauigkeit beim Rechnen ist unabhäng davon, wie genau man sich Werte ausgeben lässt.

Oft möchte man Werte in Variablen abspeichern, um später wieder auf sie zuzugreifen. Eine Zuweisung sieht in `Octave` wie folgt aus:

`Variablenname = mathematischer Ausdruck`.

Geben Sie als Beispiel `x = 4^2-1` im Befehlsfenster ein. Dadurch wird der Variablen `x` der Wert 15 zugewiesen. Wird `x` in späteren Rechnungen verwendet, wird dafür dann der Wert 15 eingesetzt (bis dieser Variablen ggf. ein neuer Wert zugewiesen wird).

Geben Sie als weiteres Beispiel `y = 5*2+3;` im Befehlsfenster ein. Dadurch wird der Variable `y` der Wert 13 zugewiesen. Das Semikolon am Zeilenende bewirkt, dass keine Ausgabe von `y` erfolgt. Das heißt, je nachdem, ob man eine Ausgabe des Wertes haben will oder nicht, lässt man das Semikolon am Zeilenende weg oder setzt es.

Mittels `z = x+y` wird nun beispielsweise die Summe der vorher abgespeicherten Werte `x` und `y` berechnet. Probieren Sie es aus. Man kann Variablen auch neue Werte zuweisen. Zum Beispiel bewirkt die Eingabe `x = 2*x`, dass der Variable `x` nun das Doppelte ihres alten Wertes zugewiesen wird, also der Wert 30. Bei einer Zuweisung wird stets erst der Term rechts vom Gleichheitszeichen ausgewertet (im Beispiel von eben also der alte Wert von `x` benutzt) und danach die Zuweisung vorgenommen.

3 Matrizen und Vektoren

Eine Matrix

$$\boldsymbol{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$$

kann in `Octave` wie folgt eingegeben werden:

`A = [`a_{11} a_{12} `...` a_{1n}`;` a_{21} a_{22} `...` a_{2n}`; ...;` a_{m1} a_{m2} `...` a_{mn}`],`

also Zeile für Zeile, wobei zwischen je zwei Einträgen einer Zeile ein Leerzeichen steht (alternativ dazu ist auch ein Komma möglich) und ein Zeilenwechsel durch ein Semikolon kenntlich gemacht wird. Das alles ist noch in eckige Klammern einzuschließen.

Zum Beispiel werden also die Matrizen

$$\boldsymbol{A} = \begin{pmatrix} 1 & 1 & -2 \\ 3 & 4 & 0 \end{pmatrix} \quad \text{und} \quad \boldsymbol{B} = \begin{pmatrix} 5 & -3 \\ 2 & 1 \\ 0 & -4 \end{pmatrix}$$

wie folgt eingegeben:

`A = [1 1 -2; 3 4 0]` und `B = [5 -3; 2 1; 0 -4].`

Vektoren können als Matrizen mit nur einer Spalte (Spaltenvektoren) oder nur einer Zeile (Zeilenvektoren) angesehen werden. Demzufolge wird durch

- `a = [-3; 1; 2]` der Spaltenvektor $\boldsymbol{a} = \begin{pmatrix} -3 \\ 1 \\ 2 \end{pmatrix}$ eingegeben und durch
- `b = [1 4]` der Zeilenvektor $\boldsymbol{b} = \begin{pmatrix} 1 & 4 \end{pmatrix}$ eingegeben.

Will man auf den konkreten Eintrag auf der Position (i,j) der Matrix $\boldsymbol{A}$ zugreifen, geht das mittels `A(i,j)`. Entsprechend kann mittels `a(i)` auf die i-te Komponente eines Vektors $\boldsymbol{a}$ zugegriffen werden (egal, ob Zeilen- oder Spaltenvektor). Für unsere obigen Matrizen und Vektoren ist also beispielsweise `A(1,3)` gleich -2 und `a(2)` gleich 1.

