WiMa-Praktikum 1

Universität Ulm, Sommersemester 2019

Woche 3

Lernziele

In diesem Praktikum sollen Sie üben und lernen:

- Arbeiten mit Kontrollstrukturen
  - Verzweigungsbefehle `if` und `switch`
  - Schleifenbefehle `for` und `while`
  - Abbruchbefehle `continue`, `break` und `return`
- Vergleichsoperatoren und logische Operatoren

Am Anfang geben wir Ihnen einen kurzen Überblick über die benötigten MATLAB-Anweisungen.

Beantworten Sie danach bitte erst einige Fragen bzw. einige off-line Aufgaben, bevor Sie sich an den Rechner setzen!

Kontrollstrukturen in Matlab

Kontrollstrukturen sind Anweisungen in prozeduralen Programmiersprachen. Man verwendet sie, um den Ablauf eines Computerprogramms zu steuern. Kontrollstrukturen dienen dazu, den Programmablauf durch bedingte Anweisungen (Verzweigungen) zu kontrollieren oder eine Anweisungsfolge wiederholt auszuführen, d. h. in einer Schleife abzuarbeiten.

Bedingte Anweisungen

Mit bedingten Anweisungen kann der Ablauf eines Programms von Bedingungen abhängig gemacht werden. In MATLAB stehen dafür die Verzweigungs-Operatoren `if` und `switch` zur Verfügung.

Die `if`-Anweisung

Mit `if` wird die Ausführung einer Anweisung vom Wert eines Ausdrucks abhhängig gemacht. Die Syntax lautet:

```
if  ausdruck
    befehle
end
```

Als Beispiel für eine `if`-Anweisung sei das folgende Skript `ex03x01.m` am Prompt ausgeführt:

```
1  % script ex03x01
2  % decreases test by 3 if test is lower or equal 2
3
4  if test <= 2
5    test = test - 3;
6  end
7  test
```

```
>> test = 5;
>> ex03x01
test =
     5
>> test = 0;
>> ex03x01
test =
    -3
>>
```

Der hier verwendete Vergleichsoperator <= wird weiter unten erklärt.

Mit dem (optionalen) `else`-Teil der `if`-Anweisung kann man zwischen zwei Alternativen auswählen. Die Syntax ist:

```
if  ausdruck
    befehle_1
else
    befehle_2
end
```

Das folgende Skript ex03x02.m,

```
% script ex03x02

if test <= 2
  test = test - 3;
else
  test = test + 3;
end
test
```

am Prompt ausgeführt, liefert dann:

```
>> test = 5;
>> ex03x01
test =
     8
>> test = 0;
>> ex03x01
test =
    -3
>>
```

Mit dem (optionalen) elseif-Teil der if-Anweisung kann man in MATLAB mehrere Ausdrücke nacheinander abfragen und die Ausführung des Programms von den Werten dieser Ausdrücke abhängig machen. (Ähnliche Möglichkeiten bietet auch die switch-Anweisung, die weiter unten erklärt wird.) Eine if-Anweisung kann mehrere elseif-Teile haben.

Die Syntax der kompletten if-Anweisung lautet:

```
if ausdruck_1
    befehle_1
[elseif ausdruck_2
    befehle_2
[elseif ausdruck_3
    befehle_3
 ...]
]
[else
    befehle_n]
end
```

Die Ausdrücke `ausdruck_1`, `ausdruck_2`, ..., `ausdruck_n` werden in der angegebenen Reihenfolge ausgewertet. Sobald ein Ausdruck wahr ist, werden die zugehörigen Befehle ausgeführt und die Abarbeitung der Kette abgebrochen. Die Ausdrücke der folgenden `elseif`-Teile werden also nicht mehr bewertet und die zugehörigen Befehls-Blöcke werden nicht mehr ausgeführt.

Ist keiner der Ausdrücke wahr, werden die Befehle des `else`-Teils ausgeführt. Dieser `else`-Teil darf auch entfallen. Dann wird keiner der Befehls-Blöcke ausgeführt.

