3.9 Iteration, weitere Beispiele

Das Iterationsverfahren, das wir in der letzten Lektion kennen lernten, hat auch in anderen Gebieten der Physik eine große Bedeutung. Wir wollen daher zunächst nochmals zwei Beispiele betrachten, die wir bereits angeführt hatten (2.4 und 3.2). Im Abschnitt 3.9.3 werden wir auch kurz zwei iterative Verfahren (Algorithmen) besprechen, mit denen wir die reellen Nullstellen einer Funktion f(x) finden können.

3.9.1 Der Fall einer Kugel in einer Flüssigkeit

Dieses Problem hatten wir bereits detailliert in Abschnitt 2.4 untersucht. Eine Kugel der Masse m und dem Radius R fällt mit der Anfangsgeschwindigkeit Null aus x = 0. Die Fallstrecke H unterteilen wir in n Intervalle, jedes von der Länge h = H/n.

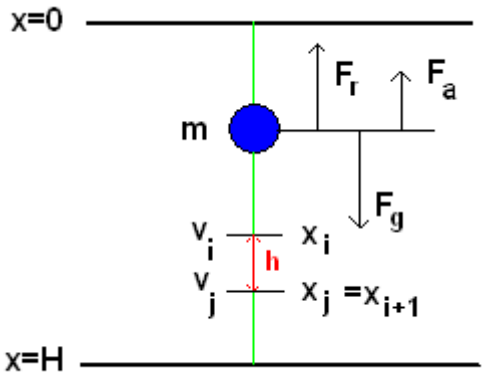


Fig. 3.9-1

Für jedes Intervall berechnen wir die mittlere Geschwindigkeit gemäß der Formel $(v_i + v_j)/2$. Auf jedem Intervall nehmen wir die Beschleunigung als konstant an. Die Beschleunigung im Intervall Nr. j ist gegeben durch

$$a_j := (v_j - v_i)/(t_j - t_i) = g[u-((v_i+v_j)/(2v_1))^2] \qquad (1)$$

Vergleichen Sie diesen Ausdruck mit der Gleichung 2.4-4 im Abschnitt 2.4-1.

Die Zeit zum Durchfallen des j-ten Intervalls beträgt

$$t_j - t_i = (2h)/(v_i + v_j) \qquad (2)$$

Diesen Ausdruck setzen wir in Gleichung (1) ein, wobei wir die folgende Abkürzung verwenden:

$$b := g*h/(2v_1)^2 \qquad (3)$$

Wir erhalten dann die folgende Iterationsformel

$$v_{i+1} = [(v_i^2 + 4buv_1^2(1+b))^{1/2} - bv_i]/(1+ b) \qquad (4)$$

Statt v_j habe ich v_{i+1} geschrieben. Ferner benutzen wir $u := 1-\rho/\rho_c$ und $v_1^2 := 8Rg\rho_c/(3C\rho)$, wo ρ die Dichte der Flüssigkeit ist (1000 kg/m^3 für Wasser), ρ_c = Dichte der Kugel (800kg/m^3), R = Radius (4 mm), C = 0.4 und g = 9.81 m/s^2 .

In Abschnitt 2.4 sahen wir, dass diese Kugel 0.25 s braucht, um eine Strecke von 0.1975 m zu durchfallen und dass ihre Geschwindigkeit in diesem Moment 1.22989 m/s beträgt. Mit dem folgenden Programm, das wesentlich ungenauer arbeitet als die Funktion `ode` von MuPAD, erhalten wir dennoch die gleichen Ergebnisse.

Um die Fallzeit T zu bestimmen, haben wir die n Teilzeiten t_j zu addieren, vgl. Gleichung (2). Für die Addition ergibt sich

$$T = \sum_{j=1}^{n} t_j = \frac{2H}{n} \sum_{i=0}^{n-1} \frac{1}{v_i + v_{i+1}} \qquad (5)$$

```
Reset(): // Fall einer Kugel in Wasser
rc:=7800:rfl:=1000:
R:=0.004:C:=0.4:g:=9.8: H:=0.2:v0:=0:
n:=100:s:=0:
u:=1-rfl/rc:
v1:=8*R*g*rc/(3*rfl*C):
b:=g*H/(2*n*v1):
d:=4*b*u*v1*(1+b):
v(0):=v0:
for i from 1 to n do
v(i):=(sqrt(v(i-1)^2+d)-b*v(i-1))/(1+b):
s:=s+1/(v(i-1)+v(i)):Berechnung der Summe
```

```
t(i):=1/(v(i-1)+v(i)):Elemente für "sum"
end_for:

print(Unquoted,"Die Zeit betraegt", 2*H*s/n,"
Sekunden");
//mit "Unquoted" werden Anfuehrungszeichen
unterdrueckt
print("T =",(2*H/n)*sum(t(j),j=1..n));
```

```
Die Zeit betraegt, 0.2521110898,  Sekunden

                "T =", 0.2521110898
```

3.9.2 Einfaches Pendel mit beliebigen Amplituden

Für diesen Abschnitt wäre es gut, wenn Sie das Kapitel 3.2 nochmals durchlesen würden. Wir untersuchten dort die einfache Gleichung y''(t) = -sin y(t) für die Anfangsbedingungen y(0) und y'(0).