Rechnen mit Matrizen und Vektoren:

- Das Apostroph `'` transponiert eine Matrix oder einen Vektor (das Apostroph ist auf den meisten deutschen Tastaturen die Zweitbelegung der #-Taste). Für unsere obigen Matrizen liefern also `A'` bzw. `b'`

$$\boldsymbol{A}^\top = \begin{pmatrix} 1 & 3 \\ 1 & 4 \\ -2 & 0 \end{pmatrix} \quad \text{bzw.} \quad \boldsymbol{b}^\top = \begin{pmatrix} 1 \\ 4 \end{pmatrix}.$$

- `+` bzw. `-` bewirken die Matrixaddition bzw. Matrixsubtraktion, sofern die Dimensionen der beiden Matrizen passen. Für unsere obigen Matrizen liefert also beispielsweise `A'+B`

$$\boldsymbol{A}^\top + \boldsymbol{B} = \begin{pmatrix} 6 & 0 \\ 3 & 5 \\ -2 & -4 \end{pmatrix}.$$

  Der Befehl `A+B` liefert hingegen einen Fehler, da die Dimensionen der Matrizen nicht übereinstimmen.

- `*` führt die Matrixmultiplikation (oder auch Matrix-Vektor-Multiplikation) aus, sofern die Spaltenanzahl des ersten Faktors mit der Zeilenanzahl des zweiten Faktors übereinstimmt. Für unsere obigen Matrizen liefert zum Beispiel `A*B` das Ergebnis

$$\boldsymbol{AB} = \begin{pmatrix} 7 & 6 \\ 23 & -5 \end{pmatrix}.$$

  Bei `A'*B` wird hingegen ein Fehler gemeldet, da die Dimensionen der Faktoren nicht passen.

  Neben der Operation `*` für die Matrixmultiplikation gibt es auch noch die Operation `.*` (bei der also noch ein Punkt vor den Stern gesetzt wird). Durch diese Operation werden die Matrizen *komponentenweise* multipliziert (es handelt sich dabei also nicht um die normale Matrixmultiplikation). Voraussetzung bei dieser Art von Multiplikation ist es, dass die Faktoren in ihren Dimensionen übereinstimmen. Für unsere obigen Matrizen wird dieses Mal somit bei `A.*B` ein Fehler gemeldet, während `A'.*B` das folgende Ergebnis liefert:

$$\begin{pmatrix} 5 & -9 \\ 2 & 4 \\ 0 & 0 \end{pmatrix}.$$

- Analog zu `.*` bewirken `./` bzw. `.^` das komponentenweise Dividieren bzw. das komponentenweise Potenzieren.

- Alle in Abschnitt 2 genannten mathematischen Funktionen (zum Beispiel `sqrt`, `exp`, `sin`) können ebenfalls auf Matrizen und Vektoren angewendet werden. Sie werden dann einfach auf jede Komponente angewendet. Beispielsweise liefert also `sqrt(b)` für unseren obigen Vektor $\boldsymbol{b}$ den Zeilenvektor $\begin{pmatrix} 1 & 2 \end{pmatrix}$.

Weitere nützliche Funktionen im Zusammenhang mit Matrizen:

- `A\b` zur Lösung eines linearen Gleichungssystems: Der Backslash-Operator zwischen einer Matrix $\boldsymbol{A}$ und einem (Spalten-)Vektor $\boldsymbol{b}$ liefert einen Vektor $\boldsymbol{x}$, der das Gleichungssystem $\boldsymbol{Ax} = \boldsymbol{b}$ löst (vorausgesetzt die Matrix $\boldsymbol{A}$ ist regulär).

- `det(A)` berechnet die Determinante einer quadratischen Matrix $\boldsymbol{A}$

- `inv(A)` berechnet die Inverse einer regulären Matrix $\boldsymbol{A}$

- `length(a)` bestimmt die Anzahl der Komponenten eines Vektors $\boldsymbol{a}$

- `size(A)` liefert einen Vektor, dessen erste Komponente die Zeilenanzahl der Matrix $\boldsymbol{A}$ ist und dessen zweite Komponente die Spaltenanzahl der Matrix $\boldsymbol{A}$ ist

- Spezielle Matrizen und Vektoren:
  - `eye(n)` liefert die $n \times n$-Einheitsmatrix,
  - `zeros(m,n)` liefert eine $m \times n$-Matrix, die nur aus Nullen besteht,
  - `ones(m,n)` liefert eine $m \times n$-Matrix, die nur aus Einsen besteht

4 Programme und Funktionen

Ein Programm besteht aus einer Abfolge von Anweisungen (`Octave`-Befehlen). Es wird beim Ausführen von oben nach unten abgearbeitet. Geschrieben wird ein `Octave`-Programm im `Octave`-Editor. Um diesen zu öffnen, gehen Sie links oben auf „Neues Skript“ oder im Befehlsfenster einfach auf den Tab „Editor“. Dadurch öffnet sich ein neues Feld, in dem das Programm implementiert werden kann.

