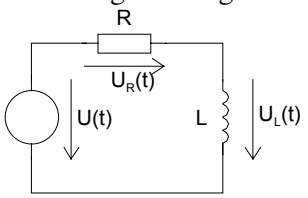
8 Gewöhnliche Differenzialgleichungen

Viele Vorgänge in der Technik werden durch Differenzialgleichungen (DGL) beschrieben. Differenzialgleichungen sind Gleichungen die eine oder mehrere Ableitungen einer unbekannten Funktion beinhalten.

Beispiele:

1. Im folgend gezeigten elektrischen Netzwerk gilt die Maschengleichung:

$$U(t) = I(t)R + \frac{dI(t)}{dt}L$$



2. Bei einer elastischen Feder ist die Rückstellkraft (und damit die Beschleunigung) proportional zur Auslenkung. Mit $D>0$ als Federkonstante schreiben wir nun die Gleichung:

$$s''(t) = -D\,s(t)$$

In der Mathematik ist es üblicher y für die abhängige Variable und x oder t für die unabhängige Variable zu verwenden. So erhalten wir für obige Differenzialgleichung die allgemeine Form:

$$y''+D\,y = 0 \qquad \text{resp.: } y'L + yR = U$$

Wie bereits aus der Analysis bekannt, ist die (analytische) Lösung von DGL je nach Art mühsam bis schwierig oder sogar unmöglich. Einzig für lineare DGL existiert eine klare Methode, die sicher zu einer eindeutigen Lösung führt.

Für andere Typen existieren fallweise Lösungsansätze. Die Lösbarkeit ist aber in keiner Weise die sichergestellt, da auch hier das Grundprinzip zur Lösung, das Wegschaffen der Differenziale durch Integration ist.

Die Numerik bietet eine ganze Reihe von Verfahren zur Lösung von Differenzialgleichungen an. Dabei wird mit einer bestimmten Genauigkeit **punktweise die Lösungsfunktion errechnet**.

Wir zeigen nachfolgend verschiedene Methoden der numerischen Lösung von gewöhnlichen Differenzialgleichungen:

- Taylorreihenverfahren: - Verfahren von Euler
  - Taylorreihenverfahren höherer Ordnung
- Runge-Kutta-Verfahren: 2. Ordnung (Trapezverfahren)
  4. Ordnung

Die beiden ersten Verfahren bilden wiederum das theoretische Fundament für die praktisch benutzten Runge-Kutta-Verfahren.

8.1 Klassifikation von Differentialgleichungen

In der Analysis ist es üblich Differenzialgleichung zur methodischen Lösung zuerst nach Ordnung, Grad, Homogenität, partielle/ gewöhnliche DGL, etc. sehr genau zu untersuchen., da das Lösungsverfahren entscheidend vom Typ der DGL abhängt.

Praktisch alle numerische Verfahren lösen grundsätzlich Gleichungen des Typs $y' = f(x,y)$, also DGL und DGL-Systeme erster Ordnung.
DGL höherer Ordnung können durch Ordnungsreduktion in ein DGL-System erster Ordnung reduziert werden.

8.2 Anfangswertprobleme: Analytische und numerische Lösung

Eine Lösung der Differenzialgleichung ist eine Funktion, die eine Lösung der Gleichung darstellt. Nachfolgend sind einige Beispiele für Differenzialgleichungen aufgeführt mit ihren Lösungen.

Gleichung	Lösung
$x' - x = e^t$	$x(t) = te^t + ce^t$
$x'' + 9x = 0$	$x(t) = c_1 \sin 3t + c_2 \cos 3t$
$y' + \frac{1}{2y} = 0$	$y(x) = \sqrt{c - x}$

In allen Beispielen verkörpert *c* eine (wählbare) Integrationskonstante. Durch Bestimmen der Konstanten erhalten wir aus der allgemeinen, mehrdeutigen Lösung eine eindeutige Lösung der Differenzialgleichung.

In der Praxis geschieht die Bestimmung der Konstanten indem die gegebenen Anfangswerte (Randwerte) eingesetzt werden.

So ist ein typisches Anfangswertproblem definiert als:

$$\begin{cases} y' = f(y, y(x)) \\ y(a) \quad \text{ist gegeben} \end{cases} \quad \text{resp.} \quad \begin{cases} x' = f(t, x(t)) \\ x(a) \quad \text{ist gegeben} \end{cases}$$

Zu bestimmen ist hier *x(t),* resp. *y(x)* als Lösungsfunktion, die auch dem vorgegebenen Anfangswert *x(a)*, resp. *y(a)* genügt.
Wird *x* als abhängige Variable verstanden und *t* als Zeitfaktor, so soll die Lösungsfunktion *x(t)* für alle *t* vor und nach *a* das zugehörige *x* bestimmen.

Einige Beispiele von Anfangswertproblemen mit den zugehörigen Lösungen:

$y' - y = 1$	$y(0) = 0$	$y(x) = e^x - 1$
$y' = 6x - 1$	$y(1) = 6$	$y(x) = 3x^2 - x + 4$
$y' = \frac{x}{y+1}$	$y(0) = 0$	$y(x) = \sqrt{x^2 + 1} - 1$

Obwohl sehr viele Lösungsverfahren zur analytischen Lösung von DGL existieren, ist die analytische Lösbarkeit nur auf wenige spezielle DGL beschränkt. Diese lösbaren Fälle liefern dann als Resultat eine formale Lösung.

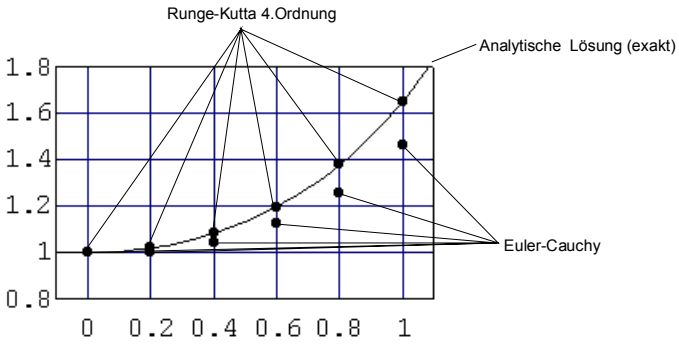
Kann keine formale Lösung erarbeitet werden, so werden numerische Verfahren eingesetzt.

Als Beispiel zeigen wir die analytische und numerische Lösung einer DGL als Anfangswertproblem für das Intervall [0,1]:

$y' = y\,x \qquad y(0) = 1$

analytische Lösung:

$\Rightarrow y(x) = e^{\frac{x^2}{2}}$



Numerische Lösungen:

x	Euler-Cauchy	Runge-Kutta
0.0	1.0	1.00000000
0.2	1.0	1.02020133
0.4	1.04	1.08328699
0.6	1.1232	1.19721701
0.8	1.257984	1.37712642
1.0	1.45926144	1.64871668

8.3 Taylorreihen-Verfahren

Diese Methode zeigt eine grundsätzliche Idee zur numerischen Lösung einer Differenzialgleichung indem die Lösungsfunktion x durch eine Taylorreihe dargestellt wird:

$$y(x+h) = y(x) + hy'(x) + \frac{1}{2!}h^2 y''(x) + \frac{1}{3!}h^3 y'''(x) + \cdots + \frac{1}{n!}h^n y^{(n)}(x) + \cdots \tag{8.1}$$

Für eine numerische Betrachtung wird die Taylorreihe nach n+1 Gliedern abgeschnitten. Wir können dann *x(t+h)* recht präzis und einfach bestimmen, wenn die Schrittweite *h* klein ist und *x(t), x'(t), x''(t),...x$^{(n)}$(t)* bekannt sind. Je nachdem wie viele Glieder *n* der Taylorreihe benutzt werden, spricht man vom Taylorreihen-Verfahren der Ordnung *n*.

8.3.1 Euler-Verfahren

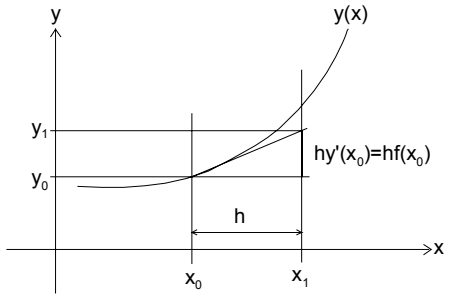
Das Euler-Verfahren ist ein Taylorreihenverfahren 1. Ordnung. Es ist das einfachste numerische Verfahren zur Lösung von Differenzialgleichungen. Leider ist es auch das ungenauste Verfahren. Es ist vom Ansatz her aber so grundlegend, dass es zum Pflichtstoff gehört.

Das Verfahren liefert Näherungslösungen für das Anfangswertproblem der Form:

$$\begin{cases} y' = f(x, y(x)) \\ y(a) = y_a \end{cases}$$

Euler-Cauchy Näherung (8.2)

Geometrische Betrachtung
Aus einem gegebenen Anfangswert $y(x_0)=y_0$ wird sukzessiv eine Näherung für $y(x_0+h)=y(x_1)$ unter Verwendung des Tangentenanstieges errechnet:



Wir erhalten also:

$$y_1 = y_0 + h\,y'(x_0) \tag{8.3}$$

Und setzen $y'(x_0) = f(x_0, y_0)$ein und erhalten als den ersten Punkt der Lösung als Näherung:

$$y_1 = y_0 + h\,f(x_0, y_0) \tag{8.4}$$

Dieses Verfahren wird nun für n Schritte wiederholt und man erhält allgemein die Rekursionsformel:

$$y_{i+1} = y_i + h\,f(x_i, y_i) \tag{8.5}$$

Mathematische Betrachtung
Das Euler-Verfahren ist ein sog. Taylorreihen-Verfahren erster Ordnung . Näherungsweise kann im Intervall $[a,b]$ die Lösungsfunktion bestimmen werden, indem die beiden ersten Terme der Taylorreihe benutzt werden:

$$y(x+h) = \sum_{k=0}^{\infty} \frac{y^{(k)}(x)h^k}{k!} = y(x) + hy'(x) + \frac{h^2}{f}y''(x) + \cdots \tag{8.6}$$

$$y(x+h) \approx y(x) + hy'(x) \tag{8.7}$$

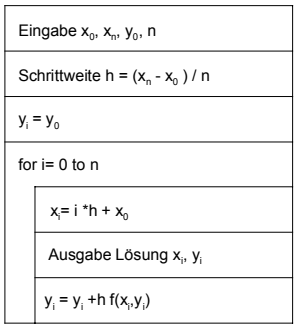
Unter Zugrundelegung, dass $y'(x) = f(x, y(x))$ erhält man die Formel

$$y(x+h) \approx y(x) + h\,f(x, y(x)) \tag{8.8}$$

um in n äquidistanten Schritten der Breite $h = \frac{b-a}{n}$ von $x_0=a$ bis $x_n=b$ zu laufen und so punktweise eine Näherungslösung zu berechnen.

8.3.2 Programmentwurf für das Euler-Verfahren

Gemäss den formalen Zusammenhängen aus dem vorherigen Kapitel entwerfen wir den Programmablauf mit einem Nassi-Shneiderman Diagramm:



Wir entwerfen das Programm konkret für unsere Test-DGL $y' = x\,y$ mit $y(0) = 1$. Dazu definieren wir die Ableitung $y'=f(x,y)$ als eigenständige Funktion:

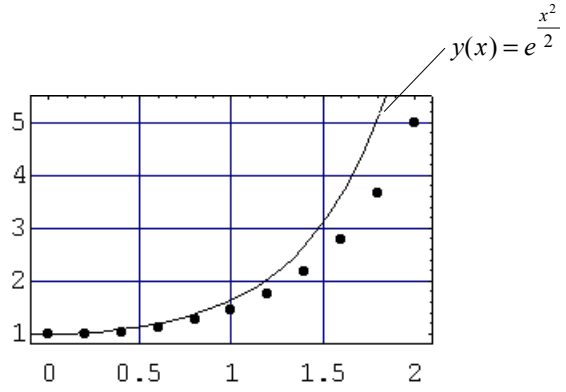
```
double f (double x, double y)
{
   return x * y;
}
```

Die eigenständige Definition ausserhalb des Hauptprogrammes hat den Vorteil der besseren Gliederung und leichteren Anpassbarkeit, falls eine andere DGL gelöst werden muss. Die Funktion `f` kann dann neu definiert werden, ohne dass man sich um den Rest kümmern muss.

Ein Testlauf ergibt für *n = 10* folgende Werte:

```
Numerische Lösung der Differenzialgleichung mit dem Euler-Cauchy Verfahren:
Intervallbeginn x0: 0
Intervallende xn: 2
Anfangswert f(x0)=y0: 1
Anzahl Schritte: 10
Lösungsfunktion:
x= 0.000        y=  1.00000000
x= 0.200        y=  1.00000000
x= 0.400        y=  1.04000000
x= 0.600        y=  1.12320000
x= 0.800        y=  1.25798400
x= 1.000        y=  1.45926144
x= 1.200        y=  1.75111373
x= 1.400        y=  2.17138102
x= 1.600        y=  2.77936771
x= 1.800        y=  3.66876538
x= 2.000        y=  4.98952091
```

Ein grafischer Vergleich mit der exakten Lösung:



Das erhaltene Resultat kann nicht als befriedigend bezeichnet werden. für x=2 erhalten wir 4.9895..

anstatt des exakten Wertes von 7.3890.