Weder MUPAD noch eine beliebige Person kann dieses Problem in geschlossener Form lösen. Eine Näherungslösung kann man nur numerisch mit Iteration finden. Im Folgenden wollen wir ein sehr einfaches iteratives Verfahren entwickeln. Es handelt sich dabei um eine einfache Erweiterung der im vorigen Abschnitt entwickelten Methode, denn wir haben hier das Problem eines durch **Bindung** (Faden) beeinflussten Fallens.

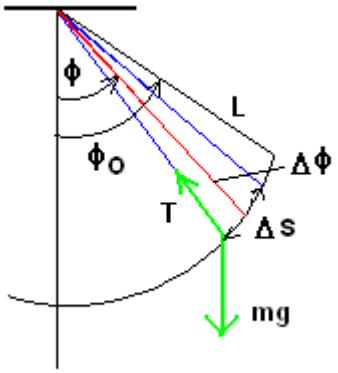


Fig. 3.9-2

Wir werden jetzt, wie in Fig. 3.9-2 gezeigt, die Amplitude φ_o in n gleiche Teilwinkel zerlegen, jeden von der Größe $\Delta\varphi = \varphi_o/n$.

Das Pendel braucht Δt Sekunden, um den Winkel $\Delta\varphi = \Delta s/L$ zu „durchfallen". Die Summe aller Δt ergibt die Periode $T := T_o \cdot K_o$, wo K_o ein Korrekturfaktor ist, der von φ_o abhängt. $T_o = 2\pi (L/g)^{1/2}$ ist die Periode des einfachen Pendels. Der Faktor K, den wir in 3.2.2 einführten, ist um $\pi/2 = 1.570796$ mal größer als das jetzige K_o.

Wir wollen annehmen, dass die Tangentialbeschleunigung in Δt konstant ist. Wir haben dann

$$a_t = (v_{i+1} - v_i)/\Delta t = g \cdot \sin\varphi \qquad (6)$$

Die mittlere Geschwindigkeit in Δt ist $(v_i + v_{i+1})/2$, und der Bogen, den das Pendel in Δt Sekunden überstreicht, ist $\Delta s = (v_i + v_{i+1})\, \Delta t/2 = L\, \Delta\varphi$. Damit ergeben sich

$$v_{i+1} = v_i + g\, \Delta t \sin\varphi \qquad (7)$$
$$\Delta t = 2\, L\, \Delta\varphi/(v_i + v_{i+1}) \qquad (8)$$

Wenn wir Δt der ersten Gleichung durch Δt der zweiten ersetzen, ergibt sich die folgende Iterationsformel für die Geschwindigkeit

$$v_{i+1} = (v_i^2 + 2\, L\, \Delta\varphi\, g \sin\varphi)^{1/2} \qquad (9)$$

Die Summe aller Δt zwischen $\varphi = \varphi_o$ und $\varphi = 0$ ergibt die Zeit T/4, und die vollständige Periode ist

$$T = \frac{8L\phi_o}{n} \sum_{\phi_o}^{0} \frac{1}{v_i + v_{i+1}} \qquad (10)$$

Der Korrekturfaktor ergibt sich aus

$$K_0 = \frac{T}{T_0} = \frac{4\phi_0}{n\pi} \sqrt{gL} \sum_{\phi_o}^{0} \frac{1}{v_i + v_{i+1}} \qquad (11)$$

K_o hängt scheinbar von g und L ab. Aber das ist nicht der Fall, denn, wenn wir eine dimensionslose Größe u einführen gemäß

$$v := u\, (g\, L)^{1/2}, \qquad (12)$$

können wir $(g\,L)^{1/2}$ eliminieren und erhalten $v_i + v_{i+1} = (g\,L)^{1/2}\,(u_i + u_{i+1})$, mit der Definition

$$u_{i+1} := (u_i^2 + 2\,\Delta\varphi\,\mathrm{sen}\varphi)^{1/2}.$$

Schließlich erhalten wir

$$K_0 = \frac{4\phi_0}{n\pi}\sum_{\phi_0}^{0}\frac{1}{u_i + u_{i+1}} \qquad (13)$$

- ```
  reset():Iteration beim Pendel
  fi1:=30://Winkel in Grad
  pi:=float(PI):
  fi0:=float(fi1*pi/180):
  DIGITS:=8:
  n:=500: //das sind viele Unterteilungen!
  for i from 1 to n do

       dfi:=fi0/n:
       fi2:=fi0-dfi/2://ver Fig.3.9-2
       v(0):=0:t(0):=0:
       b:=2*dfi*sin(fi2-(i-1)*dfi):
       v(i):=sqrt(v(i-1)^2+b):
       t(i):=1/(v(i-1)+v(i)):
       end_for:
       K0:=4*fi0/(n*pi)*sum(t(j),j=1..n);
       K:=K0*pi/2// Vergleich mit Tabelle in 3.2
  ```

```
1.0174244  (= K0)

1.5981664  (= K)   (vgl. mit Tabelle in 3.2.2, Seite 8
```

Für hinreichend kleines Δt kann die Iterationsmethode Ergebnisse liefern, die bis auf drei oder vier Dezimalstellen genau sind.

