

## Zusammenfassung vom 27.04.2020

### Numerische Integration: Gauss-Legendre Integration

- zu integrierende Funktion wird durch ein Polynom sehr hoher Ordnung approximiert
- durch geschickte Wahl der Stützstellen (Nullstellen eines orthogonalen Polynoms) findet man, dass dieses Integral exakt integriert werden kann

$$\int_a^b dx \, w(x) f(x) \approx \int_a^b dx \, w(x) p(x) = \sum_{i=0}^{n-1} \omega_i p(x_i) \quad \forall p \in \Pi_{2n-1}$$

- die Gewichte sind hier alle positiv

## 4. Gewöhnliche Differentialgleichungen

**Differentialgleichungen** (DGL) sind Basis vieler physikalischer Probleme. Oft tauchen auch **gewöhnliche DGL** auf (beispielsweise die Bewegungsgleichungen in der Mechanik)

Wir betrachten zunächst DGL einer Variable (=gewöhnliche DGL).

Die **Variable** werden wir oft als **“Zeit”  $t$**  bezeichnen.

Typisch sind zwei Arten von Problemstellungen:

- **Anfangswertproblem**: der Funktionswert ist zu einem **bestimmten “Zeitpunkt”  $t_0$**  gegeben und die DGL beschreibt die Veränderung der Funktion ausgehend von diesem Zeitpunkt.
- **Randwertproblem**: Funktionswerte sind teilweise zu verschiedenen Zeiten vorgegeben und eine Lösung soll diese **Randbedingungen** erfüllen.

Wir betrachten zunächst das Anfangswertproblem.

Wir nehmen ein **System von DGL** erster Ordnung an.

$$\dot{\mathbf{y}}(t) \equiv \frac{d}{dt} \mathbf{y}(t) = \mathbf{f}(t, \mathbf{y}(t))$$

$\mathbf{y}(t), \mathbf{f}(t, \mathbf{y}(t)), \dots$  sind dabei vektorwertige Ausdrücke

Eine **DGL höherer Ordnung** lässt sich als System aus **DGL erster Ordnung** schreiben, z.B.

$$\ddot{y}(t) + a(t) \dot{y}(t) + b(t)y(t) + c(t) = 0$$

lässt sich mit den Definitionen  $y^{(1)}(t) \equiv \dot{y}(t)$  und  $y^{(2)}(t) \equiv \ddot{y}(t)$  als

$$\dot{\mathbf{y}}(t) = \begin{pmatrix} \dot{y}^{(1)}(t) \\ \dot{y}^{(2)}(t) \end{pmatrix} = \begin{pmatrix} y^{(2)}(t) \\ -a(t)y^{(2)}(t) - b(t)y^{(1)}(t) - c(t) \end{pmatrix} \equiv \begin{pmatrix} f^{(1)}(t, y^{(1)}(t), y^{(2)}(t)) \\ f^{(2)}(t, y^{(1)}(t), y^{(2)}(t)) \end{pmatrix} \equiv \mathbf{f}(t, \mathbf{y}(t))$$

schreiben.

D.h. dass es genügt sich Systeme aus DGL erster Ordnung anzusehen.

Wir geben außer der DGL auch die Anfangsbedingung  $\mathbf{y}(t_0) = \mathbf{y}_0$  vor.

Gesucht ist  $\mathbf{y}(t)$  auf dem Intervall  $[a, b]$  mit  $t_0 \in [a, b]$

Wir führen äquidistante Stützstellen  $t_j = t_0 + j h$  mit  $j = 0, \dots, N-1$ ,  $t_0 = a$  und  $h = \frac{b-a}{N-1}$  ein. Wir möchten eine Näherung  $\mathbf{y}_n$  an den Funktionswert  $\mathbf{y}(t_n)$  bestimmen.

Formal erhält man den Funktionswert  $\mathbf{y}(t_{n+1})$  im Schritt  $n+1$  durch Integration

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \int_{t_n}^{t_{n+1}} dt \mathbf{f}(t, \mathbf{y}(t))$$

Somit wird die Differentialgleichung in eine äquivalente Integralgleichung überführt.

Aber  $\mathbf{y}(t)$  ist vor Lösung des Problems nicht bekannt!

Wir können trotzdem eine mehrdimensionale Taylor-Entwicklung zur Approximation der Funktion  $\mathbf{f}$  um ein festes  $(t_n, \mathbf{y}_n)$  machen.

Dazu betrachten wir für die Komponente  $\mu$  vom Vektor  $\mathbf{f}$  die Funktion  $g^{(\mu)}(t) = f^{(\mu)}(t, \mathbf{y}(t))$ , wobei  $\mathbf{y}(t)$  die Lösung der DGL ist.