Das Programm wird dann als Datei mit der Endung `.m` abgespeichert (wenn Sie auf „Speichern“ gehen, sollte diese Endung bereits voreingestellt sein). Zum Ausführen des Programms geben Sie dann im Befehlsfenster den Namen des Programms ein (ohne Endung `.m`). Dafür muss sichergestellt sein, dass Sie sich in dem Ordner befinden, in dem auch das Programm abgespeichert wurde (im Feld „Dateibrowser“ sollte also die Programmdatei mit auftauchen). Alternativ dazu, den Programmnamen im Befehlsfenster einzugeben, kann das Programm auch durch Betätigen von „Datei speichern und ausführen“ (gekennzeichnet durch einen gelben Pfeil im Zahnrad) im Editorfenster gestartet werden.

Beispiele für `Octave`-Programme:

1. Schreiben Sie ein Programm, das für die beiden Vektoren

$$\boldsymbol{x} := \begin{pmatrix} -1 \\ 3 \\ 2 \end{pmatrix} \quad \text{und} \quad \boldsymbol{y} = \begin{pmatrix} 1 \\ 2 \\ -1 \end{pmatrix}$$

   die (Euklidischen) Beträge, das Skalarprodukt und das Vektorprodukt der beiden Vektoren berechnet und ausgibt.

   Hinweis: Ein Musterprogramm steht auf OPAL unter dem Namen „`Vektoren1`“ zum Download bereit.

2. Erweitern Sie Ihr Programm aus Aufgabe 1, indem Sie den Nutzer die beiden Vektoren $\boldsymbol{x}$ und $\boldsymbol{y}$ eingeben lassen. Außerdem soll vor Ausgabe der Ergebnisse ein kurzer Text mitteilen, was ausgegeben wird.

   Hinweis: Die Befehle `input` und `disp` könnten hier hilfreich sein:

   - Aufforderung des Nutzers, für x einen Wert (oder einen Vektor oder eine Matrix ...) einzugeben: `x = input('Text, der auf dem Bildschirm ausgegeben wird')`
   - Ausgabe eines Textes auf dem Bildschirm: `disp('Text, der auf dem Bildschirm ausgegeben wird')`

   Hinweis: Ein Musterprogramm steht auf OPAL unter dem Namen „`Vektoren2`“ zum Download bereit.

Eine `Octave`-Funktion (`function`) besteht ebenfalls aus einer Abfolge von Anweisungen und wird ebenfalls im `Octave`-Editor geschrieben. Die Besonderheit bei einer Funktion besteht darin, dass sie im Allgemeinen von einem oder mehreren Argumenten abhängt, die ihr als Eingangsdaten übergeben werden. Mit diesen Eingangsdaten werden dann gewisse Rechnungen durchgeführt. Am Ende wird dann eine Größe zurückgegeben (Rückgabewert). `Octave`-Funktionen können von Programmen, von anderen Funktionen oder vom Befehlsfenster aus aufgerufen werden.

Die erste Zeile einer `Octave`-Funktion hat stets die folgende Gestalt:

```
function Rückgabewert = Funktionsname(Argumente)
```

(dabei ist `function` der einzige feste Bestandteil, alles andere wird durch die konkreten Namen des Rückgabewerts, der Funktion selbst und der Argumente ersetzt). In der letzten Zeile einer Funktion steht `end`. Abspeichern sollte man eine `Octave`-Funktion unter ihrem Funktionsnamen, ergänzt um die Endung `.m`.

Eine Funktion kann auch mehrere Rückgabewerte haben. In diesem Fall steht in der ersten Zeile anstelle eines einzigen Rückgabewertes vor dem Gleichheitszeichen eine ganze Liste von Rückgabewerten.

Beispiele für `Octave`-Funktionen:

1. Die durch den folgenden `Octave`-Code beschriebene Funktion berechnet die Summe zweier Größen `x1` und `x2`.

   ```
   function y = summe(x1,x2)
     y = x1+x2;
   end
   ```

   Man könnte diese Funktion nun beispielsweise vom Befehlsfenster oder von einem `Octave`-Programm aus aufrufen. Zum Beispiel würde der Aufruf `y = summe(4,2)` den Wert $4 + 2 = 6$ in der Funktion `summe` berechnen und das Ergebnis der Variable `y` zuweisen.