Um die Genauigkeit zu erhöhen, müssen wir die Methoden verfeinern. Einige dieser Methoden, die auch von MUPAD benutzt werden, diskutieren wir in einem späteren Kapitel.
```

3.9.3 Iterationsverfahren zum Auffinden der Nullstellen einer Funktion.

In den verschiedensten Anwendungsgebieten der Mathematik ist es oft notwendig, die Nullstellen einer Funktion f(x) zu finden. (Statt von Nullstellen spricht man auch von den Wurzeln der Gleichung f(x) = 0.)

Die reellen Nullstellen können wir mit beschränkter Genauigkeit am Graphen der Funktion ablesen, denn die reellen Wurzeln sind durch die Schnittpunkte (bzw. Berührungspunkte) des Graphen mit der x-Achse gegeben.
Für die meisten Funktionen gibt es nur numerische Methoden zur Auffindung präziser Nullstellen. Es handelt sich meist um iterative Methoden, die mit einem groben Anfangswert beginnen und sich dann iterativ mit einer vorgege-benen Genauigkeit ε der (unbekannten) genauen Nullstelle nähern.

Oft liefert der Graph der Funktion einen für den Start der Iteration brauch-baren Wert, vgl. z.B. Fig. 3.9-3.

Von den vielen bekannten Verfahren, betrachten wir hier nur die **Newton-Methode** und das Verfahren der **Bisektion**. (Vgl. Applets in http://techmath.uibk.ac.at/numbau/alex/applets/index.html)

In der ersten Methode zeichnen wir die Tangente im Punkt $(x_0,f(x_0))$ und anschließend nehmen wir die Schnittstelle der Tangente mit der x-Achse als neuen Näherungswert für die Nullstelle der Funktion f(x). Indem wir so fortfahren, erhalten wir eine Folge $\{x_i\}$ von Näherungswerten für die Lösung der Gleichung f(x) = 0.

Die Gleichung der Tangente ist gegeben durch $t(x) = f(x_i) + f'(x_i)(x-x_i)$. Mit t(x) := 0 ergibt sich $x = x_i - f(x_i)/f'(x_i)$. Wenn wir setzen $x := x_{i+1}$, erhalten wir eine Iterationsgleichung:

$$x_{i+1} = x_i - f(x_i)/f'(x_i) \qquad (14)$$

Beispiel:

Bestimme eine Folge von Werten x_i , die gegen die exakte Wurzel der Gleichung $x^3 - 2x - 5 = 0$ konvergiert, und illustriere die ersten Iterationsschritte graphisch.

Diese Gleichung ist nach dem englischen Mathematiker *John Wallis* (1616-1703) benannt. Man benutzt sie gern, um verschiedene Iterationsmethoden zu testen. Ihre Lösung wurde mit mehr als 4000 gültigen Dezimalstellen berechnet. Sie brauchen nur etwa 15 richtige Dezimalstellen zu finden.

Lösung:

- ```
  f:=x->x^3-2*x-5:// John Wallis (1616-1703)
  g:=plot::Function2d(f(x),x=-1..3,Color=RGB::Red):
  plot(g)
  ```

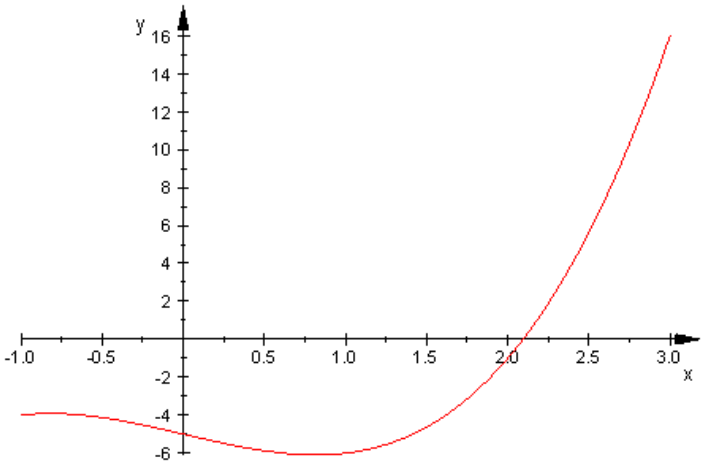


Fig. 3.9-3

Die Nullstelle der Funktion liegt nahe bei 2.1. Für den nächsten Schritt benutzen wir aber absichtlich einen weniger genauen Wert, $x_0 = 2.5$. Ich möchte zeigen, dass es in diesem Fall nicht nötig ist, den bestmöglichen Näherungswert für den Start der Iteration zu benutzen.