Wir entwickeln  $g^{(\mu)}(t)$  um  $t_n$ :

$$g^{(\mu)}(t) = f^{(\mu)}(t, \mathbf{y}(t)) = f^{(\mu)}(t_n, \mathbf{y}(t_n)) + (t - t_n) \left. \frac{d}{dt} f^{(\mu)}(t, \mathbf{y}(t)) \right|_{t=t_n} + O(h^2)$$

Wir führen Indizes ein, die die Funktionsauswertungen abkürzen -z.B. für den ersten Term

$$f^{(\mu)}(t_n, \mathbf{y}(t_n)) \equiv f_n^{(\mu)}$$

Die totale Ableitung erfordert dann partielle Ableitungen:

$$\begin{aligned} \left. \frac{d}{dt} f^{(\mu)}(t, \mathbf{y}(t)) \right|_{t=t_n} &= \underbrace{(\partial_t f^{(\mu)})(t, \mathbf{y}(t)) \Big|_{t=t_n}}_{\partial_t f_n^{(\mu)}} + \underbrace{(\partial_\nu f^{(\mu)})(t_n, \mathbf{y}(t_n))}_{\partial_\nu f_n^{(\mu)}} \underbrace{(\partial_t y^{(\nu)})(t) \Big|_{t=t_n}}_{f_n^{(\nu)}} \\ &= \partial_t f_n^{(\mu)} + \left( \partial_\nu f_n^{(\mu)} \right) f_n^{(\nu)} \end{aligned}$$

Wir nehmen die Summenkonvention für doppelte Indizes der partiellen Ableitungen an ( $\nu$  in der Gleichung oben)

In der zur DGL äquivalenten Integralgleichung kann dadurch der Integrand entwickelt werden:

$$\begin{aligned} y_{n+1}^{(\mu)} &= y_n^{(\mu)} + \int_{t_n}^{t_{n+1}} dt f^{(\mu)}(t, \mathbf{y}(t)) \\ &= y_n^{(\mu)} + \int_{t_n}^{t_{n+1}} dt \left[ f_n^{(\mu)} + (t - t_n) \left( \partial_t f_n^{(\mu)} + \left( \partial_\nu f_n^{(\mu)} \right) f_n^{(\nu)} \right) + O(h^2) \right] \\ &= y_n^{(\mu)} + h f_n^{(\mu)} + \frac{h^2}{2} \left( \partial_t f_n^{(\mu)} + \left( \partial_\nu f_n^{(\mu)} \right) f_n^{(\nu)} \right) + O(h^3) \end{aligned}$$

Einsteinsche Summenkonvention!

Dann sehen wir, dass die einfachste Näherung nur Terme bis zur Ordnung  $O(h^2)$  erfordert

$$y_{n+1}^{(\mu)} = y_n^{(\mu)} + h f_n^{(\mu)} + O(h^2)$$

Auf der Basis erhält man das **Euler-Cauchy Verfahren** zur Lösung einer DGL und wir können unseren ersten Löser für gewöhnliche DGL implementieren.

Das **Euler-Cauchy** Verfahren:

Wir wollen die Lösung der DGL

$$\dot{y}(t) = -t y(t) \equiv f(t)$$

finden, wobei die Anfangsbedingung als

$$y(t_0 = 0) = y_0 = 1$$

gegeben ist. Dann ist  $f_0 = -t_0 y_0 = 0$ .

Im ersten Schritt berechnen wir

$$y_1 = y_0 + h f_0$$

wobei  $y_0$  und  $f_0$  bekannt sind. Dann haben wir  $y_1 (= 1)$  und  $f_1 (= -h)$ , aus denen

wir  $y_2$  und  $f_2$  in dem nächsten Schritt berechnen, und so weiter.

```
/* Datei: beispiel-4.1.c    Datum: 23.4.2012 */
```

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<float.h>
#include<limits.h>
```

## Implementierung des Euler-Cauchy Verfahrens

### 1. Teil: rechte Seite der DGL wird als C Funktion zur Verfügung gestellt.

```
/* Routine, die rechte Seite der Dgl definiert.
   neq: Anzahl der Gleichungen
   t : "Zeit" zur der rechte Seite benoetigt ist
   y : Loesung y zu dieser "Zeit" (Feld der Laenge neq)
   f : Ergebnis fuer die rechte Seite (Ausgabe) (Feld der Laenge neq)
*/

void derhs(int neq,double t,double *y,double *f)
{
    /* sicherstellen, dass Anzahl der Gleichungssystem wie erwartet ist */
    if(neq!=1)
    {
        printf("neq passt nicht!\n");
        abort();
    }

    /* es wird angenommen, dass die aufrufende Funktion den Speicherplatz f
       bereitstellt
       hier     $dy/dt = -t * y(t)$  */

    f[0]=-t*y[0];
}
```



```
/* Datei: beispiel-4.1.c    Datum: 23.4.2012 */
```

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<float.h>
#include<limits.h>
```