Diese Abweichungen sind für die Praxis zu gross. Auch wenn wir kleinere Schrittweiten wählen, nimmt die Genauigkeit nur langsam zu, da das Verfahren bei Verdoppelung der Anzahl Schritte den Fehler nur etwa halbiert.

n	y_n	Fehler
5	3.71652864	3.6725
10	4.98952091	2.3995
20	5.97322600	1.4158
40	6.61146382	0.7775

Eine konkrete Implementierung in C für das Euler-Cauchy-Verfahren ist:

```
/* Programm EULER                                                    File: EULER.C
   Numerische Lösung von Differenzialgleichungen nach dem Euler-Cauchy-Verfahren.
   Das Verfahren bestimmt die numerische Lösung der DGL der Art
      y' = f(x,y)
   wobei f(x,y) als Funktion vorgegeben wird.

   Arbeitsweise:
   Ausgehend vom Startwert werden n Schritten die Loesungsfunktion punktweise bestimmt
   und tabelliert ausgegeben.

   Autor: Gerhard Krucker
   Datum: 7.6.1995
   Sprache: MS Visual-C V1.5  (QuickWin App.)
*/
#include <stdio.h>

/* Definition der Ableitung nach der Aufgabenstellung:
    y'=f(x,y)=x y
*/
double f(double x, double y)
{
   return x * y;
}

int main ()
{ double x0;    /* Intervallstart */
  double xn;    /* Intervallende */
  double y0;    /* Anfangswert y0 bei x0 */
  int n;        /* Anzahl Schritte */
  double h;     /* Schrittweite */
  double xi,yi; /* Errechnete Werte im Schritt #i */
  int i;

  printf("Numerische Lösung der Differenzialgleichung mit dem Euler-Cauchy Verfahren:\n");
  printf("Intervallbeginn x0: "); scanf("%lg",&x0);
  printf("Intervallende xn: "); scanf("%lg",&xn);
  printf("Anfangswert f(x0)=y0: "); scanf("%lg",&y0);
  printf("Anzahl Schritte: "); scanf("%d",&n);
  printf("Lösungsfunktion: \n");

  h = (xn- x0) / n;
  yi = y0;

  for (i=0; i <= n ; i++)
   {      xi = x0 + h * i;   /* Aktuelles x im Schritt #i */
     printf("x=%6.3f\ty=%12.8f\n",xi,yi );

     yi = yi + h * f(xi, yi);
   }
  return 0;
}
```

8.3.3 Taylorreihenverfahren höherer Ordnung

Wir zeigen anhand eines Beispieles wie man mit einem Taylorreihenansatz höherer Ordnung einen Ansatz zur Lösung des Anfangswertproblems bestimmt:

$$y' = 1 + y^2 + x^3 \qquad x \in [1,2]$$
$$y(1) = -4$$

Wenn nun die DGL mehrmals nach x differenziert wird, erhalten wir folgendes System:

$$y' = 1 + y^2 + x^3$$
$$y'' = 2yy' + 3x^2$$
$$y''' = 2yy'' + 2y'y' + 6x$$
$$y^{(4)} = 2yy''' + 6y'y'' + 6$$

Vorausgesetzt, dass x und $y(x)$ bekannt sind, können nacheinander $y'(x)$, $y''(x)$, $y'''(x)$ und $y^{(4)}(x)$ entwickelt werden. Somit ist es möglich diese 4 Ableitungen in der Taylorreihe einzusetzen:

$$y(x+h) \approx y(x) + hy'(x) + \frac{h^2}{2}y''(x) + \frac{h^3}{6}y'''(x) + \frac{h^4}{24}y^{(4)}(x) \qquad \text{wobei} \quad h = \frac{x_n - x_0}{n} \tag{8.9}$$

Mit der Anfangsbedingung $y(1)$=-4 haben wir einen Startwert und wir wählen ein n=100, das die Schrittweite definiert. Nun kann durch Einsetzen der Ableitungen in die Formel (8.14) eine Näherung für $y(x_0+h)$ bestimmt werden.

Dieser Vorgang wird nun für $y(x_0+2h)$, etc., wiederholt, analog dem Verfahren erster Ordnung. Dazu ein Entwurf des Verfahrens in Pseudocode:

```
PROGRAM TAYLOR
double PARAMETER x0 ← 1, xn ← 2, y ← -4
int PARAMETER n ← 100
int k
double h, x, y, y', y'', y''',y(4)
h ← (xn-x0) / n
t ← x0
output 0, x, y
FOR k=1 TO n DO
   y' ← 1 + y2 + x3
   y'' ← 2yy' + 3x2
   y''' ← 2yy'' 2(y')2 +6x
   y(4) ← 2yy'''+6y'y''+6
   y ← y + h[ y' h/2[y'' + h/3[ y''' + h/4[y(4)]]]]
   t ← x0 + kh
  output k, x, y
END DO
END PROGRAM TAYLOR
```

Hier werden das Intervall $[x_0,x_n]$ indem die Lösung vom Startpunkt x_0 entwickelt werden soll, über Parameter übergeben. Dies erfolgt in der schlussendlichen Codierung über interaktive Eingabe, oder falls das Verfahren als Funktion codiert wird, mit einer entsprechenden Parameterübergabe. Ebenso wird mit der gewünschten Schrittweite n verfahren.

Als Resultat erhalten wir im Vergleich zum Euler-Verfahren eine wesentlich bessere Genauigkeit:

Euler	Taylorreihenverfahren 4.Ordnung
y(2)≈4.2358541	y(2)≈4.3712096

Im Vergleich zur analytischen Lösung sehen wir, dass die Taylorreihenverfahren die Lösung auf 5 Dezimalstellen genau liefert.

Trotz der guten Genauigkeit der Taylorreihenverfahren höherer Ordnung ist ein wesentlicher Nachteil, dass die entsprechenden Ableitungen y'', y''', etc. alle vorliegen müssen. Diese Forderung wird durch das nachfolgende Runge-Kutta-Verfahren umgangen.

8.3.4 Taylorreihenverfahren mit EXCEL angewandt

Die schrittweise Entwicklung der Lösungsfunktion kann problemlos mit handelsüblichen Tabellenkalkulationsprogrammen, wie EXCEL vorgenommen werden. Sicherlich sind diese Programme nicht die erste Wahl zur Lösung solcher Aufgaben, jedoch haben sie einige Vorteile für die schulmässige Lösung:

- Sauberes Arbeitsblatt mit allen verwendeten Werten
- Alle Zwischenschritte aufgeführt
- Speichern der Arbeit und späteres Wiederaufgreifen möglich
- Grafische Darstellungen einfach möglich
- Auswirkungen von Änderungen der Vorgaben können sehr einfach durchgespielt werden.

8.3.4.1 Beispiel: Taylorverfahren erster und vierter Ordnung

Zu bestimmen ist die Lösungsfunktion für das Anfangswertproblem:

$$y' = xy \qquad x \in [0,2]$$
$$y(0) = 1 \qquad h = 0.2$$

Wir lösen unsere Standard DGL indem wir zuerst die Vorbereitungsarbeiten leisten:

1. Die DGL muss explizit nach *y'(x,y)* aufgelöst werden (hier gegeben).

2. Für Taylorreihenverfahren der Ordnung *n* müssen die Ableitungen bis (*n-1*) bestimmt werden.

3. Formulieren der Iterationsvorschrift für $y_{i+1}(x_i, y_i)$.

4. Das Arbeitsblatt in EXCEL wird vorbereitet indem es beschriftet wird und die Vorgaben aus der Aufgabenstellung eingetragen werden.

5. Im Arbeitsblatt wird das Gerüst für die Wertetabelle erzeugt:
   Dazu werden die Indizes *i* für jeden Rechenschritt aufgeführt und die entsprechenden Argumente x_i tabellarisch aufgeführt.

Bestimmen der Ableitungen $y'',y''',y^{(4)}$:

$$y''= y + xy'$$
$$y'''= 2y'+xy''$$
$$y^{(4)} = 3y''+xy'''$$

Hinweis: Beachten Sie, dass hier *y* eine Funktion von *x* ist und dementsprechend nach der Kettenregel abgeleitet werden muss (-> Ableiten von impliziten Funktionen).

Iterationsvorschriften:

Taylorverfahren erster Ordnung:

$$f(x_i, y_i) := y'(x_i, y_i)$$
$$y(0) = 1$$
$$y_{i+1} = y_i + h \cdot f(x_i, y_i) = y_i + h \cdot x_i y_i$$

Taylorverfahren vierter Ordnung:

$$f(x_i, y_i) := y'(x_i, y_i)$$
$$y(0) = 1$$
$$y_{i+1} = y_i + \sum_{k=1}^{4} \frac{h^k}{k!} f^{(k)}(x_i, y_i) = y_i + h \cdot x_i y_i + \frac{h^2}{2} \cdot y_i'' + \frac{h^3}{6} y_i''' + \frac{h^4}{24} y_i^{(4)}$$

Arbeitsblatt erstellen und aufbauen der Wertetabelle:

Taylorverfahren erster und vierter Ordnung:

Aufgabe: Anfangswertproblem

$y' = xy$ $\quad x \in [0,2]$

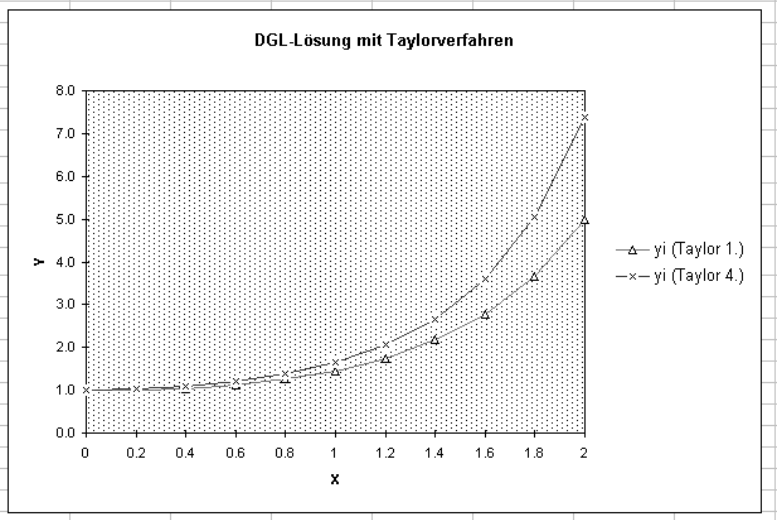
$y(0) = 1$ $\quad h = 0.2$

Vorgaben:

x_0 =	0	h =	0.2
x_n =	2		
y(0) =	1		

Wertetabellen:

i	x_i	y_i'	y_i (Taylor 1.)
0	0	0.0000	1.0000
1	0.2	0.2000	1.0000
2	0.4	0.4160	1.0400
3	0.6	0.6739	1.1232
4	0.8	1.0064	1.2580
5	1	1.4593	1.4593
6	1.2	2.1013	1.7511
7	1.4	3.0399	2.1714
8	1.6	4.4470	2.7794
9	1.8	6.6038	3.6688
10	2		4.9895



i	x_i	y_i'	y_i''	y_i'''	$y_i^{(4)}$	y_i (Taylor 4.)
0	0	0.00000	1.00000	0.00000	3.00000	1.00000
1	0.2	0.20404	1.06101	0.62028	3.30708	1.02020
2	0.4	0.43331	1.25660	1.36926	4.31750	1.08328
3	0.6	0.71831	1.62817	2.41352	6.33262	1.19718
4	0.8	1.10164	2.25836	4.00997	9.98305	1.37705
5	1	1.64856	3.29711	6.59422	16.48556	1.64856
6	1.2	2.46492	5.01201	10.94425	28.16912	2.05410
7	1.4	3.72931	7.88483	18.49739	49.55086	2.66380
8	1.6	5.75251	12.79934	31.98398	89.57240	3.59532
9	1.8	9.09077	21.41381	56.72640	166.34897	5.05043
10	2					7.38358

Taylorverfahren | Tabelle2 | Tabelle3 | Tabelle4 | Tabelle5 | Tabelle6 | Tabelle7 | Tabelle8

Zur Hilfe stehen in den wichtigen Zellen folgende Einträge:

```
B15: =$B$9+$D$9 * A15
C15: =B15 * D15
D15: =$B$11
D16: =D15 + $D$9 * B15 * D15

B28: =$B$9+$D$9 * A28
C28: =B28 * G28
D28: =G28 + B28 * C28
E28: =2 * C28 + B28 * D28
F28: =3 * D28 + B28 * E28
G28: =$B$11
G29: =G28 + $D$9 * C28 + $D$9^2 / 2 * D28 + $D$9^3 / 6 * E28 + $D$9^4 / 24 * F28
```

Wir sehen, dass das Taylorreihenverfahren vierter Ordnung sehr präzise Resultate liefert. Allerdings ist die vorgängige Bestimmung der Ableitungsfunktionen aufwendig.