- ```
  x0:=2.5:
  tang:=f'(x0)*(x-x0)+f(x0):
  t1:=plot::Function2d(tang(x),x =1.5..3.5):
  plot(g,t1)
  ```

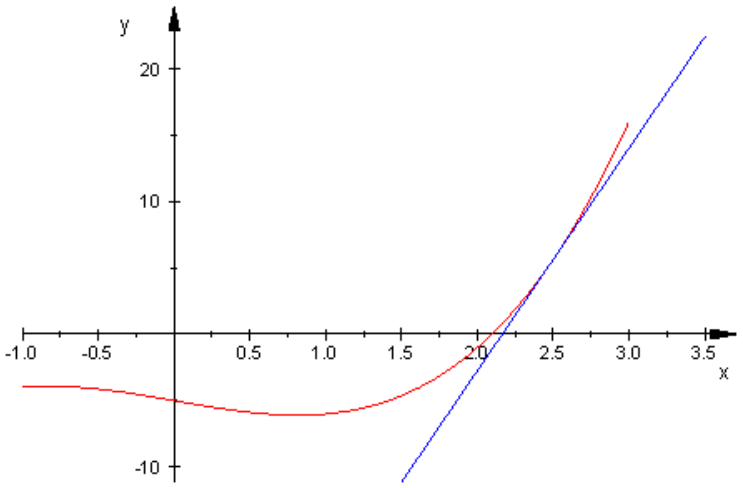


Fig. 3.9-4

Wir benutzen erneut Gl. (14) mit dem Tangentenschnittpunkt als neuem Anfangswert für die Iteration:

```
x1:=float(x0-f(x0)/f'(x0));

float(f(x0));

float(f(x1));
```

```
2.164179104

5.625

0.8079451262
```

Wir sehen, dass $f(x_1)$, mit x_1 = 2,164179104, jetzt näher bei Null ist als vorher mit x_0 = 2.5. In Fig. 3.9-5 sehen wir erneut den Graphen von f(x), jetzt mit der Tangente im Punkt $(x_1,f(x_1))$.

- ```
  tang:=f'(x1)*(x-x1)+f(x1):
  t2:=plot::Function2d(tang(x),x =1.5..2.5):
  plot(g,t2)
  ```
  ```

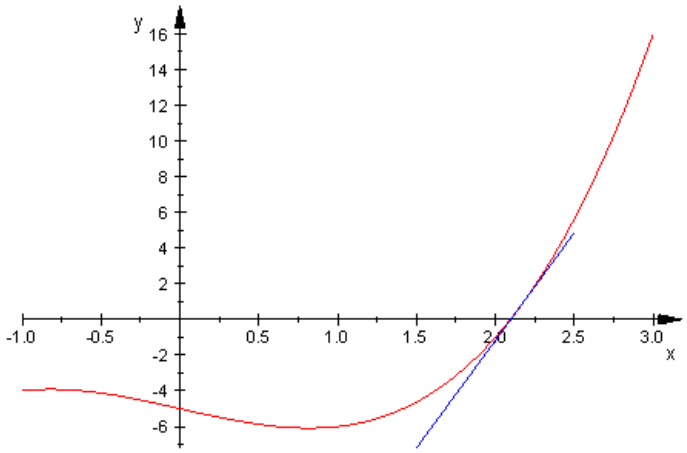


Fig. 3.9-5

Der nächste Wert, x_2 , ist noch näher am gesuchten Wert als x_1:

- **x2:=float(x1-f(x1)/f'(x1));**

  float(f(x2));

2.097135356

0.02888172122

Nun bauen wir Gl. (14) in ein Programm ein, das -ohne Graphen zu zeichnen- n Werte der Folge $\{x_i\}$ mit 25 Dezimalstellen berechnet. Als Anfangswert benutzen wir $x_0 = 3$.

- **f:=x->x^3-2*x-5:**

```
DIGITS:=25:
n:=7:
x[0]:=3:
for i from 0 to n do //n= Zahl der Iterationen

x[i+1]:=x[i]-f(x[i])/f'(x[i]);
print(float(x[i]),float(f(x[i])));

end_for;
```

In der ersten Spalte stehen die Elemente x_i , in der zweiten die Werte der Wallis-Funktion.

```
                                3.0, 16.0

                             2.36, 3.424256

  2.12719678015881649080822 4, 0.37109984624719729714211 12

 2.09513603693363411505338 1, 0.00652662595368705080876 5816

                               2.09455167382426773029167 9,
                            0.00000214614314427154840597 1071

                               2.09455148154234740613090 2,
                        0.00000000000023232140320477175703 06866

 2.09455148154232659148238 7, 2.72239072958398752900292 9e-27

 2.09455148154232659148238 7, 3.73828946945087929769236 6e-55
```

Mit demselben Programm und mit demselben Anfangswert $x_0 = 3$ können wir schnell die Nullstellen der folgenden Gleichungen finden:

1. $x - \sin x - 0{,}5 = 0$ (1,4973003...)
2. $e^{-x} + x/5 - 1 = 0$ (4,9651...)