## Implementierung des Euler-Cauchy Verfahrens

### 1. Teil: rechte Seite der DGL wird als C Funktion zur Verfügung gestellt.

```
/* Routine, die rechte Seite der Dgl definiert.
   neq: Anzahl der Gleichungen
   t : "Zeit" zur der rechte Seite benoetigt ist
   y : Loesung y zu dieser "Zeit" (Feld der Laenge neq)
   f : Ergebnis fuer die rechte Seite (Ausgabe) (Feld der Laenge neq)
*/
```

```
void derhs(int neq,double t,double *y,double *f)
{
    /* sicherstellen, dass Anzahl der Gleichungssystem wie
       if(neq!=1)
       {
           printf("neq passt nicht!\n");
           abort();
       }
    */
```

```
/* es wird angenommen, dass die aufrufende Funktion den Speicherplatz f
   bereitstellt
   hier    dy/dt = - t * y(t) */
```

```
f[0]=-t*y[0];
```

```
}
```

**f**

- hat *neq* Komponenten
- ist abhängig von *t*
- hängt von *neq* Komponenten *y* ab
- wird als Feld zurückgegeben

```
/* Datei: beispiel-4.1.c    Datum: 23.4.2012 */
```

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<float.h>
#include<limits.h>
```

## Implementierung des Euler-Cauchy Verfahrens

### 1. Teil: rechte Seite der DGL wird als C Funktion zur Verfügung gestellt.

```
/* Routine, die rechte Seite der Dgl definiert.
   neq: Anzahl der Gleichungen
   t : "Zeit" zur der rechte Seite benoetigt ist
   y : Loesung y zu dieser "Zeit" (Feld der Laenge neq)
   f : Ergebnis fuer die rechte Seite (Ausgabe) (Feld der Laenge neq)
*/
```

```
void derhs(int neq,double t,double *y,double *f)
{
    /* sicherstellen, dass Anzahl der Gleichungssystem wie
       if(neq!=1)
       {
           printf("neq passt nicht!\n");
           abort();
       }
    */
```

```
/* es wird angenommen, dass die aufrufende Funktion den Speicherplatz f
   bereitstellt
   hier    dy/dt = - t * y(t) */
```

```
f[0]=-t*y[0];
```

```
}
```

**f**

- hat *neq* Komponenten
- ist abhängig von *t*
- hängt von *neq* Komponenten *y* ab
- wird als Feld zurückgegeben

hier:  $y'(t) = -t y(t)$  (eine Komponente)

```

/* Schritt nach dem Euler-Verfahren
   neq: Anzahl der Gleichungen
   h   : Schrittweite
   t   : "Zeit" beim Start
   y   : Loesung bei t , (Aufruf)
        Loesung bei t+h, (Rueckgabe)
        Feld der Laenge neq
   f   : Hilfsfeld der Laenge neq
   derhs : Zeiger auf Funktion, die rechte Seite bestimmt
*/

```

## 2. Teil: Schritt nach dem Euler-Cauchy Verfahren

**“euler” startet mit  $y$  bei  $t$  und führt Schritt nach  $t+h$  durch.**

```

void euler(int neq, double h, double t, double *y, double *f,
           void (*derhs)(int, double, double *, double*))
{
    int i;
    /* rechte Seite bestimmen f(t,y(t)) und bei f speichern */

    (*derhs)(neq,t,y,f);

    /* Schritt einfaches Eulerverfahren:
       y(t+h) = y(t) + h * f(t,y(t)) */

    for(i=0;i<neq;i++)
    {
        y[i]+=h*f[i];
    }
}

```

$$y_{n+1}^{\mu} = y_n^{\mu} + h f_n^{\mu} + \mathcal{O}(h^2)$$

### 3. Teil: Hauptprogramm mit Ausgabe

```
int main()
{
    double h,t0,y0,tend,tstep; /* Schrittweite, startpunkt, Startwert Endpunkt, Schritt fuer Ausgabe */
    int neq=1;                /* feste Vorgabe der Anzahl der Gleichungen */
    double exact,diff;        /* Variablen, um Ergebnis zu speichern und vergleichen*/
    double *y,*f;             /* Zeiger auf Speicherplaetze, die double enthalten */
    double t,tprint,eps=1.0E-4;

    /* Eingabe der Parameter */
    printf("Bitte geben Sie h,t0,y0,tend und tstep ein: \n");
    scanf(" %le %le %le %le %le",&h,&t0,&y0,&tend,&tstep);

    y=malloc(sizeof(double)*neq); /* malloc reserviert Speicher fuer Feld mit y Werten und Hilfsfeld */
    f=malloc(sizeof(double)*neq);

    printf("\n   %20s %20s %20s %20s \n","t","exact","dgl","diff");

    y[0]=y0;
    tprint=t0;
    for(t=t0;t<=tend;t+=h)
    { if(t-tprint>=eps) /* Naechsten Ausgabepunkt erreicht?*/
        { exact=exp(-t*t*0.5);          /* known exact value */
          diff=fabs(exact-y[0])/exact;   /* and rel. error */
          printf("   %20.5le %20.5le %20.5le %20.5le \n",t,exact,y[0],diff);
          tprint+=tstep; /* Ausgabe und naechsten Punkt bestimmen */
        }
        /* Dgl.schritt ausfuehren */
        euler(neq,h,t,y,f,&(derhs));
    }
    free(y); free(f); }
```

### 3. Teil: Hauptprogramm mit Ausgabe

#### Deklarationen