8.3.5 Ansetzen der DGL für das numerische Verfahren

Aus der praktischen Aufgabenstellung werden zur Lösung folgende Schritte vorgenommen:

1. Bestimmen der analytischen DGL. In einem elektrische System wird beispielsweise ein Knoten-/Maschenansatz durchgeführt. Die DGL muss explizit in der Form

   $$y' = \frac{dy}{dx} = \ldots.$$

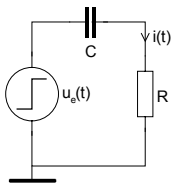
   geschrieben werden können. DGL's welche nicht explizit geschrieben werden können sind mit gezeigten Verfahren nicht lösbar. Ebenso sind nur DSGL erster Ordnung zulässig.

2. Bestimmen der Anfangswerte. Bei einem elektrischen System geht man beispielsweise davon aus, dass es hinreichend lange ausgeschaltet war und somit keine Restladungen in den Speicherelementen vorhanden sind. Andernfalls sind sie entsprechen zu verrechnen.

3. Wahl der Schrittweite h. Mindestens 10x kleiner als die kleinste relevante Zeitkonstante im System. Vorsicht: Ein kleines h bringt nicht zwangläfig ein besseres Resultat, vor bei der Euler-Methode.

4. Sukzessiv die Lösung vom Startwert aus berechnen.

Beispiel:
Bestimmen der zum System zugehörigen DGL und die ersten beiden Werte der Lösungsfunktion für den Maschenstrom mit einer Schrittweite h=0.02ms ab dem Zeitpunkt t0=0 mit dem Verfahren von Euler



Vorgaben:
C=1uF
R =100Ω
$u_e(t)$=0V bis t_0, nachher 1V

Lösung:
In der Masche gilt:

$$u_e(t) = u_C(t) + u_R(t)$$

Unter Benutzung der integro-differenziellen Beziehungen zwischen Strom und Spannung im Kondensator $u_C(t) = \frac{1}{C}\int i_C(t)dt$ wird:

$$u_e(t) = \frac{1}{C}\int i_C(t)dt + i_C(t)\cdot R$$

Die Integralgleichung wird durch Ableiten in eine Differenzialgleichung erster Ordnung übergeführt und explizit nach i(t) geschrieben:

$$\frac{du_e(t)}{dt} = \frac{1}{C}i_C(t) + R\frac{di_C(t)}{dt} \qquad \left| T = RC, i_C(t) = i(t) \right.$$
$$\frac{di(t)}{dt} = \frac{1}{T}\left(C\frac{du_e(t)}{dt} - i(t)\right)$$

Der Anfangswert ist offensichtlich 1/R. Wir begründen dies formal indem wird in die Gleichung für den Ansatz die Werte für t_0 einsetzen und die Gleichung auswerten:

$$1 = \underbrace{\frac{1}{C}\int_0^0 i_C(t)dt}_{=0} + i_C(t)\cdot R \qquad \Rightarrow i_C(t_0) = \frac{1}{R} = \frac{1}{100} = 10mA$$

Rechenschritte nach Euler:

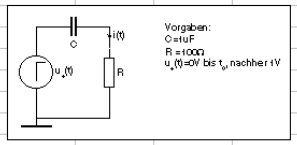
$i_C(0) = 0.01 \qquad \frac{du_e(t)}{dt} = 0$ (weil Sprungfunktion)

$$i_C(2\cdot 10^{-5}) = i_C(0) - h\frac{1}{T}i_C(0) = i_C(0)\left(1 - \frac{h}{T}\right) = 0.01\left(1 - \frac{2\cdot 10^{-5}}{1\cdot 10^{-4}}\right) = 0.008$$

$$i_C(4\cdot 10^{-5}) = i_C(2\cdot 10^{-5}) - h\frac{1}{T}i_C(2\cdot 10^{-5}) = i_C(2\cdot 10^{-5})\left(1 - \frac{h}{T}\right) = 0.008\left(1 - \frac{2\cdot 10^{-5}}{1\cdot 10^{-4}}\right) = 0.0064$$

Mit einer EXCEL-Tabelle wird dies:

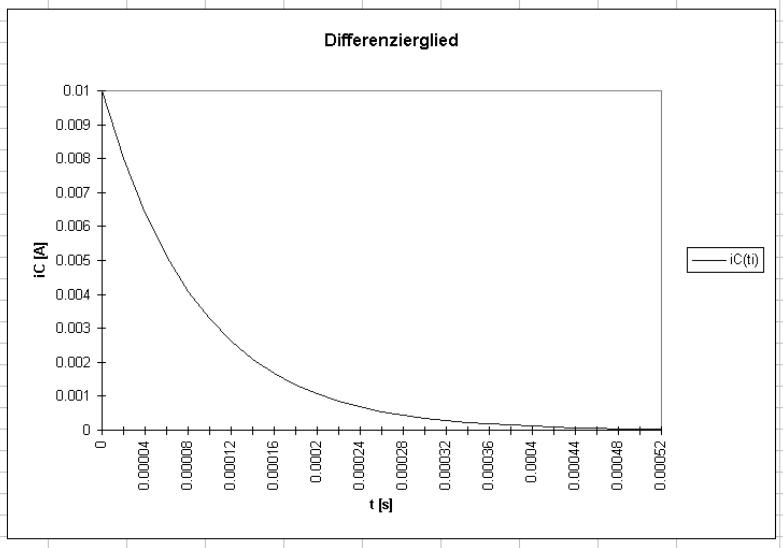
Aufgabe a):



R =	100
C =	1.00E-06
U_E =	1
t_0 =	0
h =	2.00E-05
$i_C(t_0)$ =	0.01
T =	0.0001

Rechenschritte nach Euler:

i	t_i	$i_C(t_i)$
0	0	0.01
1	0.00002	8.00E-03
2	0.00004	6.40E-03
3	0.00006	5.12E-03
4	0.00008	4.10E-03
5	0.0001	3.28E-03
6	0.00012	2.62E-03
7	0.00014	2.10E-03
8	0.00016	1.68E-03
9	0.00018	1.34E-03
10	0.0002	1.07E-03
11	0.00022	8.59E-04
12	0.00024	6.87E-04
13	0.00026	5.50E-04
14	0.00028	4.40E-04
15	0.0003	3.52E-04
16	0.00032	2.81E-04
17	0.00034	2.25E-04
18	0.00036	1.80E-04
19	0.00038	1.44E-04
20	0.0004	1.15E-04
21	0.00042	9.22E-05
22	0.00044	7.38E-05
23	0.00046	5.90E-05
24	0.00048	4.72E-05
25	0.0005	3.78E-05
26	0.00052	3.02E-05



8.4 Differenzialgleichungssysteme

Die numerischen Lösungsverfahren erbringen nur eine Lösung für einfache DGL erster Ordnung $y'=f(x,y)$. In vielen Anwendungen erscheinen aber anstatt einer Funktion $y(x)$ ein System von mehreren Funktionen y_1, y_2,... . Diese Funktionen sind dann über ein System von Differenzialgleichungen miteinander verknüpft:

Beispiel:

$$y_1' = y_1(y_2 - x) \qquad y_1(0) = y_2(0) = 1$$
$$y_2' = y_2 - \ln(y_1)$$

Die analytische Lösung ergibt:

$$y_1 = e^x \quad y_2 = 1 + x$$

Allgemein hat ein Differenzialgleichungssystem erster Ordnung die Form:

$$\begin{aligned} y_1' &= f_1(x, y_1, y_2, \cdots, y_n) \\ y_2' &= f_2(x, y_1, y_2, \cdots, y_n) \\ &\cdots \\ y_n' &= f_n(x, y_1, y_2, \cdots, y_n) \end{aligned} \qquad (8.10)$$

Wenden wir dafür die Vektorschreibweise

$$\begin{aligned}\underline{y'} &= (y_1', y_2', \cdots, y_n') \\ \underline{f} &= (f_1, f_2, \cdots, f_n) \\ \underline{y} &= (y_1, y_2, \cdots, y_n)\end{aligned} \tag{8.11}$$

an, so vereinfacht sich die Schreibweise für das Differenzialgleichungssystem zu:

$$\underline{y'} = \underline{f}(x, \underline{y}) \tag{8.12}$$

Die Lösung ist hierbei eine Vektorfunktion

$$\underline{y}(x) = (y_1(x), y_2(x), \cdots, y_n(x)) \tag{8.13}$$

Mit den Anfangsbedingungen $y_1(x_0), y_2(x_0), \cdots, y_n(x_0)$ genügt und dem Intervall $[x_0, x_n]$ stellt das System ein Anfangswertproblem erster Ordnung dar.

8.4.1 Lösung eines DGL Systems mit dem Verfahren von Euler

Die Lösung eines DGL-System erfolgt genau gleich wie bei einer einfachen DGL, nur dass alle Operationen auf einen Vektor angewandt werden. Das heisst in jedem Schritt werden mehrere Resultate berechnet. Das Vorgehen ist nicht spezifisch für das Euler-Verfahren, sondern gilt für alle DGL-Lösungsverfahren.

Beispiel:

$$\begin{aligned} y_1' &= y_1(y_2 - x) \qquad & y_1(0) = y_2(0) = 1 \\ y_2' &= y_2 - \ln(y_1) \end{aligned}$$

Der erste Schritt nach Euler liefert die Lösungen bei einer Schrittweite von h=0.1:

$$\begin{aligned} y_1(0.1) &= y_1(0) + hy'_1(0) = 1 + 0.1\big(1(1-0)\big) = 1.1 \\ y_2(0.1) &= y_2(0) + hy'_2(0) = 1 + 0.1\big(1 - \ln 1\big) = 1.1 \end{aligned}$$

8.5 Differentialgleichungen höherer Ordnung: Ordnungsreduktion

Alle bisher gezeigten numerischen Verfahren zur Lösung von Differenzialgleichungen lösen nur DGL erster Ordnung.

Sollen Differenzialgleichungen höherer Ordnung mit diesen Verfahren gelöst werden, so müssen sie zuerst in ein äquivalentes System von Differenzialgleichungen erster Ordnung umgeformt werden. Dieser Prozess wird als **Ordnungsreduktion** bezeichnet.

Das Verfahren der Ordnungsreduktion ist ein rezeptuales Vorgehen, dass eine DGL höherer Ordnung in ein System von DGL's erster Ordnung zerlegt. Dieses System kann mit einem numerischen Verfahren weiter bearbeitet werden.

Wir betrachten die Vorgehensweise an einem Beispiel:

$$y''' + \big(y'\big)^2 e^{y'} - xy = 0$$

Wesentlicher Punkt ist, dass eine neue Variable z eingeführt wird die zum Erzeugen des linearen

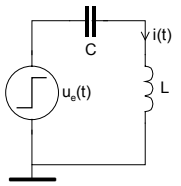
Gleichungssystems erster Ordnung verwendet wird. Dabei setzen wir $z_1 = y, z_2 = y', z_3 = y''$. Man erhält dann das System:

$$\begin{aligned} z_1' &= y' = z_2 \\ z_2' &= y'' = z_3 \\ z_3' &= y''' = -z_2^2 e^{z_2} + x z_1 \end{aligned} \qquad (8.14)$$

Dieses System kann nun mit den bekannten Methoden gelöst werden und wir erhalten unter Zuhilfenahme der Anfangswerte die punktweise Lösungsfunktion sowie deren Ableitungsfunktionen y' und y''.

Diese Reduktionsmethode lässt sich auf Differenzialgleichung beliebiger Ordnung *n* anwenden, falls diese sich nach explizit nach $y^{(n)}$ auflösen lassen.

Beispiel:
Bestimmen Sie die zugehörigen DGL und berechnen Sie den ersten Wert der Lösungsfunktion für den Maschenstrom bei einer Schrittweite h=0.02ms ab dem Zeitpunkt t_0=0 nach dem Verfahren von Euler



Vorgaben:
C=1uF
L =10mH
$u_e(t)$=0V bis t_0, nachher 1V

Wir erhalten aus dem Maschenansatz eine Integral-Differenzialgleichung. Durch Ableiten wird daraus eine DGL zweiter Ordnung:

$$\frac{d^2 i(t)}{dt^2} = \frac{1}{L}\left(\frac{du_e(t)}{dt} - \frac{1}{C} i(t)\right)$$

Die Anfangswerte werden bei einem hinreichen lange ausgeschalteten System i(0)=0, i'(0)=1/L.

