Numerik 1

Numerische Lösung von Differentialgleichungen

Die Lösung einer Differentialgleichung ist eine *Funktion,* die der Diff.Gl. über einem gewissen offenen Intervall genügt. Eine *gewöhnliche* Diff.Gl. hat die allgemeine Form

$$\varphi(x,y,y',y'',y''',\ldots, d^ny/dx^n) = 0 \quad (1)$$

Gl.(1) ist von n-ter Ordnung und hat nur eine unabhängige Variable x.

Die Funktion y = F(x) ist eine Lösung von (1), wenn sie n mal differenzierbar ist und Gl. (1) genügt.

Die Gleichungen y':= dy/dx = x+y; $y''+(1-y^2)y' + y = 0$ sind Beispiele gewöhnlicher Diff.Gln. Eine Diff.Gl. φ(x,y(x),dy/dx) = 0 kann i. Allg. folgendermaßen geschrieben werden

$$dy/dx = y' = f(x,y) \quad (2)$$

Die gewöhnlichen Diff.Gln. haben verschiedene Lösungen. Um eine bestimmte Lösung auszuwählen, sind zusätzliche Informationen nötig, normalerweise n für eine Diff.Gl. n-ter Ordnung. Wenn alle n Zusatzbedingungen für ein bestimmtes $x= x_o$ gegeben sind, so haben wir ein *Anfangswertproblem* (**AWP**). Falls sich die n Zusatzbedingungen auf mehr als ein x beziehen, so haben wir ein *Randwertproblem* (**RWP**).

Der Graph der Lösung einer Diff.Gl. heißt *Lösungskurve*. (Eine Lösungskurve ist auch eine Integralkurve.)

Es gibt graphische und numerische Methoden, mit denen man sich eine Vorstellung über die Form der Lösung bilden kann und auf die man sich beziehen kann, falls keine explizite Lösungsformel existiert oder falls diese zu kompliziert ist, um nützlich zu sein.

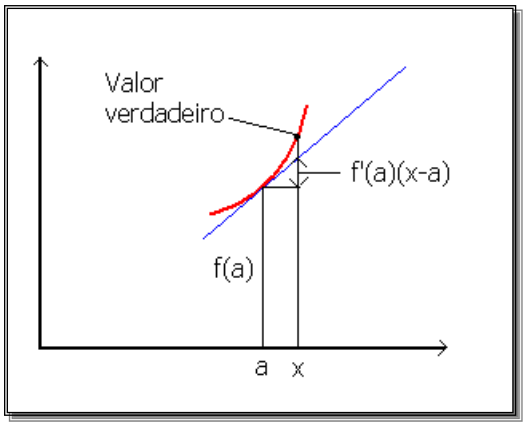
Für MuPAD gibt es ein NoteBook mit dem Titel *Das Euler-Cauchy-Verfahren*... von Agnes Havasi und Kai Gehrs, in dem die Autoren das graphische Verfahren ausführlich erklären.

Für Anwendungen in der Physik benötigen wir numerische Methoden, die eine exakte Lösung mit praktisch jeder Genauigkeit annähern. Das einfache *Euler*-Verfahren hatten wir schon in 3.8.3 besprochen. Wir werden hier nochmals darauf eingehen.

Eulersche Methode für y' = f(x,y)

Wir betrachten zunächst eine gewöhnliche Diff.Gl erster Ordnung $y' = f(x,y)$ mit einer Anfangsbedingung $y(x_o) = y_0$. Unsere Absicht ist es, numerisch eine Lösung y(x) zu finden, die die Diff.Gl. und die Anfangsbedingung erfüllt. Das nach *Euler* (1707-1783) benannte Verfahren stützt sich auf die Idee, die Werte von y(x) durch die Werte der in x_0 an die Lösungskurve gezeichneten Tangente anzunähern. (Euler beschrieb seine Methode 1768 in "*Institutiones Calculi Integralis*, sectio secunda, caput VII").