```
int main()
{
    double h,t0,y0,tend,tstep; /* Schrittweite, startpunkt, Startwert Endpunkt, Schritt fuer Ausgabe */
    int neq=1;                /* feste Vorgabe der Anzahl der Gleichungen */
    double exact,diff;        /* Variablen, um Ergebnis zu speichern und vergleichen*/
    double *y,*f;             /* Zeiger auf Speicherplaetze, die double enthalten */
    double t,tprint,eps=1.0E-4;

    /* Eingabe der Parameter */
    printf("Bitte geben Sie h,t0,y0,tend und tstep ein: \n");
    scanf(" %le %le %le %le %le",&h,&t0,&y0,&tend,&tstep);

    y=malloc(sizeof(double)*neq); /* malloc reserviert Speicher fuer Feld mit y Werten und Hilfsfeld */
    f=malloc(sizeof(double)*neq);

    printf("\n   %20s %20s %20s %20s \n","t","exact","dgl","diff");

    y[0]=y0;
    tprint=t0;
    for(t=t0;t<=tend;t+=h)
    { if(t-tprint>=-eps) /* Naechsten Ausgabepunkt erreicht?*/
        { exact=exp(-t*t*0.5); /* known exact value */
          diff=fabs(exact-y[0])/exact; /* and rel. error */
          printf("   %20.5le %20.5le %20.5le %20.5le \n",t,exact,y[0],diff);
          tprint+=tstep; /* Ausgabe und naechsten Punkt bestimmen */
        }
        /* Dgl.schritt ausfuehren */
        euler(neq,h,t,y,f,&(derhs));
    }
    free(y); free(f); }
```

### 3. Teil: Hauptprogramm mit Ausgabe

#### Deklarationen

#### Eingabe der Startwerte und Zeitpunkte für Ausgabe der Funktion

```
int main()
{
    double h,t0,y0,tend,tstep; /* Schrittweite, startpunkt, Startwert Endpunkt, Schritt fuer Ausgabe */
    int neq=1;                /* feste Vorgabe der Anzahl der Gleichungen */
    double exact,diff;        /* Variablen, um Ergebnis zu speichern und vergleichen*/
    double *y,*f;             /* Zeiger auf Speicherplaetze, die double enthalten */
    double t,tprint,eps=1.0E-4;

    /* Eingabe der Parameter */
    printf("Bitte geben Sie h,t0,y0,tend und tstep ein: \n");
    scanf(" %le %le %le %le %le",&h,&t0,&y0,&tend,&tstep);

    y=malloc(sizeof(double)*neq); /* malloc reserviert Speicher fuer Feld mit y Werten und Hilfsfeld */
    f=malloc(sizeof(double)*neq);

    printf("\n   %20s %20s %20s %20s \n","t","exact","dgl","diff");

    y[0]=y0;
    tprint=t0;
    for(t=t0;t<=tend;t+=h)
    { if(t-tprint>=-eps) /* Naechsten Ausgabepunkt erreicht?*/
        { exact=exp(-t*t*0.5); /* known exact value */
          diff=fabs(exact-y[0])/exact; /* and rel. error */
          printf("   %20.5le %20.5le %20.5le %20.5le \n",t,exact,y[0],diff);
          tprint+=tstep; /* Ausgabe und naechsten Punkt bestimmen */
        }
        /* Dgl.schritt ausfuehren */
        euler(neq,h,t,y,f,&(derhs));
    }
    free(y); free(f); }
```

### 3. Teil: Hauptprogramm mit Ausgabe

```
int main()
{
    double h,t0,y0,tend,tstep; /* Schrittweite, startpunkt, Startwert Endpunkt, Schritt fuer Ausgabe */
    int neq=1;                /* feste Vorgabe der Anzahl der Gleichungen */
    double exact,diff;        /* Variablen, um Ergebnis zu speichern und vergleichen */
    double *y,*f;             /* Zeiger auf Speicherplaetze, die double enthalten */
    double t,tprint,eps=1.0E-4;
```

#### Deklarationen

```
/* Eingabe der Parameter */
printf("Bitte geben Sie h,t0,y0,tend und tstep ein: \n");
scanf(" %le %le %le %le %le",&h,&t0,&y0,&tend,&tstep);
```

#### Eingabe der Startwerte und Zeitpunkte für Ausgabe der Funktion