Das Ordnungsreduktionsverfahren liefert uns aus der DGL zweiter Ordnung ein DGL-System mit zwei DGL erster Ordnung:

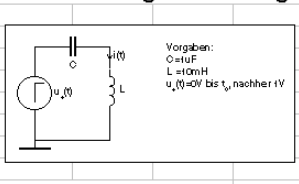
Substitution: $z_1 = i(t) \qquad z_1' = i'(t) = z2 \qquad z_2' = i''(t)$

DGL – System: $z_1' = z_2$

$$z_2' = \frac{1}{L}\left(\frac{du_e(t)}{dt} - \frac{z1}{C}\right)$$

Mit $\frac{du_e(t)}{dt} = 0$ und den Anfangswerten wird die Lösungsfunktion punktweise berechnet. Mit einer EXCEL-Tabelle werden diese:

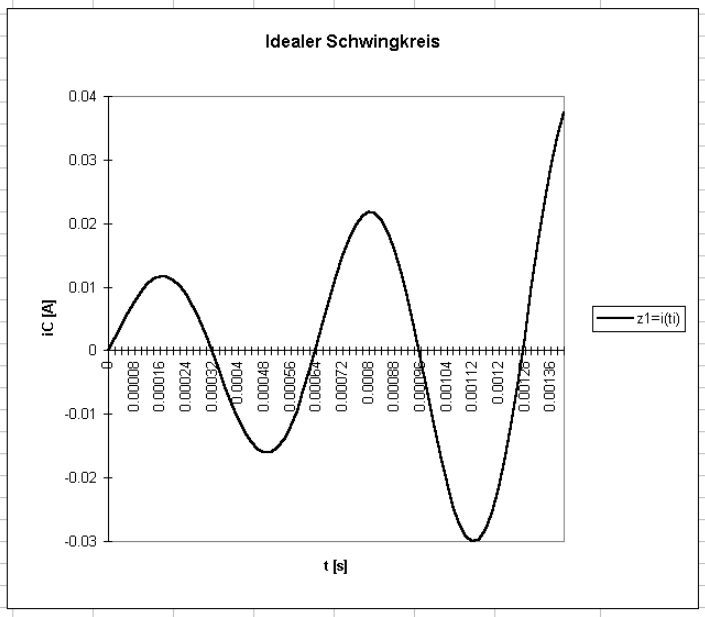
DGL2. Ordnung mit Ordnungsreduktion und Verfahren von Euler:



L =	0.01		$i(t_0)$ =	0	
C =	1.00E-06		$i'(t_0)$ =	1.00E+02	
U_E =	1		T =	1E-08	
t_0 =	0				
h =	2.00E-05				

Rechenschritte nach Euler:

i	t_i	$z1=i(t_i)$	$z2=i'(t_i)$
0	0	0	100
1	0.00002	2.00E-03	100
2	0.00004	4.00E-03	96
3	0.00006	5.92E-03	88
4	0.00008	7.68E-03	76.16
5	0.0001	9.20E-03	60.8
6	0.00012	1.04E-02	42.3936
7	0.00014	1.13E-02	21.5552
8	0.00016	1.17E-02	-0.978944
9	0.00018	1.17E-02	-24.375296
10	0.0002	1.12E-02	-47.73249
11	0.00022	1.02E-02	-70.114673
12	0.00024	8.83E-03	-90.587555
13	0.00026	7.02E-03	-108.25585
14	0.00028	4.86E-03	-122.30064
15	0.0003	2.41E-03	-132.0152
16	0.00032	-2.29E-04	-136.83774
17	0.00034	-2.97E-03	-136.37966
18	0.00036	-5.69E-03	-130.44808
19	0.00038	-8.30E-03	-119.06131
20	0.0004	-1.07E-02	-102.45662
21	0.00042	-1.27E-02	-81.08947
22	0.00044	-1.44E-02	-55.624059
23	0.00046	-1.55E-02	-26.91507
24	0.00048	-1.60E-02	4.0188821
25	0.0005	-1.59E-02	36.029437
26	0.00052	-1.52E-02	67.879236
27	0.00054	-1.38E-02	98.287858
28	0.00056	-1.19E-02	125.98131



8.6 Predictor-Corrector-Verfahren

Die oft an sich bescheidene Genauigkeit der numerischen Lösungsverfahren für DGL kann durch sog. Mehrschrittmethoden verbessert werden. Die Genauigkeit des Resultates kann dadurch bei gleicher Schrittweite *h* deutlich verbessert werden.

8.6.1 Arbeitsweise

Alle bisher gezeigten Verfahren arbeiten nach dem Prinzip, dass ausgehend von einem Wert y_n bei x_n der neue Wert y_{n+1} bei x_{n+1} extrapoliert (prädiktiert, d.h. vorhergesagt) wird. Dieser Wert ist mehr oder weniger fehlerbehaftet. Mit Hilfe eines Interpolationsschrittes wird nun der extrapolierte Wert korrigiert. Dieser Korrekturschritt bringt eines wesentliche Verbesserung des Resultates. Unter bestimmten Voraussetzungen kann diese Korrektur kann sogar mehrfach wiederholt werden und bringt eine weitere Verbesserung.

Ein einfaches Predictor-Correctorpaar ist:

$$y_{Pn+1} = y_{Cn} + h \cdot y'(x_n, y_{Cn}) \qquad \text{(Euler – Predictor)} \tag{8.15}$$

$$y_{Cn+1} = y_{Cn} + \frac{h}{2}(y'(x_n, y_{Cn}) + y'(x_{n+1}, y_{Pn+1})) \qquad \text{(Modifiziertes Euler - Verfahren als Corrector)} \tag{8.16}$$

Mit dem Predictor wird eine erste Näherung für das Resultat im Schritt *n* bestimmt. Das erhaltene Resultat wird dann zur Interpolation im Corrector eingesetzt und wir erhalten eine verbesserte Näherung.

Für die Praxis kennt man zahlreiche weitere Predictor-Correctorpaare, welche aber etwas aufwendiger sind (z.B. Methode von Adams-Moulton, Methode von Milne, u.a.). Sie sind durchwegs Mehrschritt-verfahren, welche mehrere Vorgängerwerte verrechnen. Problematisch kann bei diesen Methoden die Bildung der entsprechenden Startwerte sein (z.B. Adams und Milne brauchen 4 Startwerte).

Beispiel:
Predictor-Corrector-Verfahren mit der Methode von Euler:

Wir lösen die DGL : $y' = 2y$ $y(0) = 1$, Schrittweite $h = 0.2$, Lösungsintervall: $x \in [0,3]$

1. Der Predictorwert y_{1P} für *x=0.2* wird:
   $y_{1P} = y_0 + h \cdot y'(x_0, y_0) = 1 + 0.2 \cdot 2 \cdot 1 = 1.4$

2. Der Correctorwert y_{1C} für *x=0.2* wird jetzt:
   $y_{1C} = y_0 + \frac{h}{2} \cdot (y'(x_0, y_0) + y'(x_1, y_{1P}) = 1 + \frac{0.2}{2}(2 \cdot 1 + 2 \cdot 1.4) = 1.48$
   Dies verkörpert nun die verbesserte Lösung an der Stelle *x=0.2*.

3. Der Predictorwert y_{2P} für die nächste Stelle bei *x=0.4* wird:
   $y_{2P} = y_{1C} + h \cdot y'(x_1, y_{1C}) = 1.48 + 0.2 \cdot 2 \cdot 1.48 = 2.072$

4. Der Correctorwert y_{2C} bei *x=0.4* wird:
   $y_{2C} = y_{1C} + \frac{h}{2}(y'(x_1, y_{1C}) + y'(x_2, y_{2P})) = 1.48 + \frac{0.20}{2}(2 \cdot 1.48 + 2 \cdot 2.072) = 2.1904$

5. Die restlichen Schritte werden analog durchgeführt.

Dieses Predictor-Corrector-Prinzip zur Verbesserung der Genauigkeit ist grundsätzlich für alle bisher gezeigten Lösungsverfahren geeignet. Die Genauigkeit kann sogar noch etwas verbessert werden, wenn das Corrector-Verfahren noch einmal angewandt wird.

Wir wenden den Corrector ein zweites Mal an und erhalten die nochmals verbesserten Lösungen:

2a. Der doppelt korrigierte Wert y_{1CC} bei *x=0.2* wird:
   $y_{1CC} = y_0 + \frac{h}{2}(y'(x_0, y_0) + y'(x_1, y_{1C})) = 1 + \frac{0.2}{2}(2 \cdot 1 + 2 \cdot 1.48) = 1.496$

3. Der neue Predictorwert y_{2PP} bei *x=0.4* wird nun:
   $y_{2P} = y_{1CC} + h \cdot y'(x_1, y_{1CC}) = 1.496 + 0.2 \cdot 2 \cdot 1.496 = 2.0944$

4. Der erste Correctorwert y_{1C} bei *x=0.4* :
   $y_{2C} = y_1 + \frac{h}{2}(y'(x_1, y_{1CC}) + y'(x_2, y_{2P})) = 1.496 + \frac{0.2}{2}(2 \cdot 1.496 + 2 \cdot 2.0944) = 2.21408$

4a. Der zweite Correctorwert y_{2CC} bei *x=0.4* wird:
   $y_{2CC} = y_{1CC} + \frac{h}{2}(y'(x_1, y_{1CC}) + y'(x_2, y_{2C})) = 1.496 + \frac{0.2}{2}(2 \cdot 1.496 + 2 \cdot 2.21408) = 2.238016$

Die folgenden Werte lassen sich aus der nachfolgenden Tabelle ablesen, ebenso die relativen Fehler bezüglich der exakten Lösung der DGL:

DGL mit Predictor-Corrector-Verfahren

Predictor: Euler $y_{Pn+1} = y_{Cn} + h \cdot y'(x_n, y_{Cn})$

Corrector: Modified Euler $y_{Cn+1} = y_{Cn} + \frac{h}{2} \cdot \left(y'(x_n, y_{Cn}) + y'(x_{n+1}, y_{Pn+1})\right)$

x_0 = 0
y_0 = 1
h = 0.2

			Euler normal		Eine Corrector-Iteration			Zwei Corrector-Iterationen			
i	x_i		y_i	Fehler (rel.)	y_{iP}	y_{iC}	Fehler (rel.)	y_{iP}	y_{iC}	y_{iCC}	Fehler (rel.)
0	0		1.0000	0.00%	1.0000	1.0000	0.00%	1.0000	1.0000	1.0000	0.00%
1	0.2		1.0000	-1.98%	1.0000	1.0200	-0.02%	1.0000	1.0200	1.0204	0.02%
2	0.4		1.0400	-4.00%	1.0608	1.0828	-0.04%	1.0612	1.0833	1.0841	0.08%
3	0.6		1.1232	-6.18%	1.1695	1.1963	-0.08%	1.1709	1.1978	1.1994	0.18%
4	0.8		1.2580	-8.65%	1.3399	1.3753	-0.13%	1.3433	1.3788	1.3816	0.33%
5	1		1.4593	-11.49%	1.5953	1.6448	-0.24%	1.6027	1.6524	1.6574	0.53%
6	1.2		1.7511	-14.76%	1.9738	2.0462	-0.40%	1.9889	2.0618	2.0706	0.79%
7	1.4		2.1714	-18.51%	2.5373	2.6469	-0.66%	2.5675	2.6785	2.6940	1.11%
8	1.6		2.7794	-22.72%	3.3881	3.5596	-1.03%	3.4484	3.6229	3.6509	1.51%
9	1.8		3.6688	-27.40%	4.6987	4.9749	-1.55%	4.8191	5.1024	5.1534	1.99%
10	2		4.9895	-32.47%	6.7659	7.2235	-2.24%	7.0087	7.4828	7.5776	2.55%
11	2.2		6.9853	-37.89%	10.1130	10.8931	-3.14%	10.6086	11.4270	11.6071	3.21%
12	2.4		10.0589	-43.53%	15.6861	17.0542	-4.27%	16.7142	18.1720	18.5219	3.97%
13	2.6		14.8871	-49.31%	25.2403	27.7097	-5.66%	27.4124	30.0944	30.7917	4.84%
14	2.8		22.6284	-55.10%	42.119	46.7075	-7.33%	46.8034	51.9025	53.3303	5.81%
15	3		35.3004	-60.78%	72.86	81.645	-9.30%	83.20	93.2213	96.2292	6.90%

Hinweis:
Ein weiterer Correctorschritt bringt hier keine weitere Verbesserung, sondern verschlechtert die Resultate. Grundsätzlich stellt nur der erste Correctorschritt eine Verbesserung sicher. Ob ein weiterer Schritt eine Verbesserung bringt hängt von der Funktion *y'(x,y)* und den verwendeten Predictor-Correctorfunktionen selbst ab.

8.7 Runge-Kutta-Verfahren höherer Ordnung

Diese Verfahren wurden von Carl Runge und Wilhelm Kutta entwickelt um Differenzialgleichungen mit einer guten Genauigkeit zu lösen. Hierbei wird im Prinzip wie beim Taylorverfahren höherer Ordnung gearbeitet, ohne dass jedoch alle Ableitungen vorliegen müssen.

Runge-Kutta.-Verfahren sind in 2., 3., 4. und höherer Ordnung bekannt. Das Verfahren zweiter Ordnung dient weniger zur konkreten Berechnung der Lösung, sondern hat mehr schulmässigen Charakter um die grundsätzliche Arbeitsweise der Methode aufzuzeigen und zu analysieren.

Praktikabel für gute Resultate bei einem einfachen Aufwand ist das Runge-Kutta-Verfahren 4. Ordnung, das wir nachfolgend

8.7.1 Runge-Kutta 2. Ordnung (Heun-Verfahren)

Bei diesem Verfahren wird die Genauigkeit verbessert indem jeweils im Punkt x_i statt der Tangentensteigung der Mittelwert aus dem Anstieg in x_i und x_i+h benutzt wird, ähnlich dem Trapezverfahren bei der Integration. Auch hier nimmt der Fehler quadratisch mit der Schrittweite ab.