Die andere Methode ist nach dem österreichischen Mathematiker **Bolzano** (1781-1848) benannt. Diese Methode, auch **Bisektionsverfahren** genannt, besteht darin, ein Intervall [a,b] zu suchen, das eine Lösung der Gleichung f(x) = 0 enthält, und es immer weiter zu reduzieren, bis die gewünschte Genauigkeit erreicht ist. Die Verkürzung der Intervalle besteht in einer fort-währenden Halbierung des jeweils letzten Intervalls.

Um ein Programm für die Bolzano-Methode zu schreiben, wählen wir zwei vernünftige Werte für a und b. Der Mittelpunkt des Intervalls [a,b] ist gegeben durch x = (a+b)/2, d.h. x ist das arithmetische Mittel von a und b.

f(a) ist der Funktionswert in x = a. Wenn f(x), mit x = (a+b)/2, Null sein sollte, dann wäre auch $f(a)\cdot f(x) = 0$, und x wäre die gesuchte Lösung.

Normalerweise wird x = (a+b)/2 noch nicht die gesuchte Lösung x_0 sein.
Wenn x_0 **links** von x liegt, so ist **f(a)·f(x) < 0** und wir setzen die Suche nur im Intervall [a,x] fort, d.h. wir setzen b:=x.

Dann teilen wir dieses Intervall in zwei gleich große Teilintervalle.

Wenn x_0 **rechts** von x liegt, so ist **f(a)·f(x) > 0** und wir fahren mit der Suche in [x,b], d.h. wir setzen a:=x. Dann teilen wir dieses Intervall in zwei gleich große Teilintervalle.
Wenn wir vorhaben, eine Lösung mit der Genauigkeit $|x_0 - x| < \varepsilon$ zu erhalten, so haben wir wenigstens N Unterteilungen auszuführen, wobei wir für N die folgende Beziehung haben

$$N > (\log(b-a)-\log(\varepsilon))/\log(2). \qquad (15)$$

Beispiel: Wenn [4,6] das Anfangsintervall [a,b] ist, so müssen wir wenigsten 11 Iterationen ausführen, um $|x_0 - x| <= b - a < 10^{-3}$ zu erhalten, d.h. mit drei gültigen Dezimalstellen. Tatsächlich müssen es 12 Iterationen sein.

Das bedeutet, dass die Konvergenz des Bisektionsverfahrens sehr langsam ist, vor allem, wenn das Anfangsintervall sehr groß und ε sehr klein ist.
Die Zahl der Iterationen ist normalerweise sehr hoch, wenn man eine gute Näherungslösung erhalten will.

Um ein MUPAD-Programm zu schreiben, müssen wir unsere Strategie ein wenig ändern, denn es scheint, dass MUPAD bei solchen Bedingungen Schwierigkeiten hat, die > oder < bei einem Vergleich enthalten. Wir müssen dann <> benutzen, was eine geringe Abänderung des Algorithmus bedeutet.
Wenn f(a) und f(x) verschiedene Vorzeichen haben, dann muss die Wurzel zwischen a und x liegen. In diesem Fall setzen wir das Verfahren mit demselben a-Wert fort und ersetzen b durch x, d.h. b := x.
Wenn f(a) und f(x) gleiche Vorzeichen haben, behalten wir den b-Wert als rechte Grenze bei und benutzen x als neue linke Intervallgrenze.

Ein Intervall mit $f(a_0) = f(b_0)$ muss von Anfang an ausgeschlossen werden:

```
if sign(f(a[0]))= sign(f(b[0]))then

print(Unquoted,"Das Intervall ist unbrauchbar!")
```

Was ich eben ausgeführt habe, ist in dem folgenden Programm berücksichtigt worden.

Ausgedruckt werden für jeden Iterationsschritt die linke Intervallgrenze, das arithmetische Mittel von a und b, dann b und die Länge des Intervalls $[x_0, x]$. Am Ende des Programms steht der Code zum Zeichnen der Intervalle.

```
reset()://Bolzano (Bisektionsverfahren)

          f:= x-> exp(-x)+x/5-1:
          a[0]:= 4: b[0]:=6:
          DIGITS:=25:
          eps:=0.001:
          N:=12:

          if sign(f(a[0]))= sign(f(b[0]))then
          print(Unquoted,"Das Intervall ist unbrauchbar!")

          else

          for i from 1 to N do
          x[i]:=(a[i-1]+b[i-1])/2:

          if sign(f(a[i-1]))<> sign(f(x[i]))
          then b[i]:=x[i];a[i]:=a[i-1]:

          else

          a[i]:=x[i]:b[i]:=b[i-1]:

          end_if:

          dist:=float(abs(a[i]-b[i])):
          print(float(a[i-1]),float(x[i]),float(b[i-
          1]),dist);

          if dist < eps then break

            end_if:
          end_for:
```

```
//Zeichnung der Intervalle
for k from 0 to 10 do
l[k]:=plot::Line2d([a[k],k],[b[k],k]):
end_for:
plot(l[k] $ k=0..10)// $ ist ein Operator für die
Indizes
/*Die Intervalle werden von unten nach oben
gezeichnet. Das nächste Programm zeichnet von
oben nach unten.*/
end_if:
```