```
y=malloc(sizeof(double)*neq); /* malloc reserviert Speicher fuer Feld mit y Werten und Hilfsfeld */
f=malloc(sizeof(double)*neq);
```

#### Speicher für Zwischenwerte

```
printf("\n    %20s %20s %20s %20s \n","t","exact","dgl","diff");

y[0]=y0;
tprint=t0;
for(t=t0;t<=tend;t+=h)
{ if(t-tprint>=-eps) /* Naechsten Ausgabepunkt erreicht? */
    { exact=exp(-t*t*0.5); /* known exact value */
      diff=fabs(exact-y[0])/exact; /* and rel. error */
      printf("    %20.5le %20.5le %20.5le %20.5le \n",t,exact,y[0],diff);
      tprint+=tstep; /* Ausgabe und naechsten Punkt bestimmen */
    }
    /* Dgl.schritt ausfuehren */
    euler(neq,h,t,y,f,&(derhs));
}
free(y); free(f); }
```

### 3. Teil: Hauptprogramm mit Ausgabe

```
int main()
{
    double h,t0,y0,tend,tstep; /* Schrittweite, startpunkt, Startwert Endpunkt, Schritt fuer Ausgabe */
    int neq=1; /* feste Vorgabe der Anzahl der Gleichungen */
    double exact,diff; /* Variablen, um Ergebnis zu speichern und vergleichen */
    double *y,*f; /* Zeiger auf Speicherplaetze, die double enthalten */
    double t,tprint,eps=1.0E-4;
```

#### Deklarationen

```
/* Eingabe der Parameter */
printf("Bitte geben Sie h,t0,y0,tend und tstep ein: \n");
scanf(" %le %le %le %le %le",&h,&t0,&y0,&tend,&tstep);
```

#### Eingabe der Startwerte und Zeitpunkte für Ausgabe der Funktion

```
y=malloc(sizeof(double)*neq); /* malloc reserviert Speicher fuer Feld mit y Werten und Hilfsfeld */
f=malloc(sizeof(double)*neq);
```

#### Speicher für Zwischenwerte

```
printf("\n %20s %20s %20s %20s \n","t","exact","dgl","diff");
```

#### Schleife durchläuft alle "Zeiten" in Schritten von $h$

```
y[0]=y0;
tprint=t0;
for(t=t0;t<=tend;t+=h)
{ if(t-tprint>=-eps) /* Naechsten Ausgabepunkt erreicht? */
    { exact=exp(-t*t*0.5); /* known exact value */
      diff=fabs(exact-y[0])/exact; /* and rel. error */
      printf(" %20.5le %20.5le %20.5le %20.5le \n",t,exact,y[0],diff);
      tprint+=tstep; /* Ausgabe und naechsten Punkt bestimmen */
    }
    /* Dgl.schritt ausfuehren */
    euler(neq,h,t,y,f,&(derhs));
}
free(y); free(f); }
```

#### Führe euler Schritt aus



### 3. Teil: Hauptprogramm mit Ausgabe

```
int main()
{
    double h,t0,y0,tend,tstep; /* Schrittweite, startpunkt, Startwert Endpunkt, Schritt fuer Ausgabe */
    int neq=1;                /* feste Vorgabe der Anzahl der Gleichungen */
    double exact,diff;        /* Variablen, um Ergebnis zu speichern und vergleichen */
    double *y,*f;             /* Zeiger auf Speicherplaetze, die double enthalten */
    double t,tprint,eps=1.0E-4;
```

**Deklarationen**

```
/* Eingabe der Parameter */
printf("Bitte geben Sie h,t0,y0,tend und tstep ein: \n");
scanf(" %le %le %le %le %le",&h,&t0,&y0,&tend,&tstep);
```

**Eingabe der Startwerte und Zeitpunkte für Ausgabe der Funktion**

```
y=malloc(sizeof(double)*neq); /* malloc reserviert Speicher fuer Feld mit y Werten und Hilfsfeld */
f=malloc(sizeof(double)*neq);
```

**Speicher für Zwischenwerte**

**Schleife durchläuft alle "Zeiten" in Schritten von  $h$**

```
printf("\n   %20s %20s %20s %20s \n","t","exact","dgl","diff");
```

```
y[0]=y0;
tprint=t0;
for(t=t0;t<=tend;t+=h)
{ if(t-tprint>=-eps) /* Naechsten Ausgabepunkt erreicht? */
    { exact=exp(-t*t*0.5); /* known exact value */
      diff=fabs(exact-y[0])/exact; /* and rel. error */
      printf("   %20.5le %20.5le %20.5le %20.5le \n",t,exact,y[0],diff);
      tprint+=tstep; /* Ausgabe und naechsten Punkt bestimmen */
    }
    /* Dgl.schritt ausfuehren */
    euler(neq,h,t,y,f,&(derhs));
}
free(y); free(f); }
```

**Ausgabe gewünscht?**

**Führe euler Schritt aus**