Wir erhalten dann für die beiden Punkte die Steigungen:

$$p_i = f(x_i, y_i) \qquad \text{Steigung im Punkt } x_i \tag{8.17}$$

$$q_i = f(x_i + h, y_i + h \cdot p_i) \qquad \text{Steigung im Punkt } x_i + h \tag{8.18}$$

Mit der Euler-Cauchy-Näherung erhalten wir als Näherung y_{i+1}. Sie dient als Grundlage für die

Näherungsformel nach Heun:

$$y_{i+1} = y_i + h\,f(x_i, y_i) = y_i + h\,p_i \tag{8.19}$$

Das Verfahren von Heun benutzt den Mittelwert der beiden Steigungen p_i und q_i. Wir setzen dafür die Werte ein und erhalten die Heun-Formel:

$$y_{i+1} = y_i + h\frac{p_i + q_i}{2} \tag{8.20}$$

Heun-Näherung

8.7.2 Implementierung des Heun-Verfahrens

Wir entwerfen wiederum ein allgemeines Programm zur Lösung der DGL vom Typus $y' = f(x,y)$ im Intervall $[x_0, x_n]$.

Die wesentlichen Schritte zur Entwicklung des Algorithmus ergeben sich aus der Rechenvorschrift und können in diesem einfachen Fall direkt auscodiert werden:

```
pi = f(xi,yi);
qi = f(xi+h, yi+h*pi);
yi = yi + h*(pi+qi)/2;
```

`f(xi,yi)` wird sinnvollerweise wiederum als eigenständige Funktion aus dem Hauptprogramm ausgelagert (separat definiert), sodass bei einer Änderung der Aufgabenstellung nur gerade diese Funktion angepasst werden muss und keine Änderungen im Hauptteil notwendig sind.

Ein Test mit der DGL $y' = x \cdot y$ und $y(0) = 1$ ergibt für die Schrittweite h=0.2 (resp. n=5):

```
Numerische Lösung der Differenzialgleichung mit dem Heun-Verfahren:
Intervallbeginn x0: 0
Intervallende xn: 1
Anfangswert f(x0)=y0: 1
Anzahl Schritte: 5
Lösungsfunktion:
x= 0.000        y=  1.00000000
x= 0.200        y=  1.02000000
x= 0.400        y=  1.08283200
x= 0.600        y=  1.19631279
x= 0.800        y=  1.37528119
x= 1.000        y=  1.64483630
```

Ein Vergleich zeigt die Verbesserung der Resultate gegenüber dem Euler-Verfahren:

i	x_i	Euler	Heun	exakt
0	0.0	1.000	1.000	1.000
1	0.2	1.000	1.020	1.020
2	0.4	1.040	1.083	1.083
3	0.6	1.123	1.196	1.197
4	0.8	1.258	1.375	1.377
5	1.0	1.459	1.644	1.649

Eine mögliche Implementierung des Verfahrens in C:

```
/* Programm HEUN                                                    File: HEUN.C
   Numerische Lösung von Differenzialgleichungen nach dem Heun-Verfahren (Runge-Kutta 2. Ordnung)

   Das Verfahren bestimmt die numerische Lösung der DGL der Art

      y' = f(x,y)

   wobei f(x,y) als Funktion vorgegeben wird.
   Die Vorausschaetzung fuer yi wird hierbei aus dem Mittelwert der Steigung in den Punkten
   xi und xi+h gewonnen, was eine wesentliche Verbesserung der Genauigkeit gegenueber dem
   Euler-Cauchy-Verfahren bringt.

   Arbeitsweise:
   Ausgehend vom Startwert werden n Schritten die Loesungsfunktion punktweise bestimmt
   und tabelliert ausgegeben.

   Autor: Gerhard Krucker
   Datum: 7.6.1995
   Sprache: MS Visual-C V1.5  (QuickWin App.)
*/

#include <stdio.h>

/* Definition der Ableitung nach der Aufgabenstellung:
    y'=f(x,y)
*/
double f(double x, double y)
{
   return x * y;
}

int main ()
{ double x0;    /* Intervallstart */
  double xn;    /* Intervallende */
  double y0;    /* Anfangswert y0 bei x0 */
  int n;        /* Anzahl Schritte */
  double h;     /* Schrittweite */
  double pi, qi;/* Steigungen in den Punkten xi, xi+h */
  double xi,yi; /* Errechnete Werte im Schritt #i */
  int i;

  printf("Numerische Lösung der Differenzialgleichung mit dem Heun-Verfahren:\n");
  printf("Intervallbeginn x0: "); scanf("%lg",&x0);
  printf("Intervallende xn: "); scanf("%lg",&xn);
  printf("Anfangswert f(x0)=y0: "); scanf("%lg",&y0);
  printf("Anzahl Schritte: "); scanf("%d",&n);
  printf("Lösungsfunktion: \n");

  h = (xn- x0) / n;
  yi = y0;

  for (i=0; i <= n ; i++)
   {
     xi = x0 + h * i;   /* Aktuelles x im Schritt #i */
     printf("x=%6.3f\ty=%12.8f\n",xi,yi );
     pi = f(xi, yi);                    /* pi, und qi berechnen */
     qi = f(xi+h, yi+h*pi);
     yi = yi + h * (pi+qi)/2;           /* Heun-Formel */
   }
    return 0;
}
```

8.7.3 Fehlerbetrachtung zum Heun-Verfahren, Fehlerordnung

Ohne näher auf die gesamte Problematik der Fehlerrechnung einzugehen, sehen wir, dass das Heun-Verfahren eine wesentliche Verbesserung der Genauigkeit gegenüber dem Euler-Verfahren bringt.

Die nachfolgende Tabelle zeigt wie sich bei beiden Verfahren die Fehler mit der Halbierung der Schrittweite entwickeln:

n	Euler	Fehler	Heun	Fehler
5	1.459	0.190	1.6448	0.0039
10	1.547	0.102	1.6479	0.0008
20	1.596	0.053	1.6485	0.0002
40	1.622	0.027	1.6487	0.0000

Der Fehler nimmt beim Euler-Verfahren proportional ab und beim Heun-Verfahren quadratisch ab.

In der Numerik werden die Verfahren bezüglich Ihres Fehlers geordnet:
Ein numerisches Verfahren zur Lösung eines Anfangswertproblems bei einer gegebenen Schrittweite *h*

heisst von der Fehlerordnung p wenn der Fehler am Ende des Intervalls proportional zu h^p abnimmt.

Somit sind das Euler-Cauchy-Verfahren von der Fehlerodnung 1 und das Heun-Verfahren von der Fehlerordnung 2. Häufig genügt für praktische Anwendungen bereits das Heun-Verfahren.

8.7.4 Runge-Kutta-Formel 4. Ordnung

Mit ähnlichen Überlegungen (Taylorreihen mit zwei Variablen) wie bei den bisher gezeigten Verfahren kann die Runge-Kutta-Formel 4. Ordnung (RK4)entwickelt werden. Auf eine detaillierte Herleitung wird verzichtet. Im Bedarfsfall konsultiere man einschlägige Literatur.

Mit einfachen Worten kann aber die Arbeitsweise erklärt werden:
Während beim Taylorreihenverfahren 4. Ordnung alle Ableitungen bis $y^{(4)}$ vorliegen müssen, werden beim Runge-Kutta-Verfahren die Ableitungen $y''..y^{(4)}$ in jedem Rechenschritt numerisch bestimmt.

Die Rechnung erfolgt mit dem Formelsatz für das Runge-Kutta-Vefahren 4. Ordnung:

$$
\begin{aligned}
a_i &= f(x_i, y_i) \\
b_i &= f(x_i + h/2, y_i + h\, a_i/2) \\
c_i &= f(x_i + h/2, y_i + h\, b_i/2) \\
d_i &= f(x_i + h, y_i + h\, c_i)
\end{aligned}
$$

Runge-Kutta Formel 4. Ordnung (8.21)

$$y_{i+1} = y_i + \frac{h}{6}(a_i + 2b_i + 2c_i + d_i)$$

Durch den schrittweisen Aufbau der Rechnung werden die Anstiege a_i, b_i, c_i, d_i mit den jeweils besten verfügbaren Schätzungen bestimmt. So wird zur Berechnung von b_i der Wert von a_i, für c_i der Wert von d_i, etc, herangezogen. Die Gewichtung der Anstiegswerte in der Runge-Kutta-Formel ist mit der Simpson-Regel vergleichbar, auch dort hatten wir an den Enden das Gewicht 1/6 und in der Mitte 4/6.

Wie bereits erwähnt ist das Runge-Kutta-Verfahren ein Verfahren 4. Ordnung, d.h. der Fehler am Intervallende ist proportional zu h^4.

8.7.5 Implementierung des Runge-Kutta-Verfahrens

Die Programmstruktur ist praktisch genau gleich wie bei den vorher entwickelten Programmen. Der Unterschied liegt einzig darin, dass ein anderer Formelsatz angewandt wird. Deshalb wird auf einen feineren Entwurf verzichtet und wir zeigen eine mögliche Implementierung:

```
/* Programm RUNGE-KUTTA                                                    File: RUNKUTTA.C
   Numerische Lösung von Differenzialgleichungen nach der Methode Runge-Kutta 4. Ordnung.
   Das Verfahren bestimmt die numerische Lösung der DGL der Art

         y' = f(x,y)

   wobei f(x,y) als Funktion vorgegeben wird.

   Arbeitsweise:
   Ausgehend vom Startwert werden n Schritte mit der Runge-Kutta-Formel
   durchgeführt und die Werte tabelliert ausgegeben.

   Autor: Gerhard Krucker
   Datum: 7.6.1995
   Sprache: MS Visual-C V1.5  (QuickWin App.)
*/

#include <stdio.h>

/* Definition der Ableitung nach der Aufagebenstellung:
    y'=f(x,y)
*/
double f(double x, double y)
{
   return x * y;
}

int main ()
{ double x0;    /* Intervallstart */
  double xn;    /* Intervallende */
  double y0;    /* Anfangswert y0 bei x0 */
  int n;        /* Anzahl Schritte */
  double h;     /* Teilintervallbreite */
  double xi,yi; /* Errechnete Werte im Schritt #i */
  int i;

  printf("Numerische Lösung der Differenzialgleichung mit dem Runge-Kutta Verfahren:\n");
  printf("Intervallbeginn x0: "); scanf("%lg",&x0);
  printf("Intervallende xn: "); scanf("%lg",&xn);
  printf("Anfangswert f(x0)=y0: "); scanf("%lg",&y0);
  printf("Anzahl Schritte: "); scanf("%d",&n);
  printf("Lösungsfunktion: \n");

  h = (xn- x0) / n;
  yi = y0;

  for (i=0; i <= n ; i++)
   { double a,b,c,d;   /* Runge-Kutta Koeffizienten */

     xi = x0 + h * i;   /* Aktuelles x im Schritt #i */
     printf("x=%6.3f\ty=%12.8f\n",xi,yi );

     a = f(xi,yi);
     b = f(xi + h / 2.0, yi + h * a / 2.0);
     c = f(xi + h / 2.0, yi + h * b / 2.0);
     d = f(xi + h, yi + h * c);
     yi = yi + h * (a + 2 * b + 2 * c + d) / 6.0;
   }
  return 0;
 }
```

8.7.6 Test des Runge-Kutta Verfahrens

Als Testbeispiel verwenden wir unsere Differenzialgleichung $y'=x\,y$ auf dem Intervall [0,1] mit dem Anfangswert $y(0)=1$ und n=5:

```
Numerische Lösung der Differenzialgleichung mit dem Runge-Kutta Verfahren:
Intervallbeginn x0: 0
Intervallende xn: 1
Anfangswert f(x0)=y0: 1
Anzahl Schritte: 5
Lösungsfunktion:
x= 0.000        y=  1.00000000
x= 0.200        y=  1.02020133
x= 0.400        y=  1.08328699
x= 0.600        y=  1.19721701
x= 0.800        y=  1.37712642
x= 1.000        y=  1.64871668
```

Die exakte Lösung ist $y= \exp(x^2/2)$ und wir erhalten mit n=5 folgende Werte im Vergleich zu den anderen Verfahren:

i	x_i	Euler	Heun	RK4	exakt
0	0.0	1.000	1.000	1.00000	1.000000
1	0.2	1.000	1.020	1.020201	1.020201

2	0.4	1.040	1.083	1.083287	1.083287
3	0.6	1.123	1.196	1.197217	1.197217
4	0.8	1.258	1.375	1.377126	1.377128
5	1.0	1.459	1.644	1.648717	1.648721

Die Runge-Kutta-Näherung stimmt am Intervallende bereits auf 5 Dezimalstellen mit dem exakten Wert überein. Dies zeigt anschaulich, dass das Verfahren in der Regel für die Praxis sehr gute Resultate bei einem bescheidenen Aufwand liefert.