4.0, 5.0, 6.0, 1.0

4.0, 4.5, 5.0, 0.5

4.5, 4.75, 5.0, 0.25

4.75, 4.875, 5.0, 0.125

4.875, 4.9375, 5.0, 0.0625

4.9375, 4.96875, 5.0, 0.03125

4.9375, 4.953125, 4.96875, 0.015625

4.953125, 4.9609375, 4.96875, 0.0078125

4.9609375, 4.96484375, 4.96875, 0.00390625

4.96484375, 4.966796875, 4.96875, 0.001953125

4.96484375, 4.9658203125, 4.966796875, 0.0009765625

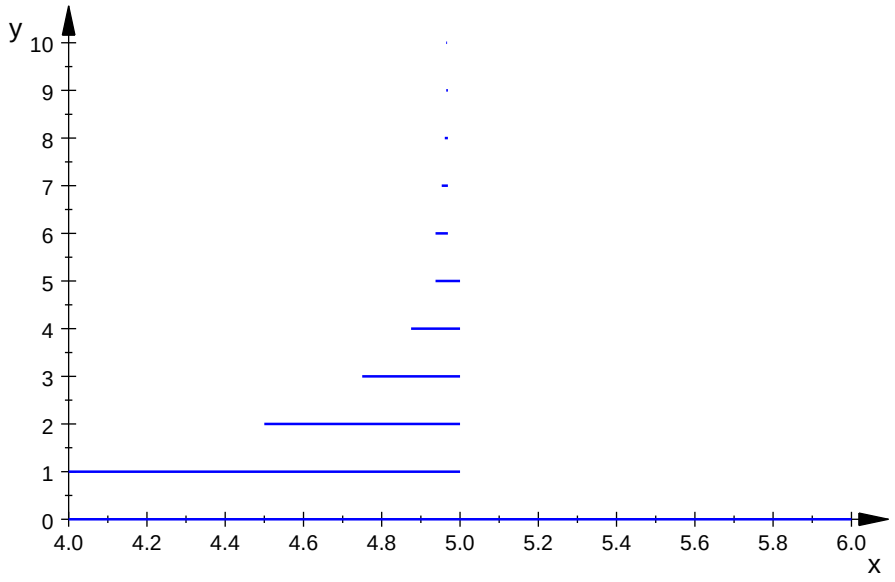


Fig.3.9-6

Der Wert `4.9658203125` hat eine Genauigkeit von nur 3 Dezimalen: 4.965

```
//Zeichnung der Intervalle von oben nach unten


for k from 0 to 5 do

l[k]:=plot::Line2d([a[k],5-k],[b[k],5-
k],Color=RGB::Red):

end_for:

plot(l[k] $ k=0..5)
end_if:
```

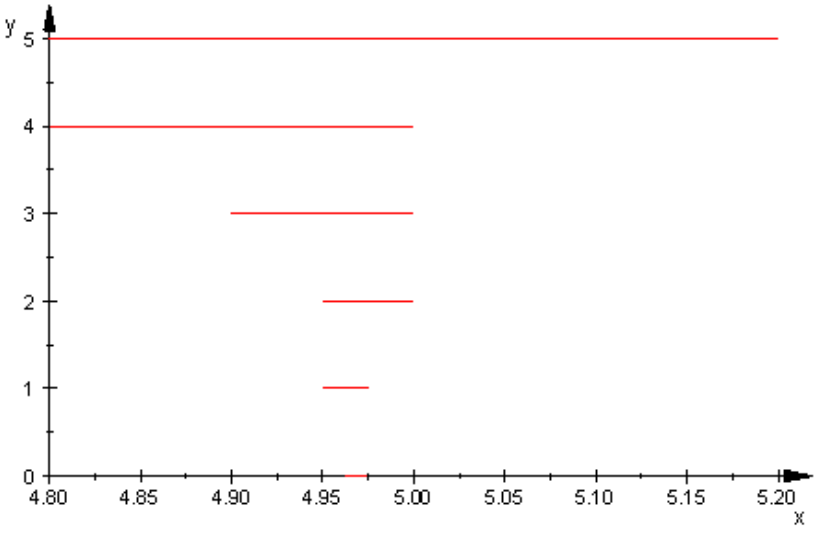


Fig. 3.9-7

Kai Gehrs und Vera Verspohl benutzen in *Analysis mit MuPAD*, SciFace Software, eine Intervallliste zusammen mit mit der Funktion **`plot::Group2d`** mit der man den Graphen einer Gruppe graphischer Objekte zeichnen kann. Um diese Liste zu bilden, wird die Anweisung **`append`** (*hinzufügen* von Elementen in eine Liste) benutzt. Mit Hilfe der Funktion **`op`** holen wir die Elemente aus der Liste **`intervalle`** .

```
intervalle :=[]://Leere Liste zum Start
for j from 0 to N do
intervalle:=append(intervalle,plot::Line2d([a[j],j],[b[j],j
])):

 //Die zu zeichnenden Intervalle werden in die Liste
"intervalle" aufgenommen.
end_for:
inter:=plot::Group2d(op(intervalle)):

//Gruppe der Elemente der Liste "intervalle"

plot(inter):
```