Das folgende Skript `ex03x03.m` soll in einer Zeichenkette die Leerzeichen sowie die Buchstaben `u` und `i` zählen.

```
% script ex03x03
% analysiert einen String mit Hilfe einer if-Anweisung

s  = 'Diese Zeichenfolge soll untersucht werden';
ls = 0;          % zaehlt die Leerzeichen (spaces) in s
lu = 0;          % zaehlt die u in s
li = 0;          % zaehlt die i in s
lr = 0;          % zaehlt die restlichen Buchstaben in s

for k = 1:length(s)
  if s(k) == ' '
    ls = ls+1;
  elseif s(k) == 'u'
    lu = lu + 1;
  elseif s(k) == 'i'
    li = li + 1;
  else
    lr = lr + 1;
  end
end

disp(['Zeichenfolge = ''', s, ''''])
disp(['Die Zeichenfolge besteht aus ', ...
                      int2str(ls+lu+li+lr),' Zeichen'])
disp(['Sie enthaelt ',int2str(ls),' Leerzeichen'])
disp(['Sie enthaelt ',int2str(lu),' mal den Buchstaben u'])
disp(['Sie enthaelt ',int2str(li),' mal den Buchstaben i'])
```

```
>> ex03x03
Zeichenfolge = 'Diese Zeichenfolge soll untersucht werden'
Die Zeichenfolge besteht aus 41 Zeichen
Sie enthaelt 4 Leerzeichen
Sie enthaelt 2 mal den Buchstaben u
Sie enthaelt 2 mal den Buchstaben i
>>
```

Die `switch`-Anweisung

Die `switch`-Anweisung bietet eine komfortable Möglichkeit, eine Auswahl aus mehreren Alternativen zu formulieren. Bei der `switch`-Anweisung kann allerdings nur auf Gleichheit gegen eine Menge von bekannten Werten getestet werden.

Die Syntax der `switch`-Anweisung ist:

```
switch ausdruck
    case ausdruck_1
        anweisungen_1
    ...
    case ausdruck_n
        anweisungen_n
    otherwise
        anweisungen
end
```

Der Switch-Ausdruck `ausdruck` wird nacheinander mit den Case-Ausdrücken `ausdruck_1` bis `ausdruck_n` verglichen. Sobald der Wert des Switch-Ausdrucks `ausdruck` gleich dem Wert eines Case-Ausdrucks `ausdruck_i` ist, wird der entsprechende Anweisungsblock `anweisungen_i` ausgeführt und die `switch`-Anweisung beendet. Die weiteren Case-Ausdrücke `ausdruck_i+1` bis `ausdruck_n` werden also nicht mehr überprüft.

Mehrere Werte für einen Case-Ausdruck `ausdruck_i` können (als Cell Array, vergleiche Praktikumsblatt 7) innerhalb geschweifter Klammern { } angegeben werden.

Der `otherwise`-Block ist optional.

Das folgende kleine Testbeispiel wurde direkt im Kommandofenster eingegeben.

```
>> test = 5;
>> switch test
     case 2
       a = 2
     case {3 4 5}
       a = 5
     otherwise
       a = 10
   end
a =
     5
>>
```

Schleifen

Schleifen dienen in allen Programmiersprachen dazu, Anweisungsfolgen wiederholt auszuführen.

Die `while`-Anweisung

Die Syntax der `while`-Anweisung lautet:

```
while ausdruck
    befehle
end
```

Bei der `while`-Schleife handelt es sich um eine abweisende Schleife, d. h. der Ausdruck `ausdruck` wird vor der ersten Ausführung der Anweisung(en) `befehle` ausgewertet. Ist der Ausdruck `ausdruck` ungleich 0 (also `true`), so werden zunächst die Anweisungen `befehle` ausgeführt, um zur erneuten Auswertung des Ausdrucks dann wieder an den Anfang zu springen. Ist der Ausdruck `ausdruck` hingegen 0 (also `false`), so werden die Anweisungen bis zum folgenden `end` nicht abgearbeitet, sondern im Programm wird mit dem ersten Befehl nach dem Ende der Schleife fortgefahren, d. h. mit der ersten Anweisung nach dem abschließenden `end` der `while`-Schleife.