8.7.7 Runge-Kutta-Verfahren mit EXCEL

Dasselbe Beispiel berechnen wir nun mit dem Tabellenkalkulationsprogramm EXCEL. Wiederum wird zuerst ein Arbeitsblatt erzeugt, dass die Vorgaben sowie ein Gerüst für die Wertetabelle (Index i und Argumente x_i) enthält. Zusätzlich werden für die Tangentensteigungen eigene Spalten definiert.

Wir erhalten dann:

	A	B	C	D	E	F	G	H	I
1	**Lösung einer DGL mit dem Runge-Kutta Verfahren:**								
2									
3	Aufgabe:	Anfangswertproblem							
4		$y' = xy$		$x \in [0,2]$					
5									
6		$y(0) = 1$		$h = 0.2$					
7									
8	Vorgaben:								
9	x_0 =	0	h =	0.2					
10	x_n =	2							
11	y(0) =	1							
12									
13	Wertetabelle:								
14	i	x_i	a_i	b_i	c_i	d_i	y_i (RK4)		
15	0	0	0.00000	0.10000	0.10100	0.20404	1.00000		
16	1	0.2	0.20404	0.31218	0.31543	0.43331	1.02020		
17	2	0.4	0.43331	0.56331	0.56981	0.71835	1.08329		
18	3	0.6	0.71833	0.88834	0.90024	1.10181	1.19722		
19	4	0.8	1.10170	1.33857	1.35988	1.64910	1.37713		
20	5	1	1.64872	1.99495	2.03303	2.46639	1.64872		
21	6	1.2	2.46530	2.99123	3.05960	3.73288	2.05442		
22	7	1.4	3.73018	4.55615	4.68004	5.76068	2.66441		
23	8	1.6	5.75444	7.09234	7.31979	9.10886	3.59652		
24	9	1.8	9.09499	11.32832	11.75265	14.80661	5.05277		
25	10	2					7.38822		
26									
27									

In der Tabelle stehen in den verfahrensrelevanten Zellen folgende Einträge:

```
B15: =$B$9+$D$9 * A15
C15: =B15 * G15
D15: =(B15 + $D$9 / 2) * (G15 + $D$9 * C15 / 2)
E15: =(B15 + $D$9 / 2) * (G15 + $D$9 * D15 / 2)
F15: =(B15 + $D$9) * (G15 + $D$9 * E15)
G15: =$B$11
G16: =G15 + $D$9 / 6 * (C15 + 2 * D15 + 2 * E15 + F15)
```

An diesem Beispiel können wir die Berechnung nach dem Runge-Kutta-Verfahren ohne grossen Aufwand schrittweise nachvollziehen.

8.7.8 Runge-Kutta-Verfahren für DGL-Systeme

Die Lösung eines DGL-Systems erfolgt grundsätzlich gleich wie bei einer einzelnen Differenzialgleichung, nur dass hier y, y' und die Funktionsanstiege vektorielle Grössen sind.

Wir beschränken uns auf das Lösungsverfahren nach Runge-Kutta, da es praktisch den grössten Nutzen erbringt. Mit den gleichen Überlegungen können aber das Euler-Cauchy- und das Heun-Verfahren für DGL-System erweitert werden.

In bekannter Weise wird das Intervall $[x_0,x_n]$ in n äquidistante Teilstücke der Länge h mit den Teilpunkten x_i zerlegt:

$$h = \frac{x_n - x_0}{n} \qquad x_i = x_0 + i\,h$$

Jede durch das System beschriebene Differenzialgleichung wird mit dem Formelsatz von Runge-Kutta in jedem Schritt berechnet. Wir erhalten dabei Vektoren für die Anstiege *a, b, c, d* und die Näherung für *y*:

$$a_{ij} = f_j(x_i, \underline{y_i}) \qquad a_i = (a_{i1}, a_{i2}, \ldots, a_{in})$$
$$b_{ij} = f_j(x_i + h/2, \underline{y_i} + h\underline{a_i}/2)$$
$$c_{ij} = f_j(x_i + h/2, \underline{y_i} + h\underline{b_i}/2)$$
$$d_{ij} = f_j(x_i + h, \underline{y_i} + h\underline{c_i})$$
$$\underline{y_{i+1}} = \underline{y_i} + h(\underline{a_i} + 2\underline{b_i} + 2\underline{c_i} + d_i)/6$$

Runge-Kutta-Formel für DGL-Systeme (8.22)

8.7.9 Implementierung der RK-Formel für DGL-Systeme

Anders als für eine einfache DGL kann der Runge-Kutta-Formelsatz nicht direkt implementiert werden, da C keine Vektoroperationen unterstützt.

Die Vektoroperationen führt man zweckmässigerweise für alle Komponenten des Vektors in einer Schleife aus. So wird dann beispielsweise eine skalare Multiplikation des Vektors `a` mit der Zahl `2`:

```
for (i=0; i< n; i++)
   a[i] = a[i] * 2;
```

Ebenso können Funktionen keine Vektoren als Funktionswert (Resultat) zurückgeben, wie auch Vektoren (d.h Arrays) nicht durch eine einfache Zuweisung '=' zugewiesen, resp. kopiert werden können.

Die Problematik der Resultatrückgabe betrachten wir an der Funktion *f(x,y)* , die den Vektor der Ableitungswerte berechnet.

Anstatt einer Rückgabe über `return` als Funktionswert werden die Resultate in einen leeren Vektor `yr` eingeschrieben, der vom Aufrufer bereitgestellt wurde:

Beispiel: $\underline{y}' = (y_1(y_2 - x), y_2 - \ln y_1)$
Wahl der Datentypen: Grunddatentyp double für alle Steigungen und Funktionswerte und Argumente.
Da wir hier ein DGL-System erster Ordnung mit zwei Bestimmungsgleichungen haben, organisieren wir die Daten in einem Array mit 2 Komponenten:

```
double a[2],b[2], c[2],d[2];    /* Tangentensteigungen */
double yi[2], y0[2];
```

oder etwas eleganter über eine explizite Typendefinition (siehe Programmbeispiel).

Der Entwurf der Ablaufstruktur für ein Programm zur Lösung eines Systems mit *n* Gleichungen erfolgt nach den gleichen Gesichtspunkten bezüglich Funktionalität wie beim RK-Verfahren für eine einzelne DGL.

```
/* Programm RUNGE-KUTTA fuer Differenzialgleichungssysteme.                          File: RKSYS.C
   Numerische Lösung von Differenzialgleichungssystemen erster Ordnung nach der Methode
   Runge-Kutta 4. Ordnung.

   Das Verfahren bestimmt die numerische Lösung der DGL der Art

      y1' = f1(x,y1,y2,...,yn)
      ....
      yn' = fn(x,y1,y2,...,yn)

   Arbeitsweise:
   Ausgehend vom Startwert werden n Schritte mit der Runge-Kutta-Formel
   fuer jede Funktion durchgeführt und die Werte tabelliert ausgegeben.

   Autor: Gerhard Krucker
   Datum: 27.6.1995
   Sprache: MS Visual-C V1.5  (QuickWin App.)
*/

#include <stdio.h>
#include <math.h>

#define MAX 10               /* Maximale Anzahl der Bestimmungsgleichungen */

typedef double vektor[MAX];

/* Definition der Ableitung nach der Aufgabenstellung:
   Hierbei wird das Resultat ueber das Array yr in Form eines Call by reference zurueckgegeben.
   Parameter: y[] Vorgaengerwerte
              x aktuelles Argument (Punkt xi im Intervall)
              yr[] Naeherung fuer den Punkt x

  Hier: y1'=y1 (y2 -x)
        y2'=y2 - ln y1
*/
void f(double x,vektor y, vektor yr)
{  yr[0] = y[0] * (y[1] - x);          /* y1'= y1 (y2 - x) */
   yr[1] = y[1] -log(y[0]);           /* y2'= y2 - ln y1 */
}

int main ()
{ int r;        /* Anzahl Bestimmungsgleichungen */
  int n;        /* Anzahl Teilintervalle */

  double h;     /* Teilintervallbreite */
  double x0;    /* Intervallstart */
  double xn;    /* Intervallende */
  double xi;    /* Argument im Schritt #i */
  vektor y;    /*  */
  vektor yi;     /* Errechneter Wert fuer y im Schritt #i */

  int i,j;
```

```
  printf("Numerische Lösung eines Differenzialgleichungssystems mit dem Runge-Kutta Verfahren:\n");
  printf("Anzahl der Gleichungen: "); scanf("%d",&r);
  printf("Intervallbeginn x0: "); scanf("%lg",&x0);
  printf("Intervallende xn: "); scanf("%lg",&xn);
  printf("Anzahl Schritte: "); scanf("%d",&n);

  printf("Anfangswerte der yi(x0):\n");
  for (i=0; i < r; i++)
    { printf("y%d[x0]: ",i+1); scanf("%lg",&yi[i]); }
    printf("Lösungsfunktion: \n");
    printf("   x         ");
    for (i = 0; i < r; i++)
    printf("y%d(x)          ",i+1);
    printf("\n");

    h = (xn- x0) / n;

    for (i=0; i <= n ; i++)
    { vektor a,b,c,d;   /* Runge-Kutta Tangentensteigungen */

      xi = x0 + h * i;   /* Aktuelles x im Schritt #i */
      printf("%6.3f : ",xi);
      for (j = 0; j < r; j++)
        printf("%12.8f\t",yi[j]);
      printf("\n");

      /* Runge Kutta Formel */
      for (j = 0; j < r; j++) y[j]= yi[j];                f(xi,y,a);
      for (j = 0; j < r; j++) y[j]= yi[j] + h * a[j]/2.0; f(xi + h / 2.0, y,b);
      for (j = 0; j < r; j++) y[j]= yi[j]+h*b[j]/2.0;     f(xi + h / 2.0, y, c);
      for (j = 0; j < r; j++) y[j]= yi[j]+h*c[j];         f(xi + h, y, d);
      for (j = 0; j < r; j++)
         yi[j] = yi[j] + h * (a[j] + 2 * b[j] + 2 * c[j] + d[j]) / 6.0;
    }

  return 0;
}
```

8.7.10 Testbeispiele

1. Wir prüfen unser Programm am bereits gezeigten Anfangswertproblem

$$y_1' = y_1(y_2 - x) \qquad y_1(0) = y_2(0) = 1$$
$$y_2' = y_2 - \ln(y_1)$$

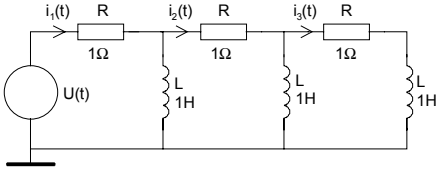
im Intervall [0,1] mit einem n=4 (Schrittweite h=0.25), indem wir die numerische Lösung mit der analytischen Lösung vergleichen:

i	x	$y_1(x)$ RK-Näherung	$y_1(x)$ exakt $y_1(x) = e^x$	$y_2(x)$ RK-Näherung	$y_2(x)$ exakt $y_2(x) = 1 + x$
0	0.0 0	1.00000000	1.00000000	1.00000000	1.00000000
1	0.25	1.28403742	1.28400254	1.25002444	1.25000000
2	0.50	1.64876289	1.64872127	1.50005229	1.50000000
3	0.75	2.11710255	2.11700002	1.75008256	1.75000000
4	1.00	2.71849752	2.71828183	2.00011380	2.00000000

Selbst für die grobe Schrittweite mit h=0.25 erhalten wir bereits eine brauchbare Näherung. Mit einer feinen Schrittweite von h=0.1 wird bereits eine Genauigkeit von 5 Nachkommastellen erreicht.

Die Genauigkeit des Runge-Kutta-Verfahrens für Systeme entspricht dem Verfahren für einfache Differentialgleichungen, hier der Fehlerordnung = 4.

2. Das elektrische Netzwerk soll mit einer Rechteckspannung gespiesen werden und wir möchten die Maschenströme bestimmen:



Dabei ist *U(t)* eine Rechteckspannung mit der Periodendauer von 10s und einer Amplitude von 10V. Wir stellen das Gleichungssystem für die drei Maschen auf:

$$U(t) = i_1(t)R + L\left(\frac{di_1(t)}{dt} - \frac{di_2(t)}{dt}\right) \qquad i_1' = \frac{di_1(t)}{dt} = \frac{1}{L}\left(-3R\,i_1 - 2R\,i_2 - R\,i_3 + 3U\right)$$

$$0 = L\left(\frac{di_2(t)}{dt} - \frac{di_1(t)}{dt}\right) + i_2(t)R + L\left(\frac{di_2(t)}{dt} - \frac{di_3(t)}{dt}\right) \quad \rightarrow \quad i_2' = \frac{di_2(t)}{dt} = \frac{1}{L}\left(-2R\,i_1 - 2R\,i_2 - R\,i_3 + 2U\right)$$

$$0 = L\left(\frac{di_3(t)}{dt} - \frac{di_2(t)}{dt}\right) + i_3(t)R + L\frac{di_3(t)}{dt} \qquad i_3' = \frac{di_3(t)}{dt} = \frac{1}{L}\left(-R\,i_1 - R\,i_2 - R\,i_3 + U\right)$$

Analytische Lösung

Vorgehen: Lineares inhomogenes DGL System hat die Lösung, die sich aus der Lösung des homogenen Systems sowie der partikulären Lösung zusammensetzt. Die Lösung des homogenen Systems wird aus den Eigenwerten und Eigenvektoren der Koeffizientenmatrix A gewonnen. Dieses Verfahren ist aber ziemlich aufwendig.