```
int main()
{
```

### 3. Teil: Hauptprogramm mit Ausgabe

```
double h,t0,y0,tend,tstep; /* Schrittweite, startpunkt, Startwert Endpunkt, Schritt fuer Ausgabe */
int neq=1; /* feste Vorgabe der Anzahl der Gleichungen */
double exact,diff; /* Variablen, um Ergebnis zu speichern und vergleichen */
double *y,*f; /* Zeiger auf Speicherplaetze, die double enthalten */
double t,tprint,eps=1.0E-4;
```

#### Deklarationen

```
/* Eingabe der Parameter */
printf("Bitte geben Sie h,t0,y0,tend und tstep ein: \n");
scanf(" %le %le %le %le %le",&h,&t0,&y0,&tend,&tstep);
```

#### Eingabe der Startwerte und Zeitpunkte für Ausgabe der Funktion

```
y=malloc(sizeof(double)*neq); /* malloc reserviert Speicher fuer Feld mit y Werten und Hilfsfeld */
f=malloc(sizeof(double)*neq);
```

```
printf("\n %20s %20s %20s %20s \n","t","exact","dgl","diff");
```

#### Speicher für Zwischenwerte

```
y[0]=y0;
tprint=t0;
for(t=t0;t<=tend;t+=h)
{ if(t-tprint>=-eps) /* Naechsten Ausgabepunkt erreicht? */
{ exact=exp(-t*t*0.5); /* known exact value */
diff=fabs(exact-y[0])/exact; /* and rel. error */
printf(" %20.5le %20.5le %20.5le %20.5le \n",t,exact,y[0],diff);
tprint+=tstep; /* Ausgabe und naechsten Punkt bestimmen */
}
/* Dgl.schritt ausfuehren */
euler(neq,h,t,y,f,&(derhs));
}
```

#### Schleife durchläuft alle "Zeiten" in Schritten von $h$

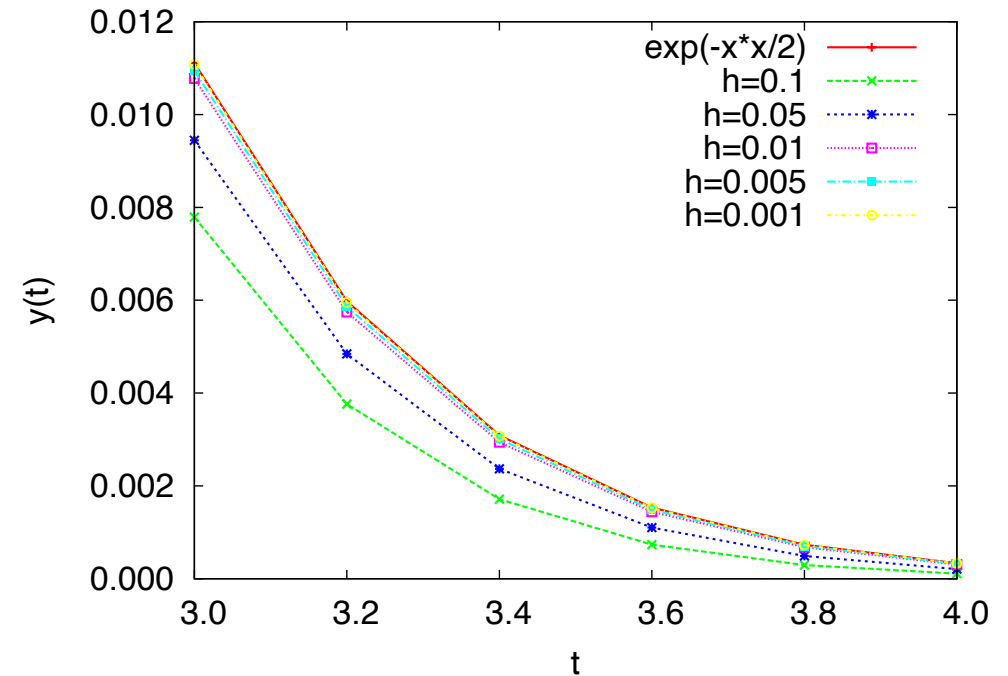
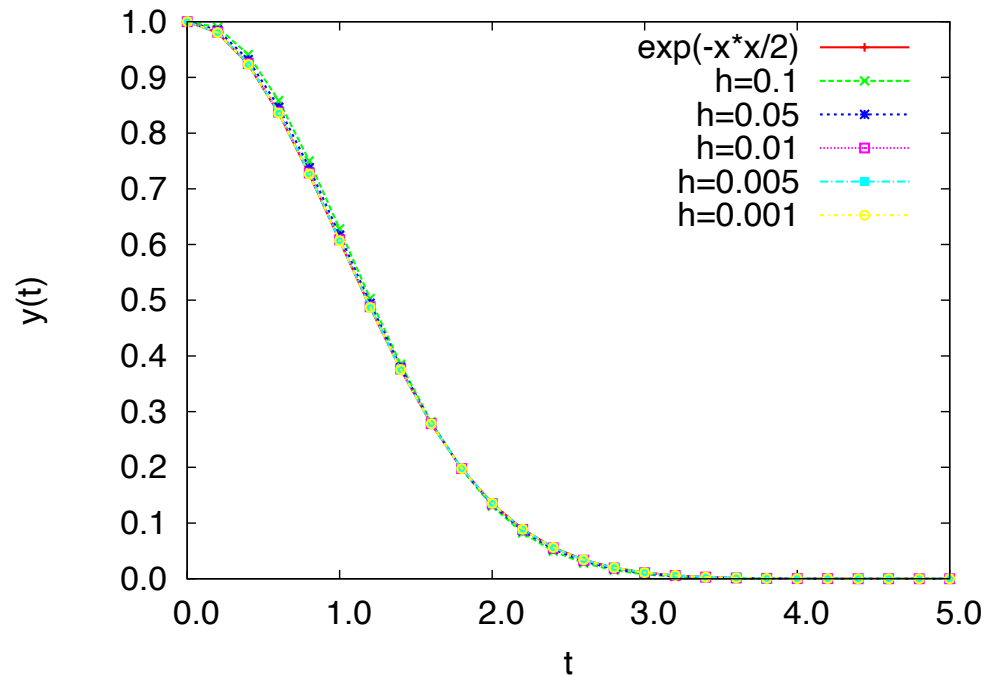
#### Ausgabe gewünscht?

#### Führe euler Schritt aus