Das folgende Skript `ex03x04.m` soll die relative Maschinengenauigkeit bestimmen.

```
1  % script ex03x04
2  % Berechne relative Maschinengenauigkeit,
3  % d.h. kleinste Zahl eps,
4  % so dass 1 und 1+eps auf dem Rechner noch
5  % unterschiedliche Zahlen sind
6
7  clear
8  epsilon = 1;         % Variable enthaelt gesuchte Zahl
9  while 1 ~= 1 + epsilon
10   epsilon = epsilon / 2;
11 end
12 disp(['Relative Maschinengenauigkeit: ',num2str(2*epsilon)])
```

Die Ausgabe lautet beispielsweise auf einer 64-bit-Architektur:

```
>> ex03x04
Relative Maschinengenauigkeit: 2.2204e-16
>>
```

Die `for`-Anweisung

Die `for`-Anweisung bietet die Möglichkeit, einfache Initialisierungen und Zählvorgänge übersichtlich zu formulieren. Eine `for`-Schleife wird dann verwendet, wenn die Anzahl der Schleifendurchläufe im Vorfeld fest steht.

Die Syntax der `for`-Anweisung lautet:

```
for index = werte
    befehle
end
```

Hat man zuvor einen Vektor `werte` definiert, so nimmt `index` nacheinander die Werte `werte(1)`, `werte(2)`, ..., `werte(end)` an. `werte` kann auch die Form `anfangswert : endwert` oder `anfangswert : inkrement : endwert` haben.

Das folgende Skript `ex03x05.m` erzeugt mit zwei geschachtelten `for`-Schleifen eine Matrix A mit den Einträgen

$$a_{i,j} := \begin{cases} 2 & \text{falls } i = j, \\ -1 & \text{falls } |i-j| = 1, \\ 0 & \text{sonst.} \end{cases}$$

```
1  % script ex03x05
2  % Erzeugen einer [-1,2,-1]-Tridiagonal-Matrix
3
4  clear
5  n = 5;
6  A = zeros(n,n);                  % Praeallokiere Matrix
7  for i = 1:n                      % Durchlaufe alle Zeilen
8    for j = max(1,i-1):min(n,i+1) % und je bis zu 3 Spalten
9      A(i,j) = 2-3*abs(i-j);
10   end
11 end
```

Die durch das Skript ex03x05 angelegte Matrix `A`:

```
>> A

A =

     2    -1     0     0     0
    -1     2    -1     0     0
     0    -1     2    -1     0
     0     0    -1     2    -1
     0     0     0    -1     2
```

Sprunganweisungen

Mit den Sprunganweisungen `break`, `continue` und `return` wird der Programmablauf dadurch geändert, dass ohne weitere Bedingung zu einer anderen Stelle gesprungen wird.

Die `break`-Anweisung

Die `break`-Anweisung bewirkt, dass die innerste umgebende Schleife einer `for`- oder `while`-Anweisung abgebrochen wird. Die Syntax der `break`-Anweisung lautet:

```
break
```

Im folgenden Skript `ex03x06.m` wird eine Zeichenfolge in einer `for`-Schleife zeichenweise durchsucht. Beim Auftreten eines bestimmten Zeichens wird die Schleife abgebrochen.

```
 1 % script ex03x06
 2 % Zeichenfolge wird in einer for-Schleife analysiert.
 3 % Beim Auftreten eines bestimmten
 4 % Zeichens wird die Schleife abgebrochen
 5
 6 clear
 7 s = 'Zeichenfolge mit einem x an einer Stelle.';
 8 c = 'x';
 9 ist_enthalten = 'nicht ';
10 for j = s
11   if j == c
12     ist_enthalten = '';
13     break
14   end
15 end
16 disp(['Zeichenfolge = ''', s, ''''])
17 disp(['Der Buchstabe ''', c, ''' ist in der ', ...
18       'Zeichenfolge ', ist_enthalten, 'enthalten!'])
```