2. Schreiben Sie eine `Octave`-Funktion `vektorprodukt`, die für zwei Vektoren mit je drei Komponenten das Vektorprodukt berechnet. Der Funktion sollen dabei zwei Vektoren $\boldsymbol{x}$ und $\boldsymbol{y}$ als Eingangswerte übergeben werden. Zurückgegeben werden soll das Vektorprodukt $\boldsymbol{z} := \boldsymbol{x} \times \boldsymbol{y}$ als Rückgabewert.

   Hinweis: Ein Musterprogramm steht auf OPAL unter dem Namen „`vektorprodukt`“ zum Download bereit.

   Ändern Sie Ihr Programm aus dem Beispiel 2 zu `Octave`-Programmen (siehe Seite 6 oben) dahingehend ab, dass es zur Berechnung des Vektorprodukts die Funktion `vektorprodukt` benutzt.

   Hinweis: Das entsprechend abgeänderte Programm aus Beispiel 2 steht auf OPAL unter dem Namen „`Vektoren2_modifiziert`“ zum Download bereit.

5 Spezielle Strukturen innerhalb von Programmen

In diesem Abschnitt geht es um drei wichtige algorithmische Strukturen, die für die Erstellung von Programmen und Funktionen von Bedeutung sind.

5.1 if-Verzweigung

Manchmal sollen gewisse Anweisungen nur dann ausgeführt werden, wenn eine bestimmte Bedingung erfüllt ist. Wenn diese Bedingung nicht erfüllt ist, sollen gegebenenfalls andere Anweisungen ausgeführt werden. Dann kommt es also im Programm zu einer Verzweigung.

Realisieren lässt sich eine solche Verzweigung mittels `if - else - end`:

```
if (Bedingung)
   Anweisungen
else
```

```
   andere Anweisungen
end
```

Die Anweisungen im `if`-Zweig werden dabei nur dann ausgeführt, wenn die in Klammern stehende Bedingung erfüllt ist. Ist diese nicht erfüllt, werden die Anweisungen im `else`-Zweig ausgeführt. Manchmal sollen keine Anweisungen ausgeführt werden, wenn die in Klammern stehende Bedingung nicht erfüllt ist – dann kann man den `else`-Zweig auch weglassen.

Zur Eingabe der Bedingung sind die folgenden Operatoren hilfreich:

- Vergleiche: `==` (Test auf Gleichheit), `~=` (Test auf Ungleichheit), `<` (kleiner), `<=` (kleiner oder gleich), `>` (größer), `>=` (größer oder gleich)
- logische Operatoren zur Verknüpfung mehrerer Bedingungen: && (logisches „Und“), || (logisches „Oder“)

Beispiel: Schreiben Sie eine `Octave`-Funktion `verzweigung1`, der eine natürliche Zahl n als Eingangswert übergeben wird. Der Rückgabewert, den wir mit r bezeichnen wollen, soll sich wie folgt berechnen: falls $n = 1$ oder n eine gerade Zahl ist, dann ist $r = n^2$. Andernfalls ist $r = 2n$.

Hinweis: Ein Musterprogramm steht auf OPAL unter dem Namen „`verzweigung1`“ zum Download bereit.

Manchmal müssen auch mehr als zwei Fälle unterschieden werden. Das geht mittels `if - elseif - else - end`, das heißt, weitere Zweige werden mit `elseif` eingeleitet:

```
if (Bedingung 1)
   Anweisungen
elseif (Bedingung 2)
   Anweisungen
else
   Anweisungen
end
```

Beispiel: In diesem Beispiel soll eine ähnliche `Octave`-Funktion geschrieben werden wie im vorhergehenden Beispiel. Dieses Mal soll aber $r = 0$ sein, wenn $n = 1$ ist, der Fall $n = 1$ soll also dieses Mal zu einer anderen Berechnungsvorschrift führen als der Fall, dass n gerade ist. Alles andere soll so bleiben wie im vorhergehenden Beispiel. Nennen Sie die Funktion `verzweigung2`.

Hinweis: Ein Musterprogramm steht auf OPAL unter dem Namen „`verzweigung2`“ zum Download bereit.