Die Anpassung an unser Programm kann man folgendermaßen machen:

```
reset():
f:= x-> exp(-x)+x/5-1:
a[0]:= 4: b[0]:=6:
p:=plot::Function2d(5*f(x),x = 4..6,
Color=RGB::Green):
//f wird der groesseren Klarheit wegen
vergroessert

DIGITS:=25:
eps:=0.001:
N:=10:
if sign(f(a[0]))= sign(f(b[0]))then
print(Unquoted,"Intervall unbrauchbar!")
else

for i from 1 to N do
x[i]:=(a[i-1]+b[i-1])/2:
if sign(f(a[i-1]))<> sign(f(x[i]))
then b[i]:=x[i];a[i]:=a[i-1]:

else

a[i]:=x[i]:b[i]:=b[i-1]:

end_if:
end_for:

//Hier kommt der neue Code:

intervalle :=[]://leere Liste
for j from 0 to N do
intervalle:=append(intervalle,
plot::Line2d([a[j],j],[b[j],j])):
//zu zeichnende Intervalle kommen in die Liste
"intervalle"
end_for:
inter:=plot::Group2d(op(intervalle)):

//Gruppe der Elemente in der Liste"intervalle"

plot(p,inter):
// 5*f wird zusammen mit den Intervallen
gezeichnet
end_if:
```

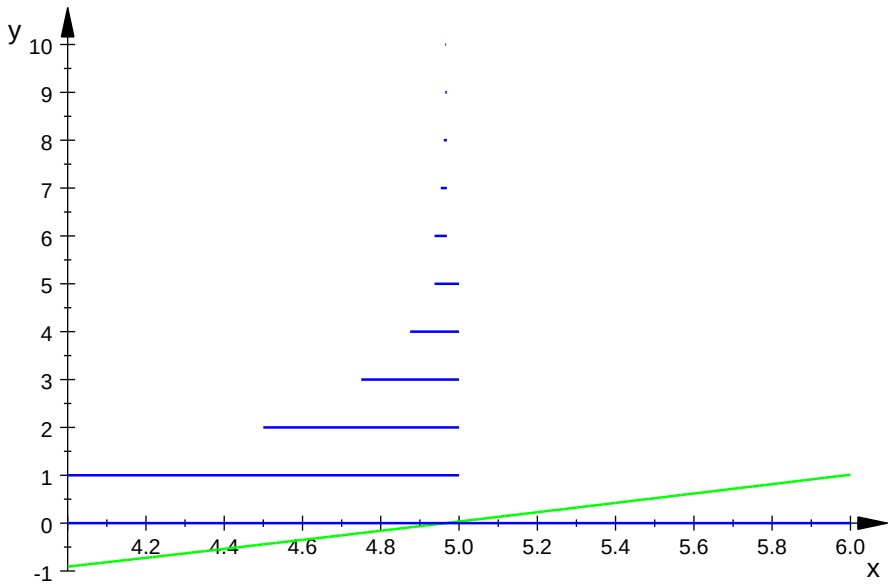
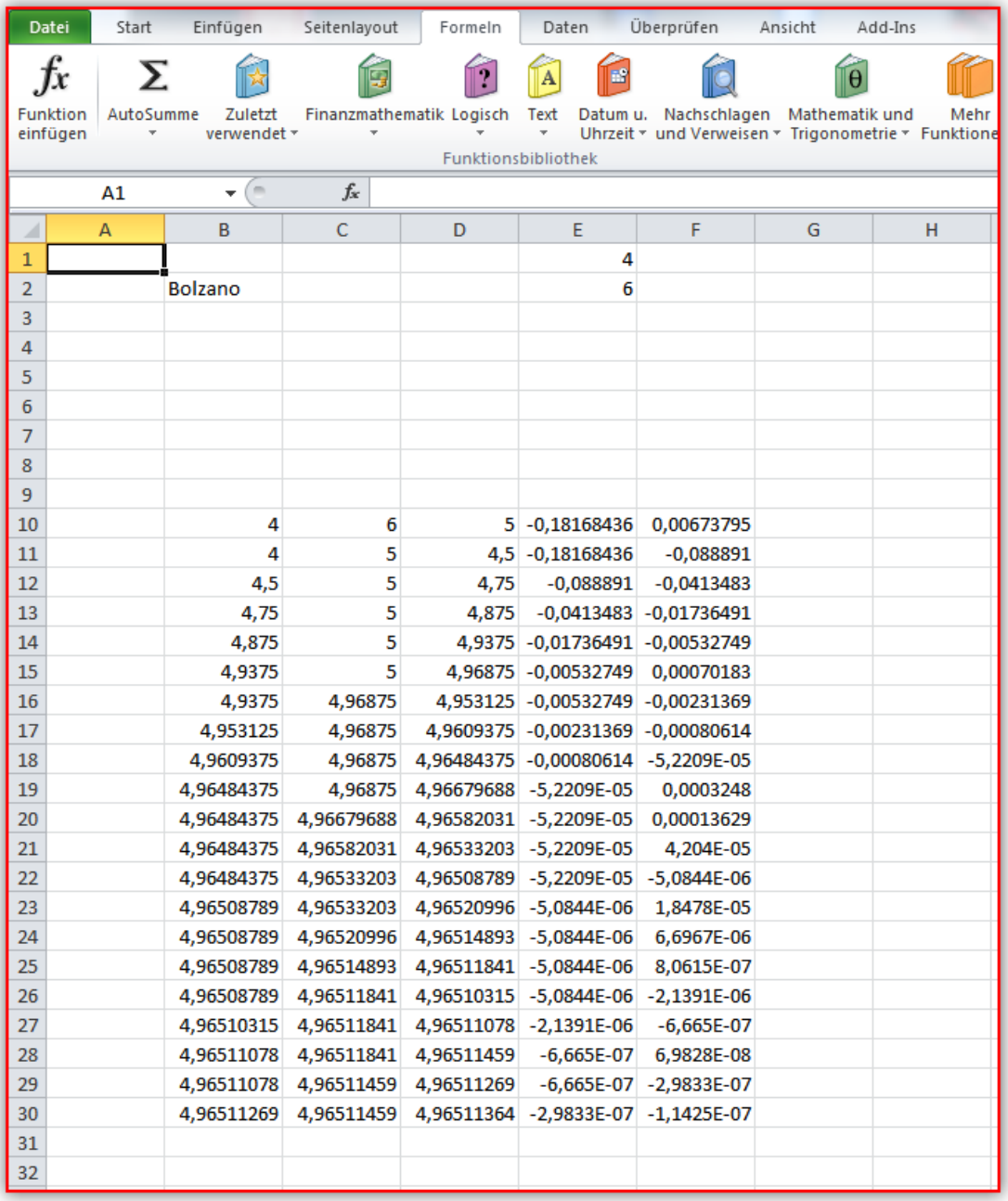