Besser ist die Lösung über eine Laplace-Transformation, die die DGL in eine algebraische Gleichung im Bildbereich überführt, diese nach i_1, i_2 und i_3 zu lösen und wieder in den Zeitbereich zurück zu transformieren.

Numerische Lösung

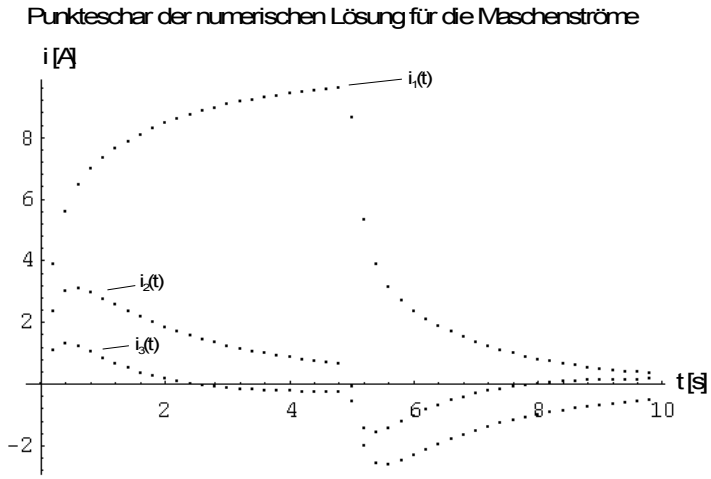
Wir definieren die Ableitungsfunktionen, die explizit vorliegen, als Vektorfunktion. Ferner wird die periodische Eingangsspannung über eine separat definierte Funktion modulo 10s ausgewertet:

```
/* Definieren der Rechteckeingangsspannung U(t) die mit einer Periodendauer von 10s eine Spannung von 0-10V
   liefert: */

double U(double t)
{ if (fmod(t, 10.0) < 5.0)
     return 10.0;
   else
    return 0;
}

/* Definition der Ableitung nach der Aufgabenstellung:
   Hierbei wird das Resultat ueber das Array yr in Form eines Call by reference zurueckgegeben.
   Parameter: y[] Vorgaengerwerte
              t aktuelle Zeit  (Punkt xi im Intervall)
              yr[] Naeherung fuer den Punkt t

*/
void f(double t,vektor i, vektor ir)
{ const double L = 1.0;
  const double R = 1.0;

   ir[0] = 1/L * (-3 * R * i[0] - 2 * R * i[1] - R * i[2] + 3 * U(t)); /* i1'= 1/L (-3R i1 -2R i2 - R i3 + 3U) */
   ir[1] = 1/L * (-2 * R * i[0] - 2 * R * i[1] - R * i[2] + 2 * U(t)); /* i2'= 1/L (-2R i1 -2R i2 - R i3 + 2U) */
   ir[2] = 1/L * (    - R * i[0] -     R * i[1] - R * i[2] +   U(t));  /* i2'= 1/L (-R  i1 - R i2 - R i3 + U) */
}
```

Wir starten das Programm und berechnen die Lösungen für das Intervall [0,10] und erhalten die Wertetabelle:

```
Numerische Lösung eines Differenzialgleichungssystems mit dem Runge-Kutta Verfahren:
Anzahl der Gleichungen: 3
Intervallbeginn x0: 0
Intervallende xn: 10
Anzahl Schritte: 50
Anfangswerte der yi(x0):
y1[x0]: 0
y2[x0]: 0
y3[x0]: 0
Lösungsfunktion:
```

x	y1(x)	y2(x)	y3(x)
0.000 :	0.00000000	0.00000000	0.00000000
0.200 :	3.89800000	2.35800000	1.10866667
0.400 :	5.59762902	3.03378658	1.31984760
0.600 :	6.45197925	3.10342376	1.22564620
0.800 :	6.96732711	2.96492948	1.04253284
1.000 :	7.33475049	2.76447566	0.84778659
1.200 :	7.62781012	2.55480723	0.66733963
1.400 :	7.87608047	2.35411337	0.50838553
1.600 :	8.09252136	2.16789313	0.37145881
1.800 :	8.28368493	1.99706638	0.25488783
2.000 :	8.45353773	1.84097317	0.15641708
2.200 :	8.60490581	1.69847484	0.07378105
2.400 :	8.74003330	1.56835097	0.00489209
2.600 :	8.86080879	1.44943604	-0.05211230
2.800 :	8.96886602	1.34066060	-0.09887155
3.000 :	9.06563584	1.24105804	-0.13682014
3.200 :	9.15237830	1.14975960	-0.16720952
3.400 :	9.23020588	1.06598585	-0.19112956
3.600 :	9.30010206	0.98903763	-0.20952826
3.800 :	9.36293691	0.91828753	-0.22322933
4.000 :	9.41948059	0.85317203	-0.23294767
4.200 :	9.47041522	0.79318461	-0.23930311
4.400 :	9.51634518	0.73786947	-0.24283233
4.600 :	9.55780622	0.68681610	-0.24399944
4.800 :	9.59527352	0.63965437	-0.24320520
5.000 :	8.62916875	-0.07061648	-0.57412841
5.200 :	5.34582184	-2.01494274	-1.43357746
5.400 :	3.89607994	-2.56269475	-1.55374647
5.600 :	3.15393009	-2.60850326	-1.42624891
5.800 :	2.69773668	-2.48310838	-1.23038328
6.000 :	2.36808437	-2.30782617	-1.02977032
6.200 :	2.10335770	-2.12630160	-0.84540290
6.400 :	1.87857778	-1.95346792	-0.68275716
6.600 :	1.68263091	-1.79372890	-0.54183096
6.800 :	1.50977365	-1.64770153	-0.42084014
7.000 :	1.35644904	-1.51470798	-0.31756149
7.200 :	1.22008033	-1.39368900	-0.22980448
7.400 :	1.09860254	-1.28353194	-0.15556262
7.600 :	0.99027081	-1.18318264	-0.09304846
7.800 :	0.89357369	-1.09167800	-0.04068807
8.000 :	0.80718729	-1.00815027	0.00289733
8.200 :	0.72994630	-0.93182160	0.03891255
8.400 :	0.66082266	-0.86199593	0.06840888
8.600 :	0.59890836	-0.79805047	0.09230274
8.800 :	0.54340089	-0.73942779	0.11139225
9.000 :	0.49359061	-0.68562855	0.12637197
9.200 :	0.44884976	-0.63620514	0.13784580
9.400 :	0.40862279	-0.59075591	0.14633830
9.600 :	0.37241783	-0.54892015	0.15230466
9.800 :	0.33979922	-0.51037355	0.15613944
10.000 :	1.31038091	0.19184238	0.49151757



8.7.11 RK-Verfahren mit variabler Schrittweite

Bei den bisherigen Verfahren wurde immer von einer konstanten Schrittweise *h* ausgegangen. Es liegt auf der Hand, dass mit einer feineren Schrittweite eine bessere Genauigkeit erreicht wird (abzüglich Rundungsfehler), jedoch steigt der Rechenaufwand stark an. Man beachte, dass das Verfahren mit kleinerer Schrittweite sehr schnell kleine Fehler bringt, aber umgekehrt führt eine Verdoppelung der Schrittweite zu einem ca. 16-fachen Fehler. Es gilt somit ein Kompromiss für die Schrittweite zu finden bei der eine gute Genauigkeit bei einem vernünftigen Rechenaufwand erreicht wird.

Wünschenswert wäre also ein Verfahren das in gewissen Grenzen selbstständig ein gute Schrittweite wählt und somit die Lösungskurve gut darstellen kann.

Empfehlenswert ist die Faustregel zur Wahl der Schrittweite *h* beim Runge-Kutta-Verfahren, so dass gilt:

$$k = K \cdot h \approx 0.1 \tag{8.23}$$

wobei

$$K = \max\left|f_y(x,y)\right| \tag{8.24}$$

f_y ist die partielle Ableitung nach y. Für *K* ist hier das Maximum im entsprechenden Intervall zu nehmen.

k heisst die **Schrittkennzahl** des Verfahrens.

Beispiel:
Die Test-DGL $y'=f(x,y)=xy$ auf dem Intervall [0,2]:

$$f_y(x,y) = \frac{\partial}{\partial y}(xy) = x \qquad K = \max\left|f_y\right| \qquad \text{(Maximum für } x \in [0,2]\text{)}$$

$$K \cdot h = 0.1 \qquad \rightarrow h \approx 0.05 \quad \text{bzw. } n = 40$$

Dieses Verfahren zur Bestimmung einer geeigneten Schrittweite kann recht aufwendig werden, da f_y normalerweise noch von *y* abhängt, das ja nicht bekannt ist. So ist dieser Rechengang zur Bestimmung ungeeignet.

Das Runge-Kutta-Verfahren liefert uns aber bei jedem Schritt eine Schätzung für die momentane Schrittkennzahl *k*, denn es gilt:

$$k = K \cdot h \approx 2\left|\frac{c-b}{b-a}\right| \tag{8.25}$$

wobei *a, b, c, d* die RK-Tangentensteigungen verkörpern.

Wir können nun in jedem Verfahrensschritt auf einfache Weise eine angemessene Schrittweite h bestimmen. Als brauchbar hat sich folgende Regel herausgestellt:

$k=K \cdot h < 0.01$: Schrittweite h verdoppeln
$k=K \cdot h > 0.08$: Schrittweite h halbieren
sonst: Schrittweite unverändert

Zweckmässigerweise wird die Startschrittweite relativ klein gewählt, da der Fehler der in den ersten Verfahrensschritten entsteht nachher nicht mehr zu korrigieren ist.

8.7.12 Implementierung RK mit Schrittweitensteuerung

Das bestehende Programm zur Rechnung mit dem Runge-Kutta-Verfahren (`RUNKUTTA.C`,8-20) wird nun mit einer Schrittweitensteuerung erweitert.

Statt der festen Schrittweite arbeiten wir mit einer *Startschrittweite h*, die dann nach jedem Schritt gemäss den vorher gezeigten Überlegungen angepasst wird. Der Code für die einzufügende Schrittweitensteuerung könnte lauten:

```
k = 2.0 * fabs((c-b)/(b-a));
if (k < 0.01) k*= 2;
if (k > 0.08) k/=2;
```

Vielfach interessiert uns nicht der Funktionsverlauf, sondern nur der Wert am Intervallende. Um sicherzustellen, dass auch für das Intervallende x_n ein Näherungswert vorliegt, endet der letzte Schritt "zwangsweise" exakt bei x_n durch:

```
if ((x+h) > xn) h = xn -x;
```

Bemerkungen:

1. Obwohl nach einer bestimmten Anzahl von Schritten der Endwert x_n bereits erreicht werden sollte, ist aufgrund von Rundungsfehlern die Abbruchbedingung $x >= x_n$ noch nicht erfüllt. Oft sind dann die Schritte so klein, dass sich die Resultate kaum noch vom letzten Schritt unterscheiden.

2. Zusätzlich kann *(b-a)* so klein werden, dass eine Division durch Null auftritt. Dieser Sachverhalt kann aber abgefangen werden indem man dafür sorgt, dass *(b-a)* ein Minimum nicht unterschreiten darf (Bsp. $eps=10^{-12}$).