5.2 for-Schleife

Die Nutzung einer `for`-Schleife (auch *Zählschleife*) bietet sich an, wenn eine oder mehrere (gleichartige) Anweisungen mehrmals wiederholt werden sollen und die Anzahl an Wiederholungen/Durchläufen von vornherein bekannt ist. Das sieht dann im Programm wie folgt aus:

```
for k = startwert:endwert
   Anweisungen
end
```

Die sogenannte *Laufvariable* `k` wird während des Programmdurchlaufs zu Beginn auf den Startwert gesetzt. Für diesen Startwert werden zunächst alle Anweisungen ausgeführt, die zwischen `for` und `end` stehen. Sind alle diese Anweisungen ausgeführt, wird `k` um 1 erhöht

und es wird wieder zum Anfang der `for`-Schleife zurückgegangen. Alle Anweisungen werden dann für den erhöhten Wert von `k` erneut ausgeführt. So geht es weiter, bis man bei `k = endwert` angekommen ist und die Anweisungen ein letztes Mal ausgeführt wurden. Insgesamt werden also `endwert - startwert + 1` Durchläufe ausgeführt.

Die Anweisungen dürfen (und werden in der Regel) von der Laufvariablen `k` abhängen, das heißt, für unterschiedliche Werte von `k` werden im Allgemeinen zwar ähnliche, aber nicht genau die gleichen Anweisungen ausgeführt.

Beispiele:

1. Schreiben Sie ein `Octave`-Programm, das die Folgenglieder der Folge (a_k) mit $a_k := \frac{3k-1}{2k+5}$ für $k = 0, \ldots, n$ berechnet und ausgibt. Dabei soll die Zahl n vom Nutzer eingelesen werden.

   Hinweis: Ein Musterprogramm steht auf OPAL unter dem Namen „`Folgenglieder`“ zum Download bereit.

2. Schreiben Sie ein `Octave`-Programm, das für eine vom Nutzer einzugebende natürliche Zahl $n \in \mathbb{N}$ die Partialsumme
$$s_n := \sum_{k=1}^{n} \frac{1}{k \cdot 2^k}$$
der Reihe $\displaystyle\sum_{k=1}^{\infty} \frac{1}{k \cdot 2^k}$ berechnet. Nutzen Sie zur Berechnung der Partialsumme eine `for`-Schleife.

   Hinweis: Ein Musterprogramm steht auf OPAL unter dem Namen „`Partialsumme1`“ zum Download bereit.

5.3 while-Schleife

Bei einer `while`-Schleife werden eine oder mehrere (gleichartige) Anweisungen mehrmals wiederholt, und zwar so lange, wie eine gewisse Bedingung erfüllt ist. Sobald diese nicht mehr erfüllt ist, ist die `while`-Schleife abgearbeitet. Im Programm hat eine `while`-Schleife die folgende Struktur:

```
while (Bedingung)
   Anweisungen
end
```

Es wird zunächst überprüft, ob die in Klammern angegebene Bedingung erfüllt ist. Falls ja, dann werden die Anweisungen, die zwischen `while` und `end` stehen, ausgeführt. Nachdem alle diese Anweisungen ausgeführt worden sind, wird zum Anfang der `while`-Schleife gegangen. Wieder wird die in Klammern stehende Bedingung auf Gültigkeit überprüft. Falls sie erfüllt ist, werden die Anweisungen erneut ausgeführt. So geht es weiter, bis die Bedingung einmal nicht mehr erfüllt ist. Bei der Verwendung von `while`-Schleifen sollte man darauf achten, dass garantiert ist, dass die Bedingung nach endlich vielen Durchläufen nicht mehr gilt und die `while`-Schleife beendet wird. Ansonsten besteht die Gefahr, eine Endlosschleife zu programmieren.

Zur Eingabe der Bedingung helfen dieselben Operatoren, die schon im Abschnitt für die `if`-Verzweigung genannt wurden. Insbesondere kann diese Bedingung auch die Verknüpfung mehrerer einzelner Bedingungen mittels des logischen „Und“ oder des logischen „Oder“ enthalten.