Nach Ausführung des Skriptes ex03x06 erhält man die Ausgabe

```
>> ex03x06
Zeichenfolge = 'Zeichenfolge mit einem x an einer Stelle.'
Der Buchstabe 'x' ist in der Zeichenfolge enthalten!
>>
```

Die `continue`-Anweisung

Die `continue`-Anweisung bewirkt, dass der aktuelle Schleifendurchlauf beendet und die Schleife mit der nächsten Iteration fortgesetzt wird. Es wird also nicht die gesamte Schleife, sondern nur ein Schleifendurchlauf abgebrochen. Die Syntax der `continue`-Anweisung lautet:

```
continue
```

Im folgenden Skript `ex03x07.m` wird eine Zeichenfolge zeichenweise durchsucht. Beim Auftreten eines bestimmten Zeichens wird der jeweilige Schleifendurchlauf abgebrochen.

```
1  % script ex03x07
2
3  clear
4  s = 'Zeichenfolge mit einigen i-Zeichen.';
5  c = 'i';
6  laenge = 0;     % enthaelt Laenge der Zeichenfolge s
7                  % ohne Anzahl der Zeichen c
8  for j = s
9    if j == c
10     continue
11   end
12   laenge = laenge + 1;
13 end
14 disp(['Zeichenfolge = ''', s, ''''])
15 disp(['Laenge der Zeichenfolge: ', int2str(length(s))])
16 disp(['Laenge der Zeichenfolge ohne die Anzahl des ', ...
17       'Zeichens ''', c, ''': ', int2str(laenge)])
```

Die Ausführung des Skriptes `ex03x07` liefert die Ausgabe

```
>> ex03x07
Zeichenfolge = 'Zeichenfolge mit einigen i-Zeichen.'
Laenge der Zeichenfolge: 35
Laenge der Zeichenfolge ohne die Anzahl des Zeichens 'i': 29
>>
```

Die `return`-Anweisung

Mit einer `return`-Anweisung kann eine Funktion an einer beliebigen Stelle verlassen und zur aufrufenden Stelle zurückgesprungen werden. Normalerweise kehrt eine Funktion zur aufrufenden Stelle zurück, wenn das Ende einer Funktion erreicht ist. Anders als in C können mit der `return`-Anweisung keine Werte an die aufrufende Stelle zurückgegeben werden.

Vergleichsoperatoren und logische Operatoren in Matlab

MATLAB stellt folgende logische Operatoren und Vergleiche zur Verfügung. Zu beachten ist hierbei, dass MATLAB den (Zahlen-)Wert 0 in den logischen Wert 0 entsprechend dem logischen Wert „false“ konvertiert und jede andere Zahl als logischen Wert 1 bzw. logischen Wert „true“ interpretiert.

Operator	Bedeutung
<	kleiner
<=	kleiner gleich
>	größer
>=	größer gleich
==	gleich
~=	ungleich
&	logisches und
|	logisches oder
~	logisches nicht
xor	logisches entweder oder
any	„true“, falls mindestens ein Element ungleich 0 ist
all	„true“, falls alle Elemente ungleich 0 sind
&&	bedingte „und“ Auswertung
||	bedingte „oder“ Auswertung

Die meisten logischen Operatoren arbeiten elementweise auf Matrizen, geben also wieder eine Matrix (mit Booleschen Werten) aus. Die Befehle `any` und `all` wirken bei Matrizen dagegen spaltenweise. Außerdem ist zu beachten, dass die Vergleichsoperatoren eine stärkere Bindung haben als die logischen Operatoren. Man kann also häufig auf Klammern verzichten.

```
>> x = [1,2;3,4]  > 1.9

x =

     0     1
     1     1

>> ~x

ans =

     1     0
     0     0

>> all(x)

ans =

     0     1
```

```
>> any(x)

ans =

      1     1

>>
```