3. Ein anderes Problem ist ein starker Anstieg der Lösungsfunktion, z.B. bei einer Polstelle. Dabei würde die Schrittweite h dauernd halbiert, sodass das Programm in einer Endlosschleife laufen würde. Dies kann ebenfalls durch Vorgabe einer minimalen Schrittweite h_{min} unterbunden werden. h_{min} ist natürlich stark von der Funktion abhängig. Je kleiner desto besser, für die meisten Anwendungen reicht aber $5 \cdot 10^{-3}$:

```
/* Programm RUNGE-KUTTA-VH                                                    File: RUNKUVH.C
   Numerische Lösung von Differenzialgleichungen nach der Methode
   Runge-Kutta 4. Ordnung mit Schrittweitensteuerung.

   Das Verfahren bestimmt die numerische Lösung der DGL der Art
         y' = f(x,y)
   wobei f(x,y) als Funktion vorgegeben wird.

   Arbeitsweise:
   Ausgehend von der Startschrittweite wird mit variabler Schrittweite mit der Runge-Kutta-Formel
   durchgeführt und die Werte tabelliert ausgegeben.

   Autor: Gerhard Krucker
   Datum: 20.6.1995
   Sprache: MS Visual-C V1.5  (QuickWin App.)
*/

#include <stdio.h>
#include <math.h>

#define EPS      1E-10
#define HMIN     5E-3

/* Definition der Ableitung nach der Aufgabenstellung:
    y'=f(x,y)
*/
double f(double x, double y)
{
   return x * y;
}

int main ()
{ double x0;    /* Intervallstart */
  double xn;    /* Intervallende */
  double y0;    /* Anfangswert y0 bei x0 */
  double h;     /* Schrittweite */
  double xi,yi; /* Errechnete Werte im Schritt #i */
  int i;

  printf("Numerische Lösung der Differenzialgleichung mit dem Runge-Kutta Verfahren:\n");
  printf("Intervallbeginn x0: "); scanf("%lg",&x0);
  printf("Intervallende xn: "); scanf("%lg",&xn);
  printf("Anfangswert f(x0)=y0: "); scanf("%lg",&y0);
  printf("Startschrittweite: "); scanf("%lg",&h);
  printf("Lösungsfunktion: \n"
                 "  i         h             x        y\n");

  yi = y0; xi=x0; i = 0;
  do
   { double a,b,c,d;   /* Runge-Kutta Koeffizienten */
     double k;

     i++;                /* Durchlaufzaehler erhoehen */
     a = f(xi,yi);
     b = f(xi + h / 2.0, yi + h * a / 2.0);
     c = f(xi + h / 2.0, yi + h * b / 2.0);
     d = f(xi + h, yi + h * c);
     yi = yi + h * (a + 2 * b + 2 * c + d) / 6.0;
     xi = xi + h;
     printf("%3d\t%8.3f\t%6.3f\t%12.8f\n",i, h, xi,yi );

     /* Schrittweitensteuerung */
     k = 2.0 * fabs((c-b)/(b-a));
     if (k < 0.01) h *=2;
     if (k > 0.08) h /=2;

     /* Ueberlaufkontrollen und Begrenzungen */
     if (fabs(b-a) < EPS) b = a + EPS;
     if (h < HMIN) h = HMIN;
     if (xi+h > xn) h = xn - xi;
    } while ( xi < xn);
  return 0;
 }
```

8.7.13 Testbeispiele zum Runge-Kutta Verfahren mit variabler Schrittweite

Die nachfolgenden Beispiele stammen in neu gerechneter Form aus [Kausen, S. 191-193] und zeigen typisch Betrachtungen zur numerischen Lösung von DGL:

1. Lösung der Differenzialgleichung $y'=x\,y$ auf dem Intervall [0,1] mit dem Anfangswert $y(0)=1$ und der Startschrittweite $h=0.01$:

```
Numerische Lösung der Differenzialgleichung mit dem Runge-Kutta Verfahren:
Intervallbeginn x0: 0
Intervallende xn: 1
Anfangswert f(x0)=y0: 1
Startschrittweite: 0.01
Lösungsfunktion:
  i         h            x        y
  1        0.010         0.010    1.00005000
  2        0.020         0.030    1.00045010
  3        0.040         0.070    1.00245300
  4        0.080         0.150    1.01131352
  5        0.160         0.310    1.04922311
  6        0.160         0.470    1.11678046
  7        0.160         0.630    1.21951089
  8        0.080         0.710    1.28666019
  9        0.080         0.790    1.36622281
 10        0.080         0.870    1.46001958
 11        0.080         0.950    1.57027353
 12        0.050         1.000    1.64872098
```

Ferner betrachten wir die unterschiedlichen Ergebnisse y_n für verschiedene Startschrittweiten für dieselbe Anfangswertaufgabe:

	Start-h	n	h_{min}	h_{max}	Resultat
mit Schrittsteuerung:	0.001	14	0.001	0.128	1.64872097
	0.01	12	0.01	0.16	1.64872098
	0.05	11	0.05	0.1	1.64872115
	0.1	8	0.05	0.2	1.64872025
	0.2	8	0.05	0.2	1.64872064
Ohne Schrittsteuerung:	0.05	20			1.64872126
	0.1	10			1.64872101
	0.2	5			1.64871668

2. Besonders vorteilhaft wirkt sich die Schrittweitensteuerung bei grösseren Intervallen aus. Wir betrachten dieselbe Aufgabe aber im Intervall [0,4]:

	Start-h	n	h_{min}	h_{max}	Resultat
mit Schrittsteuerung:	0.001	137	0.001	0.16	2980.95404728
	0.01	135	0.01	0.16	2980.95410334
	0.05	150	0.0125	0.1	2980.95531111
	0.1	146	0.0125	0.2	2980.95368186
	0.2	146	0.125	0.2	2980.95440027
Ohne Schrittsteuerung:	0.01	400			2980.95782217
	0.05	80			2980.86589981
	0.1	40			2979.67718964
	0.2	20			2965.46119870

Wir sehen, dass die Resultate bei gleichem Rechenaufwand mit der Schrittweitensteuerung deutlich besser sind. Die Startschrittweite hat nur einen relativ geringen Einfluss auf die Gesamtzahl der Schritte.

3. Wir betrachten die Differenzialgleichung:

   $y'=x\,e^{y}$ *mit* $y(0)=1$

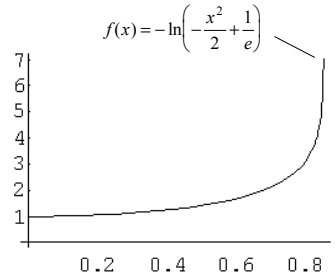
   Die analytische Lösung erfolgt mit Separation der Variablen und wir erhalten als allgemeine Lösung:

$$y=-\ln(-\frac{x^2}{2}+c)$$

   Durch Einsetzen der Anfangsbedingung $y(0)=1$ erhalten wir $c=1/e$ wird die Lösungsfunktion:

$$y=-\ln(-\frac{x^2}{2}+\frac{1}{e})$$

Diese Funktion hat an der Stelle $x=\sqrt{\frac{2}{e}}\approx 0.857763884...$einen Pol:



Durch die Schrittweitenbegrenzung wird ab x=0.75 mit der Minimalschrittweite h_{min}=0.005 gerechnet. Hätte man keine Minimalschrittweite vorgegeben, so würde das Programm wegen ständiger Schrittweitenhalbierung in einer Endlosschleife laufen. Bei x=0.86 bricht das Programm wegen zu grossem y-Wert der Polstelle ab:

```
Numerische Lösung der Differenzialgleichung mit dem Runge-Kutta Verfahren:
Intervallbeginn x0: 0
Intervallende xn: 1
Anfangswert f(x0)=y0: 1
Startschrittweite: 0.01
Lösungsfunktion:
  i         h            x        y
  1        0.010         0.01     1.0001359
  2        0.020         0.03     1.001224
  3        0.040         0.07     1.0066821
........
 25        0.010         0.75     2.4461207
 26        0.005         0.755    2.4905244
 27        0.005         0.76     2.5373079
 28        0.005         0.765    2.5867202
 29        0.005         0.77     2.6390514
 30        0.005         0.775    2.694643
........
 46        0.005         0.855    6.0449689
 47        0.005         0.86     6082751.5
 48        0.005         0.865  7.7525516e+305
```

8.8 Aufgaben

1. Welche wichtige Bedingung muss an die DGL gestellt werden, damit sie überhaupt mit den vorgängig gezeigten numerischen Verfahren gelöst werden kann?
   Zeigen Sie eine DGL, die beispielsweise nicht mit Euler lösbar ist.

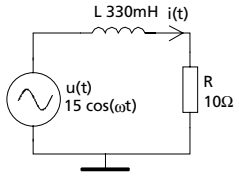
Euler, Taylorreihenverfahren *n*-ter Ordnung, Heun

2. Wie lauten die Hauptgleichung für den Lösungsalgorithmus, wenn ein Taylorverfahren n-ter Ordnung auf die folgenden DGL angewandt werden soll?

   a.) $y'=y+e^{y}$ $\quad n=4$

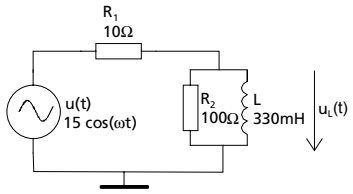
   b.) $y'=y^2-\cos x$ $\quad n=5$

3. Wie lautet die Differenzialgleichung und die Lösungsfunktion für den Strom *i(t)* in der nachfolgenden Schaltung im Intervall [0,25ms] und *n*=10 für Netzfrequenz mit dem Verfahren nach Euler?



4. Dieselbe Frage, aber die Quelle liefert eine Rechteckspannung.

5. Wie lautet die Differenzialgleichung und die Lösungsfunktion für die Spannung $u_L(t)$ in der nachfolgenden Schaltung im Intervall [0,25ms] und *n*=10 für Netzfrequenz mit dem Verfahren nach Euler?



Runge-Kutta Methode

6. Formen Sie die nachfolgenden DGL so um, dass sie in der Runge-Kutta Formel eingesetzt werden können:

   a.) $y+2yy'-y'=0$
   b.) $\log x'=t^2-x^2$
   c.) $(y')^2(1-x^2)=y$

7. Lösen Sie die Differenzialgleichung $y'=-y^2x$ mit Anfangswert $y(0)=1$ für x=-0.2 indem Sie *einen* Runge-Kutta Rechenschritt durchführen.

8. Zur numerischen Lösung ist es notwendig die Ableitungsfunktionen explizit in C zu codieren. Formulieren Sie die Ableitungsfunktionen konkret so, dass im Programm `RKSYS.C` verwendet werden können:

   a.) $y'=x^2+xy'-2yy'$
   b.) $y'=e^x+y'\cos y+x^2$

Ordnungsreduktion

9. Man formuliere die durch Ordnungsreduktion entstehenden Gleichungssysteme für die folgenden Differenzialgleichungen:

   a.) $y''=-5y'+6y-e^x$
   b.) $y''+y'+y^3=0$

10. Lösen Sie die DGL $y''-y'+xy=0$ mit der Methode von Euler im Intervall [0,1] mit n=5 und den Anfangswerten:

a.) $y(0) = 2, y'(0) = 0$
b.) $y(0)= 1, y'(0)= 1$

Programmieraufgaben
Alle Aufgaben sind mit einer Unterteilung n=10 durchzuführen, falls nicht anders verlangt.

11. Lösen Sie das Anfangswertproblem $y'=\sin y+\cos x$ im Intervall [2,5] und dem Anfangswert $y(2)$=0.32 mit der Methode von Euler.

12. Bestimmen Sie $y(1)$ mit den Verfahren von Euler und Heun bei einer Schrittweite von h=0.1 und ermitteln Sie den Fehler zur exakten Lösung, wenn gilt:

a.) $y'=1+y^2$ $y(0)=0$ Hinweis: $1+\tan^2 y=\sec^{2y}$
b.) $y'=(1+x)^{-1}y$ $y(0)=1$

Die exakten Lösungen der Differenzialgleichungen lauten *tan y* resp. *1+y*.

13. Entwickeln Sie ein Programm nach der Methode Taylorreihen n-ter Ordnung das die Lösung für die DGL $y'=e^x y$ mit y(2)=1 auf dem Intervall [0,2] bestimmt. Berücksichtigen Sie dafür Terme bis zu h^4.

14. Bestimmen Sie die Lösungen mit der Methode von Euler und h=0.01:

a.) $y'=1+y^2$	$y(0)=1$	Intervall: [0,0.9]
b.) $y'=y-x$	$y(1)=1$	Intervall: [1,1.75]
c.) $y'=xy+x^2y^2$	$y(2)=-0.6396625333$	Intervall: [2,5]

15. Eine Runge-Kutta Methode fünfter Ordnung lautet:

$$y(x+h) = y(x) + \frac{1}{24}F_1 + \frac{5}{48}F_4 + \frac{27}{56}F_5 + \frac{125}{336}F_6$$

Die zugehörigen Koeffizienten F_i sind nachfolgend definiert:

$$F_1 = hf(x,y)$$
$$F_2 = hf\left(x+\frac{1}{2}h, y+\frac{1}{2}F_1\right)$$
$$F_3 = hf\left(x+\frac{1}{2}h, y+\frac{1}{4}F_1+\frac{1}{4}F_2\right)$$
$$F_4 = hf\left(x+h, y-F_2+F_3\right)$$
$$F_5 = hf\left(x+\frac{2}{3}h, y+\frac{7}{27}F_1+\frac{10}{27}F_2+\frac{1}{27}F_4\right)$$
$$F_6 = hf\left(x+\frac{1}{5}h, y+\frac{28}{625}F_1-\frac{1}{5}F_2+\frac{546}{625}F_3+\frac{54}{625}F_4-\frac{378}{625}F_5\right)$$

Implementieren Sie diese Methode und testen Sie sie mit geeigneten Beispielen aus.

16. Lösen Sie das Anfangswertproblem $y' = y\sqrt{y^2-1}$ mit $y(0)=1.0$ mit RK4 im Intervall $0 \le x \le 1.6$ und überlegen Sie ob dabei irgendwelche Schwierigkeiten zu erwarten sind. Anschliessend lösen Sie dieselbe DGL rückwärts im gleichen Intervall mit negativen h indem Sie den Anfangswert $y(1.6)=1.0$ nehmen.

17. Lösen Sie die Differenzialgleichung $x - y' + 2xy = 0$ im Intervall [0,10] mit RK4 und $h=0.1$ mit EXCEL. Vergleichen Sie die Resultate mit der exakten Lösung $\frac{1}{2}\left(e^{x^2}-1\right)$. Zeichnen Sie den Graph der Lösungsfunktion in linearer und logarithmischer Darstellung.