Beispiele:

1. Von der unendlichen Reihe
$$\sum_{k=1}^{\infty} \frac{1}{k \cdot 2^k}$$
ist bekannt, dass sie konvergiert. Schreiben Sie ein `Octave`-Programm, welches einen Näherungswert für den Grenzwert dieser Reihe berechnet, indem die Partialsumme
$$\sum_{k=1}^{n} \frac{1}{k \cdot 2^k}$$
ausgewertet wird, wobei n so gewählt sein soll, dass $\frac{1}{n \cdot 2^n}$ der erste Summand ist, der kleiner als die Maschinengenauigkeit eps ist (mit anderen Worten: die Summation soll abgebrochen werden, sobald ein Summand kleiner als die Maschinengenauigkeit ist).
Hinweis: Ein Musterprogramm steht auf OPAL unter dem Namen „`Partialsumme2`“ zum Download bereit.

2. Dieses Beispiel ist etwas komplexer. Innerhalb der `while`-Schleife wird hier eine `if`-Verzweigung benötigt.
Vorgegeben sei eine natürliche Zahl n. Die Folge (a_k) sei rekursiv definiert durch
$$a_0 := n, \qquad a_{k+1} := \begin{cases} \frac{1}{2} a_k & \text{, falls } a_k \text{ gerade,} \\ 3a_k + 1 & \text{, falls } a_k \text{ ungerade.} \end{cases}$$
Die Collatz-Vermutung besteht darin, dass für jede Zahl n ein Index $k \in \mathbb{N}$ existiert, sodass für das k-te Folgenglied gilt: $a_k = 1$.
Schreiben Sie ein `Octave`-Programm, das für eine natürliche Zahl n, die vom Nutzer einzugeben ist, das kleinste $k \geq 0$ bestimmt, für welches $a_k = 1$ ist. Falls das bis $k \leq 100$ noch nicht eingetreten ist, soll das Programm mit einer entsprechenden Meldung abgebrochen werden.
Hinweis: Ein Musterprogramm steht auf OPAL unter dem Namen „`Collatz`“ zum Download bereit.

6 Einfache Graphiken mit dem `plot`-Befehl

Gegeben seien zwei Spalten- oder Zeilenvektoren `x` und `y` in `Octave`, deren Komponentenanzahl übereinstimmt. Dann können diese beiden Vektoren mit dem Befehl `plot(x,y)` im Koordinatensystem gegeneinander abgetragen werden. Genauer gesagt, wird durch diesen Befehl ein extra Graphikfenster mit einem xy-Koordinatensystem geöffnet. In das Koordinatensystem werden die Punkte (x_i, y_i) $(i = 1, \ldots, n)$ eingetragen und geradlinig miteinander verbunden. Dabei seien x_i, y_i die Bezeichnungen für die Komponenten von `x` und `y`.

Mit Hilfe des `plot`-Befehls lässt sich beispielsweise der Graph einer Funktion $y = f(x)$ in einem gewissen Intervall $[a, b]$ veranschaulichen. Dazu könnte man wie folgt vorgehen: man zerlegt das Intervall in mehrere gleich lange Teilintervalle und definiert sich den Vektor `x` so, dass seine Komponenten gerade die Randpunkte dieser Teilintervalle sind:
$$x_1 = a, \qquad x_{i+1} = x_i + \frac{b-a}{n} \quad (i = 1, \ldots, n).$$
Dabei sei n die Anzahl der Teilintervalle, empfehlenswert ist etwa $n = 100$. Mit `Octave` lässt sich ein solcher Vektor sehr einfach erzeugen, und zwar mittels

```
x = a:(b-a)/n:b
```

(also erst die linke Intervallgrenze, dann ein Doppelpunkt, dann die Länge der Teilintervalle, dann wieder ein Doppelpunkt und schließlich die rechte Intervallgrenze). Anschließend bestimmt man den Vektor `y`, dessen Komponenten sich wie folgt ergeben: $y_i = f(x_i)$. Mit `plot(x,y)` wird dann, wie gesagt, der Funktionsgraph im Intervall $[a, b]$ gezeichnet.

Beispiel: Schreiben Sie ein Programm, das mit Hilfe des `plot`-Befehls den Graphen der Funktion $f(x) = \frac{3x-1}{2x+5}$ im Intervall $[0, 20]$ zeichnet.
Hinweis: Ein Musterprogramm steht auf OPAL unter dem Namen „`Funktionsgraph1`“ zum Download bereit.