In der Abbildung sehen wir den Graphen einer Funktion y = f(x) in der vergrößerten Umgebung des Punktes mit x = a zusammen mit der Tangente an die glatte Kurve in diesem Punkt. ("Valor verdadeiro" = wahrer (Funktions)-wert.)



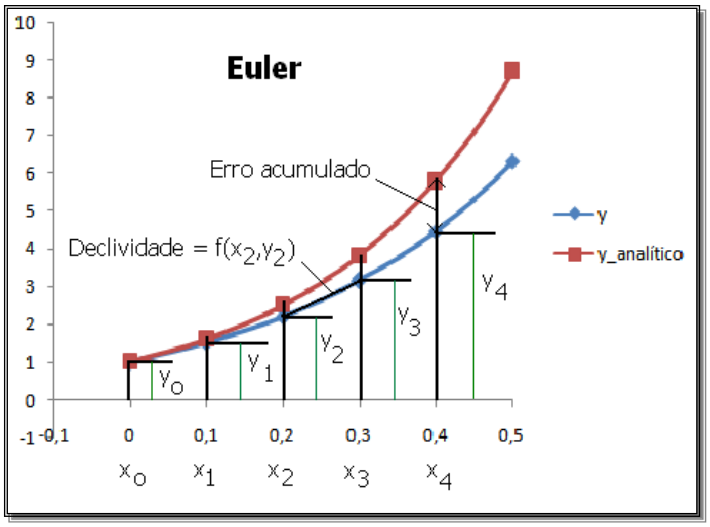
In der Nähe des Berührungspunktes sind die wahren Funktionswerte kaum von denen der Tangente zu unterscheiden. Die Tangente geht durch den Punkt (a,f(a)) und hat die Steigung f´(a). Ihre Gleichung lautet

$$y = f(a) + f'(a)(x-a) \quad (3)$$

Wir können näherungsweise die Werte von f durch die y-Werte der Tangente ersetzen. Für x-Werte in der Nähe von a können wir für den wahren Funktionswert f(x) der Funktion f an der Stelle x schreiben

$$f(x) \approx f(a) + f'(a)(x-a) \quad (4)$$

In der folgenden Abbildung ist der Zusammenhang noch ausführlicher dargestellt.



Die Euler-Methode stützt sich also auf die Annahme, dass die Tangente an die Lösungskurve von y' = f(x,y) mit $y(x_0) = y_0$ in $(x_i,y(x_i))$ diese über dem Intervall $[x_i, x_{i+1}]$ annähert. Die Steigung ("Declividade") der Lösungskurve in $(x_i, y(x_i))$ ist $y'(x_i) = f(x_i,y(x_i))$ und die Gleichung der Tangente an die Lösungskurve in $(x_i,y(x_i))$ ist

$$y = y(x_i) + f(x_i, y(x_i))(x-x_i) \quad (5) \text{ (Tangente)}$$

Wir schreiben $x = x_{i+1} = x_i + h$ und erhalten

$$y_{i+1} = y(x_i) + h\cdot f(x_i, y(x_i)) \quad (6)$$

Hierin ist h die Schrittweite und y_{i+1} der Wert von y *bis zur Tangente* im Punkt x_{i+1}. y_{i+1} nehmen wir als Näherung für $y(x_{i+1})$.

Da $y(x_0)=y_0$ gegeben ist, können wir (6) mit i = 0 benutzen, um y_1 zu berechnen

$$y_1= y(x_0) + h\cdot f(x_0, y(x_0)) = y_0 + h\cdot f(x_0, y_0) \quad (7)$$

Jetzt setzen wir i =1, und Gl. (6) geht über in

$$y_2 = y(x_1) + h\cdot f(x_1, y(x_1)) \quad (8)$$

Aber diese Gleichung ist unbrauchbar, da wir $y(x_1)$ nicht kennen. (Nur $y(x_0)$ ist bekannt, und wir nennen es y_0.) Hier nun kommt die *Näherung*:

Wir substituieren den Wert $y(x_1)$, den wir nicht kennen, durch den Wert y_1, der nur bis zur Tangente reicht und der in der Abbildung deutlich kleiner ist als y(x) in x_1.

$$y(x_2) \approx y_2 = y_1 + h\cdot f(x_1,y_1)$$

Im nächsten Schritt ersetzen wir $y(x_2)$ durch y_2:

$$y_3 = y_2 + h \cdot f(x_2,y_2)$$

Wir können den Prozess wiederholen bis der gewünsche x-Wert erreicht ist. Allgemein beginnt die Euler-Methode mit dem gegebenen Wert $y(x_0) = y_0$ und erzeugt $y_1, y_2, \ldots y_n$ mifhilfe der Rekursionsformel

$$\mathbf{y_{i+1} = y_i + h \cdot f(x_i,y_i)}, \quad 0 <= i <= n - 1 \quad (9)$$

(Es handelt sich um eine Rekursionsformel und nicht um eine Iterationsformel, denn im Falle einer Iteration, die ein Spezialfall der Rekursion ist, sucht man i.Allg. den Grenzwert eines Prozesses. Aber der Sprachgebrauch unterscheidet meist nicht so streng zwischen den beiden Prozessen.) Die Zahlen y_1, y_2, y_3 usw. sind Näherungswerte von $y(x_1), y(x_2), y(x_3)$ usw.

Beispiel:

y' = 1-x+4y, y(0) = 1; exakte (analytische) Lösung: $y(x) = x/4 - 3/16 + (19/16) \cdot e^{4x}$

Lösung nach Euler: (h = 0.1)