```
free(y); free(f); }
/* Ausgabe: Bitte geben Sie h,t0,y0,tend und tstep ein: 0.01 0.0 1.0 4.0 1.0
```

t	exact	dgl	diff
0.00000e+00	1.00000e+00	1.00000e+00	0.00000e+00
1.00000e+00	6.06531e-01	6.08566e-01	3.35565e-03
2.00000e+00	1.35335e-01	1.34880e-01	3.36144e-03
3.00000e+00	1.11090e-02	1.07757e-02	2.99989e-02
4.00000e+00	3.35463e-04	3.07068e-04	8.46438e-02 */

## Welche Genauigkeiten erhält man mit dem Euler-Cauchy Verfahren?



- für unser Beispiel sind die Ergebnisse generell gut
- beachte Lösung bei **großen  $t$**  und die relativ schlechte Genauigkeit
- Konvergenz für verschiedene  $h = 0.1, \dots, 0.001$
- welche Genauigkeiten ergeben sich bei höheren Ordnungen: **Runge-Kutta Verfahren**
- betrachten später auch noch **Stabilität** der Methode

Ausgangspunkt ist wieder das Integral  $\mathbf{y}_{n+1} = \mathbf{y}_n + \int_{t_n}^{t_{n+1}} dt \mathbf{f}(t, \mathbf{y}(t))$

Durch Entwicklung des Integranden um  $t_{n+\frac{1}{2}} \equiv t_n + \frac{h}{2}$  (*Mittelpunkt!*) findet man

$$y_{n+1}^{(\mu)} = y_n^{(\mu)} + \int_{t_n}^{t_{n+1}} dt \left[ f^{(\mu)}(t, \mathbf{y}(t)) \Big|_{t=t_{n+\frac{1}{2}}} + (t - t_{n+\frac{1}{2}}) \frac{d}{dt} f^{(\mu)}(t, \mathbf{y}(t)) \Big|_{t=t_{n+\frac{1}{2}}} + O(h^2) \right]$$

$$= y_n^{(\mu)} + h f_{n+\frac{1}{2}}^{(\mu)} + O(h^3)$$

Der Term  $O(h^2)$  fällt wegen der symmetrischen Integrationsgrenze heraus!

Nun brauchen wir  $f_{n+\frac{1}{2}}^{(\mu)}$  nur bis zur Ordnung  $O(h)$  zu kennen, um das Integral bis zur Ordnung  $O(h^2)$  zu erhalten. Dafür reicht uns ein halber Schritt mit Euler-Cauchy.