Es gibt mehrere Optionen für den Befehl `plot`, die Einfluss auf das Aussehen des Graphen haben. Standardmäßig wird der Graph durchgehend und in blauer Farbe ausgegeben. Möchte man das Aussehen verändern, ist der Befehl `plot(x,y)` wie folgt zu erweitern: `plot(x,y,'[optionen]')`. Anstelle von `[optionen]` sind dann die Kürzel für die gewünschten Optionen anzugeben. Zum Beispiel sind da möglich:

- Farbe des Graphen. Standardmäßig wird der Graph in blau ausgegeben. Das würde man auch mittels `b` als Option erhalten. Weiterhin kann man einstellen: `k` (schwarz), `g` (grün), `r` (rot), `c` (cyan/türkis), `m` (magenta), `y` (gelb). Zum Beispiel zeichnet also `plot(x,y,'r')` den Graphen in roter Farbe.

- Linientyp des Graphen. Standardmäßig wird der Graph durchgehend gezeichnet. Das würde man auch mittels `-` als Option erhalten. Weiterhin kann man einstellen: `:` (gepunktete Linie), `--` (gestrichelte Linie), `-.` (abwechselnd gestrichelte und gepunktete Linie). Zum Beispiel zeichnet also `plot(x,y,'--')` den Graphen gestrichelt.

Die Optionen können auch kombiniert werden. Beispielsweise wird mit `plot(x,y,'r--')` ein roter, gestrichelter Graph gezeichnet.

Bemerkung: Zum Zeichnen von Funktionsgraphen kann alternativ zum oben beschriebenen Vorgehen der `fplot`-Befehl verwendet werden. Nehmen wir also an, dass der Graph einer Funktion $y = f(x)$ gezeichnet werden soll. Dann ist es empfehlenswert, die Funktionsvorschrift zunächst als *function handle* zu implementieren. Ein Beispiel soll verdeutlichen, wie das geht. Angenommen, die Funktion $f(x) = \frac{3x-1}{2x+5}$ soll implementiert werden. Dann schreibt man zum Beispiel folgenden Befehl in den Quelltext des Programms bzw. in das Befehlsfenster:

```
fun = @(x) (3*x-1)./(2*x+5);
```

Wichtig: Werden in der Funktionsvorschrift einer function handle zwei von x abhängige Ausdrücke multipliziert oder dividiert, ist in `Octave` der Punkt vor dem Multiplikations- bzw. Divisionszeichen erforderlich (wie bei der komponentenweise Multiplikation oder Division)!
Im weiteren Verlauf des Programms bzw. der Anweisungen im Befehlsfenster lässt sich der Funktionswert an einer Stelle x dann einfach mit `fun(x)` berechnen. So liefert beispielsweise die Zuweisung `y=fun(0)` den Wert $y = \frac{3\cdot 0-1}{2\cdot 0+5} = -\frac{1}{5}$.
Soll nun der Graph einer Funktion $y = f(x)$, deren Vorschrift als function handle unter `fun` implementiert wurde, in einem vorgegebenen Intervall $[a, b]$ gezeichnet werden, lässt sich das wie folgt mit Hilfe des `fplot`-Befehls bewerkstelligen:

```
fplot(fun,[a,b])
```

(als Argumente der `fplot`-Anweisung werden also der Name der Funktion (bzw. ihrer function-handle-Implementierung) sowie das Intervall benötigt). Dahinter können noch, genau wie bei der `plot`-Anweisung, Befehle folgen, die das Aussehen des Graphen (Farbe, Linientyp usw.) beeinflussen.

Auf OPAL steht unter dem Namen „`Funktionsgraph2`“ ein Programm für Sie zum Download bereit, in dem, wie schon in „`Funktionsgraph1`“, der Graph der Funktion $f(x) = \frac{3x-1}{2x+5}$ im Intervall $[0, 20]$ gezeichnet wird, dieses Mal aber mit dem `fplot`-Befehl.

7 `Octave`-Hilfe

`Octave` bietet zu jedem Befehl eine recht ausführliche Hilfe. Um zu dieser für einen konkreten Befehl zu gelangen, geben Sie einfach im Befehlsfenster help und dahinter den Namen des Befehls ein. Es erscheint dann eine Beschreibung darüber, was der Befehl macht. Außerdem werden, sofern es sich um eine Funktion handelt, die möglichen Eingabeargumente und die Rückgabewerte beschrieben.