Im Folgenden soll überprüft werden, ob die zwei Matrizen A und B gleich sind:

```
>> A = [1 2 3; 4 5 6; 7 8 9]; B = [9 8 7; 6 5 4; 3 2 1];
>> A == B

ans =

     0     0     0
     0     1     0
     0     0     0

>>
```

Man sieht, dass die Matrizen in einem Element, dem mittleren, übereinstimmen. Aber um zu überprüfen, ob die Matrizen vollständig übereinstimmen, reicht der elementweise Vergleich nicht aus. Hierfür könnte man die Funktion `all` in Kombination mit dem Doppelpunktoperator verwenden oder, um das Ganze einfacher und besser lesbar zu machen, kann man allerdings auch die spezielle Funktion `isequal` auswerten.

```
>> all(A(:)==B(:))

ans =

     0

>> isequal(A,B)

ans =

     0

>>
```

Bedingte Auswertung (auch Kurzschlussauswertung, englisch short-circuit evaluation) bezeichnet das vorzeitige Abbrechen einer Auswertung eines booleschen Ausdrucks, sobald das Auswertungsergebnis durch einen Teilausdruck eindeutig bestimmt ist. Dies ist in MATLAB mit den Operatoren `&&` und `||` für Vergleiche von Skalaren möglich. Im nachfolgenden Beispiel macht es wenig Sinn, den zweiten Term auf der rechten Seite auszuwerten, wenn b gleich Null ist. Um unliebsame Fehlermeldungen zu vermeiden, wird dieser Fall vorher durch eine bedingte Auswertung abgefangen, d. h. falls $b = 0$ gilt, wird a/b nicht mehr ausgeführt und der Wert `false` zurückgegeben.

```
x = (b ~= 0) && (a/b > 42)
```

Offline Aktivitäten

Übereinstimmen

Schreiben Sie vor jeden Begriff auf der linken Seite den passenden Buchstaben der Beschreibung, die am besten mit der aus der rechten Spalte übereinstimmt.

_____	1.	`==`	a.	Zuweisungsoperator
_____	2.	`if`	b.	Negation
_____	3.	`=`	c.	Gleichheitsoperator
_____	4.	`Pseudocode`	d.	logisches UND
_____	5.	`<=`	e.	relationaler Operator
_____	6.	`Schleife`	f.	Bedingte Anweisung
_____	7.	`&`	g.	Dient dazu, Anweisungsfolgen wiederholt auszuführen.
_____	8.	`~`	h.	Bedingte Auswertung
_____	9.	`||`	i.	Gehört zu jeder `switch`-Anweisung.
_____	10.	`end`	j.	Dient lediglich zur Veranschaulichung eines Algorithmus.

Ihre Antwort:

Füllen Sie die Lücken aus

Ergänzen Sie die folgenden Sätze.

11. Kontrollstrukturen sind _____ in prozeduralen Programmiersprachen.
12. Mit _____ Anweisungen kann der Ablauf eines Programms geändert werden.
13. Verzweigungs-Operatoren in MATLAB sind _____ und _____ .
14. Bei der `while`-Schleife handelt es sich um eine _____ Schleife.
15. Die Sprunganweisungen `break`, `continue` und `return` benötigen keine weitere _____ .
16. Die _____ -Anweisung bewirkt, dass der aktuelle Schleifendurchlauf beendet wird und die Schleife mit der nächsten Iteration fortgesetzt wird.
17. Matrizen werden in MATLAB _____-weise gespeichert.
18. Der Befehl `all(vec)` angewendet auf einen Vektor `vec` bestimmt, ob alle Einträge _____ sind.
19. Der Befehl `[1,2]&[0,1]` liefert in MATLAB _____ Fehlermeldung.
20. Für quadratische Matrizen A, B liefert `isequal(A*B,B*A)` _____ Eins.