f(x,y) = 1 - x +4y
$y_1 = y_0 + h \cdot f(x_0,y_0) = 1 + 0.1 \cdot (1-0+4 \cdot 1) = 1+0,5 =$ **1,5** ; $x = x_1 = h = 0.1$
exakter Wert: y(0,1) = 1,609041828
$y_2 = y_1 + h \cdot f(x_1,y_1) = 1,5 + 0,1 \cdot (1-0,1+4 \cdot 1,5) = 1,5 +0,69 =$ **2,19**; $x = x_2 = 0,2$
exakter Wert: y(0,2)= 2.505329853
$y_3 = y_2 + h \cdot f(x_2,y_2) = 2,19 + 0,956 =$ **3,146**; $x = x_3 = 0,3$
exakter Wert: y(0,3) = 3,830138846

- ```
  x:=0:y:=1:h:=0.1://Euler für y'=f(x,y)
  DIGITS:=10:
  for i from 0 to 5 do
  f:=1-x+4*y://f(xi,yi)
  y:=y+h*f://yi+1
  x:=x+h://xi+1
  //xi:=x-h://vorheriger Wert xi
  //yi:=y-h*f:// vorheriger Wert yi

       F0:=x/4-3/16+(19/16)*exp(4*x)://anal. Lösung
       print(i,x,y,F0):
       end_for:
  ```
  ```

i x_{i+1} y_{i+1} $y(x_{i+1})$

0, 0.1, 1.5, 1.609041828

1, 0.2, 2.19, 2.505329853

2, 0.3, 3.146, 3.830138846

3, 0.4, 4.4744, 5.794226004

4, 0.5, 6.32416, 8.712004117

5, 0.6, 8.903824, 13.05252195

Verbesserte Euler-Verfahren (Methode von *Heun*)

Das Euler-Verfahren $\mathbf{y_{i+1} = y_i + h \cdot f(x_i,y_i)}$ benutzt immer die Steigung der Tangente an die Lösungskurve am Anfang des Intervalls $[x_i , x_{i+1}]$ und nimmt an, dass diese Steigung über dem ganzen Intervall konstant ist. Aber normalerweise sehen wir, dass sich die Tangentensteigung an die Lösungskurve in dem Intervall $[x_i, x_{i+1}]$ ändert. Man kann eine Verbesserung des Verfahrens erhalten, wenn man die Tangentensteigung in der Intervallmitte oder sogar einen Mittelwert aus mehreren Steigungen in $[x_i , x_{i+1}]$ benutzt. So verwendet die nach ***Heun*** (1859-1929) benannte Methode das Mittel aus den Steigungen an den Intervallgrenzen. Im Übrigen werden aber dieselben Schritte ausgeführt, die uns oben zu den Formeln von (5) bis (9) führten.

Wir kehren also wieder zu Gl. (6) zurück und ersetzen die Steigung $(f(x_i, y(x_i))$ durch das Mittel

$$m_i = (f(x_i,y(x_i)) + f(x_{i+1},y(x_{i+1}))/2 \quad (10)$$

Gl. (6) lautet jetzt $y_{i+1} = y(x_i) + h\,(f(x_i,y(x_i)) + f(x_{i+1},y(x_{i+1})))/2$ und ist ein Näherung für $y(x_{i+1})$. Wie vorher benutzen wir y_i als Näherung für $y(x_i)$.

Aber i.Allg. ist $y(x_{i+1})$ nicht bekannt, und wir werden es durch die Näherung $y(x_{i+1}) \approx y_{i+1} = y_i + h\, f(x_i,y_i)$ ersetzen.

Die Rekursionsformel der *verbesserten Eulermethode* oder der *Methode von Heun* lautet schließlich

$$y_{i+1} = y_i + h/2 \cdot (f(x_i,y_i) + f(x_{i+1}, y_i + h\, f(x_i,y_i))) \quad (11)$$

Für das praktische Rechnen ist die Einführung der folgenden Ausdrücke sinnvoll:

$$k_{1i} = f(x_i, y_i) \quad (12)$$
$$k_{2i} = f(x_i+h,\ y_i + hk_{1i}) \quad (13)$$
$$y_{i+1} = y_i + h\,(k_{1i} + k_{2i})/2 \quad (14)$$

Die Methode von *Heun* erzeugt wesentlich bessere Ergebnisse als das einfache *Eulerverfahren*, vor allem eine Variante des *verbesserten Eulerverfahrens*, die in jedem Punkt x_i zunächst das Ergebnis durch *innere Iteration* verbessert ehe zum nächsten Punkt x_{i+1} weitergegangen wird. Im Programm "Heun2" wird dies ausgeführt.

(Die *Heunmethode* hat einen globalen Rundungsfehler von der Größenordnung $O(h^2)$, und man kann feststellen, dass man durch Halbierung der Schrittweite h eine Fehlerreduzierung um den Faktor ¼ erhält. Das Symbol $O(h^2)$ will besagen, dass die *Heunmethode* mit einer *Taylorentwicklung* bis auf Terme der Ordnung h^2 übereinstimmt.)

```
reset()://einfache Heun-Methode
Heun:=proc(h,schritte)
local i,n,x,y,dy,f,k1,k2,F0;
begin
x:=0:y:=1:
DIGITS:=6:
f:=(x,y)->-2*y+x^3*exp(-2*x):
tabelle:=array(1..schritte,1..4):
for i from 1 to schritte do
k1:=f(x,y):
x:=x+h:
k2:=f(x,y+h*k1):
dy:=h*(k1+k2)/2:
y:=y+dy:
F0:=exp(-2*x)*(x^4+4)/4://analytische Lösung
tabelle[i,1]:=x:
tabelle[i,2]:=y:
tabelle[i,3]:=F0:
tabelle[i,4]:=F0-y:
end_for:
return(tabelle)
end_proc:

     Heun(0.1,5)
```

Ergebnisse:

X	Y	exakt	F0-y
0.1,	0.820041,	0.818751,	-0.00128972
0.2,	0.672734,	0.670588,	-0.00214627
0.3,	0.552598,	0.549923,	-0.00267466
0.4,	0.455161,	0.452205,	-0.00295597
0.5,	0.376681,	0.373628,	-0.00305369

Das Programm ist als Prozedur konzipiert. Die Deklaration der lokalen Variablen ist nicht unbedingt nötig. Dennoch verlangt eine guter Programmierstil den Gebrauch lokaler Variablen, um Konflikte zu vermeiden, die dann auftreten können, wenn man eine Prozedur als Modul in einem anderen Programm verwenden will.

Das *modifizierte Euler*-Verfahren benutzt die Tangente als angenäherte Steigung in der *Mitte* des Intervalls. Man gelangt so zu der folgenden Rekursionsformel (Formel des Mittelpunktes oder Halbschrittverfahren)

$$y_{i+1} = y_i + h\, f(x_i+h/2,\ y_i + f(x_i,y_i)\cdot h/2) \qquad (15)$$

(Die Gleichung der Tangente im Punkt (x_0,y_0) ist bei *Euler* $y = y_0 + f(x_0,y_0)(x-x_0)$. Die *Mittelpunktsmethode* ersetzt die Steigung $f(x_0,y_0)$ durch die im Zentrum des Intervalls, d.h. durch die Tangentensteigung im Punkt $(x_0+h/2, y(x_m))$. Der Wert $y(x_m)$ ist nicht bekannt, kann aber angenähert werden durch den Mittelwert $(y_0 + y_1)/2$. Den unbekannten Wert von y_1 berechnen wir mithilfe der *Euler*schen Gleichung $y_1 = y_0 + h\cdot f(x_0,y_0)$. Die modifizierte Form für das erste Intervall lautet

$$y_1 = y_0 + h\, f(x_m,\ (y_0 + (y_0+h\cdot f(x_0,y_0))) = y_0 + h\cdot f(x_0+h/2,\ y_0+f(x_0,y_{,0})\cdot h/2)$$

und daraus ergibt sich dann (15).)

Im folgenden Programm finden wir diesen Algorithmus zusammen mit einigen Anmerkungen.

```
reset():://Mittelpunktmethode
Midpoint:=proc(h,schritte)
local i,n,x,y,dy,f,k1,k2,ym,xm,F0;
begin
x:=0:y:=1:
DIGITS:=8:
f:=(x,y)->-2*y+x^3*exp(-2*x):
tabelle:=array(1..schritte,1..4):
for i from 1 to schritte do
k1:=f(x,y)://Steigunng in xi
ym:=y+h*k1/2://y_Mitte berechnet mit Euler
xm:=x+h/2://x_Mitte
k2:=f(xm,ym)://Steigung in der Mitte(Näherung)
x:=xm+h/2://xi+1
y:=y+h*k2://yi+1
F0:=exp(-2*x)*(x^4+4)/4://Exakte Lösung
tabelle[i,1]:=x:
tabelle[i,2]:=y:
tabelle[i,3]:=F0:
tabelle[i,4]:=F0-y:
end_for:
return(tabelle)
end_proc:
Midpoint(0.1,5)
```

```
+-                                                    -+
|  0.1, 0.82001131, 0.81875122, -0.0012600891          |
|                                                      |
|  0.2, 0.67265111, 0.67058817, -0.0020629394          |
|                                                      |
|  0.3, 0.55246799, 0.54992298, -0.0025450119          |
|                                                      |
|  0.4, 0.45500468, 0.45220467, -0.0028000139          |
|                                                      |
|  0.5, 0.37652114, 0.37362756, -0.0028935785          |
+-                                                    -+
```

Wir werden jetzt die *Euler*sche Methode mit der *Heun*schen verbinden, um den folgenden recht effektiven Algorithmus zu erhalten:

- ```
  reset():Euler_Heun Methode
  Euler_Heun:=proc(h,schritte)
  local i,t,y,v,f,F0;
  begin
  t:=1:y:=2:v:=0:
  DIGITS:=8:
  f:=(t,y)->-v/t+4*y/t^2:
  tabelle:=array(1..schritte,1..4):
  for i from 1 to schritte do
  va:=v:
  a:=f(t,y):
  v:=va+h*a:
  y:=y+(h/2)*(va+v):
  t:=t+h:
  F0:=t^2+1/t^2:
  tabelle[i,1]:=t:
  tabelle[i,2]:=v:
  tabelle[i,3]:=y:
  tabelle[i,4]:=F0:
  end_for:
  return(tabelle)
  end_proc:
  Euler_Heun(0.05,20)
  ```

```
+-                                                  -+
|  1.05,     0.4,       2.01,   2.0095295  |
|                                             |
|   1.1, 0.74557823, 2.0386395, 2.0364463  |
|                                             |
|  1.15,  1.0486535, 2.0834952, 2.0786437  |
|                                             |
|   1.2,  1.3181442, 2.1426652, 2.1344444  |
|                                             |
|  1.25,  1.5608139, 2.2146391,   2.2025   |
|                                             |
|   1.3,  1.7818552, 2.2982059,  2.281716  |
|                                             |
|  1.35,  1.9852993, 2.3923847, 2.3711968  |
|                                             |
|   1.4,  2.1743085, 2.4963749, 2.4702041  |
|                                             |
|  1.45,  2.3513868, 2.6095173, 2.5781243  |
|                                             |
|   1.5,  2.5185344, 2.7312653, 2.6944444  |
|                                             |
|  1.55,  2.6773624, 2.8611628, 2.8187331  |
|                                             |
|   1.6,  2.829178,  2.9988263,  2.950625  |
|                                             |
|  1.65,  2.9750495,  3.143932, 3.0898095  |
|                                             |
|   1.7,  3.1158557, 3.2962046, 3.2360208  |
```
```

```
|                                                  |
|  1.75,  3.2523239, 3.4554091, 3.3890306          |
|                                                  |
|   1.8,  3.3850597, 3.6213437,  3.548642          |
|                                                  |
|  1.85,   3.51457,  3.7938344, 3.7146841          |
|                                                  |
|   1.9,  3.6412813, 3.9727307, 3.8870083          |
|                                                  |
|  1.95,  3.7655539, 4.1579016, 4.0654849          |
|                                                  |
|   2.0,  3.8876943, 4.3492328,    4.25            |
+-                                                -+
```

Das folgende Programm zeigt den *Heun*schen Algorithmus mit *innerer Iteration*:

- ```
  reset():Heun-Methode mit innerer Iteration (Heun-2)
  Heun:=proc(h,schritte)
  local i,j,n,x,y,ya,ye,dy,f,k1,k2,F0;
  begin
  x:=0:y:=1:
  DIGITS:=6:
  f:=(x,y)->2*(x^2+y):
  tabelle:=array(1..schritte,1..3):
  for i from 1 to schritte do
  ya:=y:// alter y-Wert
  k1:=f(x,y):
  x:=x+h:
  ye:=y+h*k1:
  //print(ye):Euler-Wert
  for j from 1 to 4 do//4 innere Iterationen
  k2:=f(x,ye):
  dy:=h*(k1+k2)/2:
  y:=ya+dy:
  //print(y):
  ye:=y:
  end_for:
  ya:=y:
  F0:=1.5*exp(2*x)-x^2-x-0.5://analytische Lösung
  //print(i,x,y,F0):
  tabelle[i,1]:=x:
  tabelle[i,2]:=y:
  tabelle[i,3]:=F0://zeigt x,y,exakter Wert
  end_for:
  return(tabelle)
  end_proc:
  Heun(0.1,5)
  ```
  ```

Ergebnisse:

```
+-                          -+
|  0.1, 1.22333,  1.2221     |
|                            |
|  0.2, 1.50073, 1.49774     |
|                            |
|  0.3, 1.84867, 1.84318     |
|                            |
|  0.4, 2.28726, 2.27831     |
|                            |
|  0.5, 2.84109, 2.82742     |
+-                          -+
```

Das Runge-Kutta-Verfahren

Selbst das verbesserte Verfahren von *Heun* kann nicht konkurrieren mit dem Verfahren von *Runge* und *Kutta*. Das *Runge-Kutta*-Verfahren zählt zu den beliebtesten Methoden bei der Lösung von Anfangswertproblemen. (C. Runge 1856-1927, W. Kutta 1867-1944). Im Gegensatz zur *Heunschen* Methode wird beim R-K-Verfahren die Funktion f(x,y) nicht nur zweimal, sondern viermal berechnet (an vier Punkten des Intervalls $[x_i,x_{i+1}]$), und man reduziert damit den globalen Rundungsfehler auf $O(h^4)$. (Das R-K-Verfahren, das wir benutzen werden, ist auch bekannt als RK vierter Ordnung. "Heun" war von zweiter Ordnung und "Euler" von erster.)

Ich werde den RK-Algorithmus so darstellen, als wäre er eine einfache Abwandlung des *Euler*-Verfahrens.

Algorithmus von Runge-Kutta vierter Ordnung für y' = f(t,y).

Da wir in den Anwendungen meistens die Zeit als unabhängige Variable haben, benutzen wir t statt x und v statt y´. Das Symbol <v> bezeichnet den Mittelwert von vier Ableitungen (Geschwindigkeiten) des RK-Verfahrens.

$$t_{n+1} = t_n + h$$

$$y_{n+1} = y_n + h\langle v\rangle,$$

mit

$$\langle v\rangle := (v_1+2v_2+2v_3+v_4)/6 \qquad (16)$$

Die vier Ableitungen berechnen wir nach dem folgenden Schema:

$$v_1 := f(t,y)$$

$$v_2 := f(t + h/2,\ y + v_1 \cdot h/2)$$

$$v_3 := f(t + h/2,\ y + v_2 \cdot h/2)$$

$$v_4 := f(t + h,\ y + v_3 \cdot h) \qquad (17)$$

(Weiter unten werden wir sehen, dass die Verallgemeinerung dieser Methode auf Diffgln. zweiter Ordnung y'' = f(t,x,y') sehr einfach ist.)

Wir werden gleich sehen, wie einfach die Implementation der Schemata (16) und (17) in MuPAD ist.

- ```
  reset():://Runge-Kutta y'(t,y)
  t:=0:y:=0:
  h:=0.1:
  DIGITS:=5:
  f:=(t,y)->1/(1+t^2)-2*y^2:
  for i from 1 to 10 do
  v1:=f(t,y):
  v2:=f(t+h/2,y+v1*h/2):
  v3:=f(t+h/2,y+v2*h/2):
  v4:=f(t+h,y+v3*h):
  y:=y+h*(v1+2*v2+2*v3+v4)/6:
  t:=t+h:
  F0:=t/(1+t^2):
  print(t,y,F0,F0-y):
  end_for:
  ```

```
                    0.1, 0.099009, 0.09901, 4.1895e-7

                    0.2, 0.19231, 0.19231, 9.5851e-7

                    0.3, 0.27523, 0.27523, 1.7105e-6

                    0.4, 0.34482, 0.34483, 2.6161e-6

                        0.5, 0.4, 0.4, 3.5041e-6

                    0.6, 0.44117, 0.44118, 4.1971e-6

                     0.7, 0.46979, 0.4698, 4.5926e-6

                     0.8, 0.4878, 0.4878, 4.6811e-6

                    0.9, 0.49723, 0.49724, 4.5182e-6

                        1.0, 0.5, 0.5, 4.1847e-6
```
```

Die Ergebnisse zeigen, dass die Fehler nur von der Ordung 10^{-6} sind. Bei *Euler* fanden wir Fehler der Größenordnung 10^{-2}. Die *Euler*-Methode hat aber einen großen didaktischen Wert und ist hilfreich beim Studium genauerer Verfahren.