Fig. 3.9-8

Es ist übrigens sehr einfach, den Bisektionsalgorithmus in **Excel** zu implementieren. Mit den Beziehungen **f(a)·f(x) < 0** und **f(a)·f(x) > 0** können wir folgendermaßen vorgehen:

1. Die Spalten B, C, D auf 16 Stellen vergroessern
2. Schreibe den Wert von a in E1 und den von b in E2.
3. B10 :**+E$1**; C10: **+E$2**; D10: +**(B10+C10)/2**
   E10: @**EXP(-B10)+B10/5-1** (= f(a))
   F10: @**EXP(-D10)+D10/5-1** (= f(x))
4. Wenn wir 30 Iterationen machen wollen, müssen wir D10, E10, F10 bis D30, E30, F30 kopieren
5. B11: @**WENN(E10*F10>0;D10;B10** (a:=x)
   C11: @**WENN(E10*F10<0;D10;C10** (b:=x)
6. Kopiere B11, C11 bis B30..C30

Resultate:



	A	B	C	D	E	F	G	H
1					4			
2		Bolzano			6			
3								
4								
5								
6								
7								
8								
9								
10		4	6	5	-0,18168436	0,00673795		
11		4	5	4,5	-0,18168436	-0,088891		
12		4,5	5	4,75	-0,088891	-0,0413483		
13		4,75	5	4,875	-0,0413483	-0,01736491		
14		4,875	5	4,9375	-0,01736491	-0,00532749		
15		4,9375	5	4,96875	-0,00532749	0,00070183		
16		4,9375	4,96875	4,953125	-0,00532749	-0,00231369		
17		4,953125	4,96875	4,9609375	-0,00231369	-0,00080614		
18		4,9609375	4,96875	4,96484375	-0,00080614	-5,2209E-05		
19		4,96484375	4,96875	4,96679688	-5,2209E-05	0,0003248		
20		4,96484375	4,96679688	4,96582031	-5,2209E-05	0,00013629		
21		4,96484375	4,96582031	4,96533203	-5,2209E-05	4,204E-05		
22		4,96484375	4,96533203	4,96508789	-5,2209E-05	-5,0844E-06		
23		4,96508789	4,96533203	4,96520996	-5,0844E-06	1,8478E-05		
24		4,96508789	4,96520996	4,96514893	-5,0844E-06	6,6967E-06		
25		4,96508789	4,96514893	4,96511841	-5,0844E-06	8,0615E-07		
26		4,96508789	4,96511841	4,96510315	-5,0844E-06	-2,1391E-06		
27		4,96510315	4,96511841	4,96511078	-2,1391E-06	-6,665E-07		
28		4,96511078	4,96511841	4,96511459	-6,665E-07	6,9828E-08		
29		4,96511078	4,96511459	4,96511269	-6,665E-07	-2,9833E-07		
30		4,96511269	4,96511459	4,96511364	-2,9833E-07	-1,1425E-07		
31								
32								

Fig. 3.9-9