Ihre Antwort:

Kurz und knapp

Geben Sie bitte eine kurze Antwort zu jeder der folgenden Fragen. Ihre Antwort sollte so kurz und präzise wie möglich sein; versuchen Sie es mit zwei bis drei Sätzen.

21. Erklären Sie den Unterschied zwischen `&` und `&&` sowie zwischen = und == .

    Ihre Antwort:

22. Nennen Sie zwei Anweisungen für Verzweigungs- und drei Anweisungen für Sprungbefehle.

    Ihre Antwort:

Programmausgaben

Für jedes der folgenden Programmsegmente lesen Sie zuerst die Zeilen und schreiben Sie die Ausgabe an die dafür vorgesehene Stelle.

23. Wie lautet die Ausgabe des folgenden Skripts?

```
1  x = 1;
2  while (x <= 5)
3    x = x + 1;
4    disp([ 'Der Wert von x ist: ' , int2str(x) ])
5  end
6  disp(['Der letzte Wert von x ist: ', int2str(x)])
```

24. Was ist die Ausgabe der folgenden `for`-Schleife?

```
1  for i = 0 : 2 : 5
2    disp(['Der Wert von i ist: ', int2str(i)])
3  end
```

25. Wie lautet die Ausgabe des folgenden Skripts?

```
for k = 1:10
  if ( k == 7 ), break, end
  if ( k == 3), continue, end
  disp(['Der Wert von k ist: ', int2str(k)])
end
disp(['Der letzte Wert von k ist: ', int2str(k)])
```

26. Wie lautet die Ausgabe des folgenden Skripts?

```
for k = 1:10
  switch ( k )
    case 1
      disp('Der Wert von k ist: 1')
    case {2,3}
      disp('Der Wert von k ist: 2 oder 3')
    case{5}
      disp('Der Wert von k ist: 5')
    otherwise
      disp(['Der Wert von k ist weder 1', ...
            ' noch 2, 3 oder 5.'])
  end
end
```

Korrigieren Sie den Code

Für jedes der folgenden Codesegmente sollen Sie feststellen, ob ein Fehler enthalten ist. Falls ein Fehler vorliegt, markieren Sie diesen und spezifizieren Sie, ob es sich dabei um einen Semantik- oder Syntaxfehler handelt. Schreiben Sie die korrigierten Anweisungen jeweils in jeden dafür vorgesehenen Bereich unter der Problemstellung. Falls das Segment keinen Fehler enthält, schreiben Sie einfach „kein Fehler".
Bemerkung: Es kann sein, dass ein Programm mehrere Fehler enthält.

27. Das folgende MATLAB-Segment soll ermitteln, ob jemand einen „bestanden"-Status hat. Falls dies so ist, soll **Bestanden.** ausgegeben werden. Andernfalls soll **Durchgefallen.** und **Sie müssen diese Veranstaltung nochmals belegen.** ausgegeben werden.

```
1  if (grade >= 60)
2    disp('Bestanden.')
3  end
4  disp('Durchgefallen.')
5  disp('Sie muessen diese Veranstaltung nochmals belegen.')
```

Ihre Antwort:

28. Das folgende MATLAB-Segment soll das Produkt aller natürlichen Zahlen von 1 bis einschließlich 5 ermitteln.

```
1  for i = 1:5
2    produkt = 1;
3    produkt = i * produkt;
4  end
```

Ihre Antwort:

29. Das folgende MATLAB-Segment soll das Produkt aller natürlichen Zahlen von 1 bis einschließlich 5 ermitteln.

```
1  k = 1;
2  produkt = 1;
3  while ( k <= 5 )
4    produkt = k * produkt;
5  end
```

Ihre Antwort:

30. Die folgende `switch`-Anweisung soll entweder **x ist 5.**, **x ist 10.** oder **x ist weder 5 noch 10.** ausgeben.

```
1  switch (x)
2    case 1
3      disp('x ist 5.')
4    case 2
5      disp('x ist 10.')
6    otherwise
7      disp('x ist weder 5 noch 10.')
8  end
```

Ihre Antwort:

31. Die `for`-Schleife soll die Potenz von 2 hoch `k` plus 1 ausgeben. Ist zum Beispiel die Schleifenvariable 3, dann soll das Programm **`2^3 + 1 = 9.`** ausgeben. Die Schleife soll von 1 bis 10 laufen .

```
1  tmp = 1;
2  for k = 1 : 1 : 10
3    tmp = 2 * tmp;
4    disp([int2str(k),'^2 + 1 = ', int2str(tmp)+1])
5  end
```

Ihre Antwort:

Praktikumsaufgabe - Kontrollstrukturen

Lesen Sie die Aufgabenstellung, studieren Sie dann die vorgegebenen Programmzeilen. Ersetzen Sie dann die %% Kommentare im vorgegebenen Code durch Matlab-Anweisungen und führen Sie das Programm aus.

32. i.) Schreiben Sie ein Skript `agm.m`, welches die Umsetzung des folgenden Pseudocodes in MATLAB ist. Der Algorithmus nennt sich AGM (arithmetic geometric mean) und berechnet näherungsweise π.

AGM-Algorithmus zur Berechnung von π

$$X_0 = \sqrt{2}$$
$$\pi_0 = 2 + \sqrt{2}$$
$$Y_0 = \sqrt[4]{2}$$

Für $k = 0, 1, \ldots$ wiederhole man die Iteration

$$X_{k+1} = \frac{1}{2}\left(\sqrt{X_k} + \frac{1}{\sqrt{X_k}}\right)$$
$$\pi_{k+1} = \pi_k \left(\frac{X_{k+1} + 1}{Y_k + 1}\right)$$
$$Y_{k+1} = \frac{Y_k\sqrt{X_{k+1}} + \frac{1}{\sqrt{X_{k+1}}}}{Y_k + 1}$$

Der Wert von π ergibt sich als Grenzwert π_∞.

ii.) Betrachten Sie wie in Aufgabe 29 der zweiten Woche des Wima-Praktikums den relativen Fehler $\frac{|\pi - \pi_k|}{\pi}$ für $k = 1, \ldots, 4$ und vergleichen sie diesen mit dem Ergebnis aus Aufgabe 29. Was fällt Ihnen auf?

33. Das Inverse einer reellen Zahl $x \neq 0$ kann mit der folgenden Newton-Regel bestimmt werden.

Newton-Regel zur Bestimmung des Inversen Y einer reellen Zahl x.

$$Y_{k+1} = Y_k(2 - xY_k)$$

Die Folge der Y_k konvergiert lokal quadratisch gegen $1/x$. Schreiben Sie eine Funktion `Inverse.m`, welche als Parameter eine Zahl $1 \leq x < 10$ bekommt und als Rückgabewert eine Näherung an $1/x$ liefert.

Wählen Sie als Startwert $Y_0 = 2/11$.

34. Auch für die Berechnung der Wurzel einer positiven Zahl x gibt es eine Newton-Regel.

Newton-Regel zur Bestimmung der Quadratwurzel Y einer positiven Zahl x.

$$Y_{k+1} = \frac{1}{2}Y_k(3 - xY_k^2)$$

Die Folge der Y_k konvergiert lokal quadratisch gegen $1/\sqrt{x}$. Eine Multiplikation mit x liefert $\sqrt{x}$. Schreiben Sie eine Funktion `SquareRoot.m`, welche als Parameter $1 \leq x < 100$ bekommt und als Rückgabewert eine Näherung an $\sqrt{x}$ liefert.

Wählen Sie als Startwert

$$Y_0 := \begin{cases} 1/2 & \text{für } x < 10, \\ 1/6 & \text{für } 10 \leq x < 100. \end{cases}$$