Mit

$$y_{n+\frac{1}{2}}^{(\mu)} = y_n^{(\mu)} + \frac{h}{2} f^{(\mu)}(t_n, \mathbf{y}_n) + O(h^2) \equiv y_n^{(\mu)} + \frac{k^{(\mu)}}{2} + O(h^2)$$

erhält man dann


$$f_{n+\frac{1}{2}}^{(\mu)} = f^{(\mu)}(t_{n+\frac{1}{2}}, \mathbf{y}_{n+\frac{1}{2}}) = f^{(\mu)}(t_{n+\frac{1}{2}}, \mathbf{y}_n + \frac{\mathbf{k}}{2}) + O(h^2)$$

Ausgangspunkt ist wieder das Integral  $\mathbf{y}_{n+1} = \mathbf{y}_n + \int_{t_n}^{t_{n+1}} dt \mathbf{f}(t, \mathbf{y}(t))$

Durch Entwicklung des Integranden um  $t_{n+\frac{1}{2}} \equiv t_n + \frac{h}{2}$  (*Mittelpunkt!*) findet man

$$y_{n+1}^{(\mu)} = y_n^{(\mu)} + \int_{t_n}^{t_{n+1}} dt \left[ f^{(\mu)}(t, \mathbf{y}(t)) \Big|_{t=t_{n+\frac{1}{2}}} + (t - t_{n+\frac{1}{2}}) \frac{d}{dt} f^{(\mu)}(t, \mathbf{y}(t)) \Big|_{t=t_{n+\frac{1}{2}}} + O(h^2) \right]$$

$$= y_n^{(\mu)} + h f_{n+\frac{1}{2}}^{(\mu)} + O(h^3)$$


 $f_{n+\frac{1}{2}}^{(\mu)} = f^{(\mu)}\left(t_{n+\frac{1}{2}}, \mathbf{y}_{n+\frac{1}{2}}\right) \approx f^{(\mu)}\left(t_{n+\frac{1}{2}}, \mathbf{y}\left(t_{n+\frac{1}{2}}\right)\right)$

Der Term  $O(h^2)$  fällt wegen der symmetrischen Integrationsgrenze heraus!

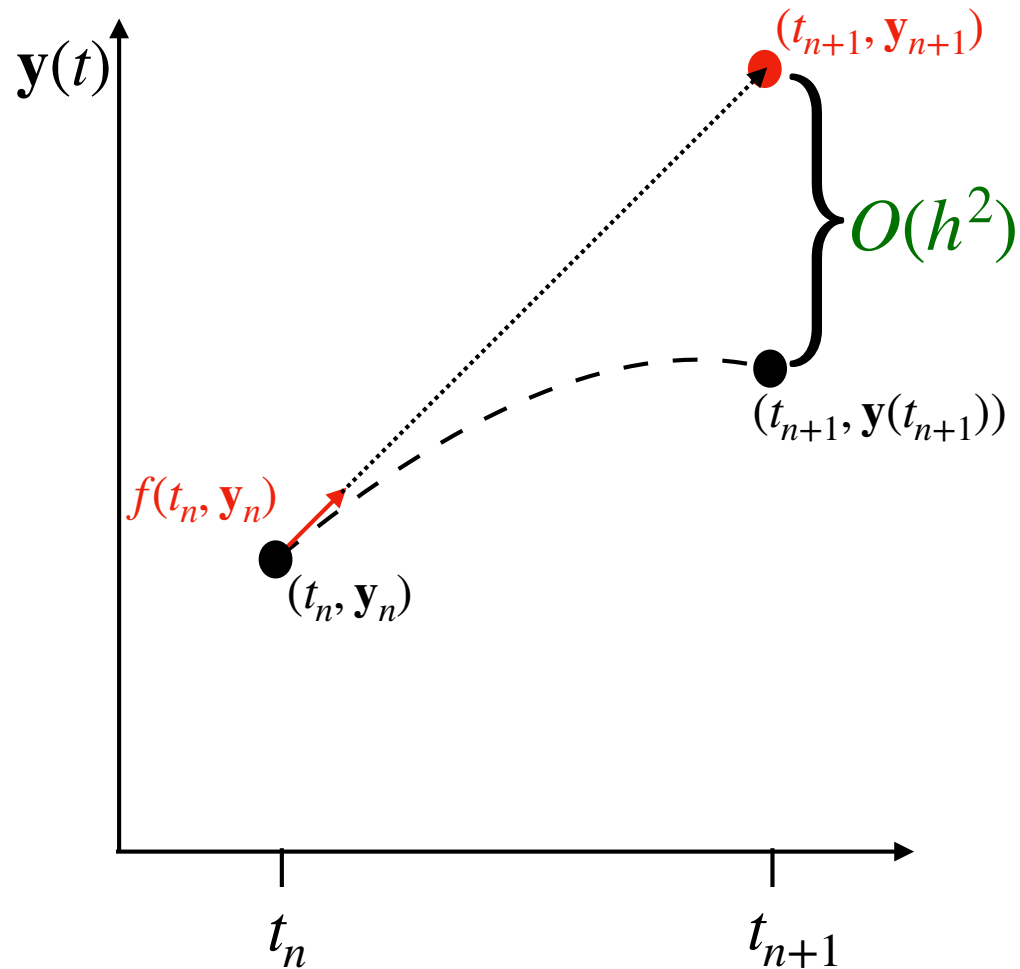
Nun brauchen wir  $f_{n+\frac{1}{2}}^{(\mu)}$  nur bis zur Ordnung  $O(h)$  zu kennen, um das Integral bis zur Ordnung  $O(h^2)$  zu erhalten. Dafür reicht uns ein halber Schritt mit Euler-Cauchy.

Mit

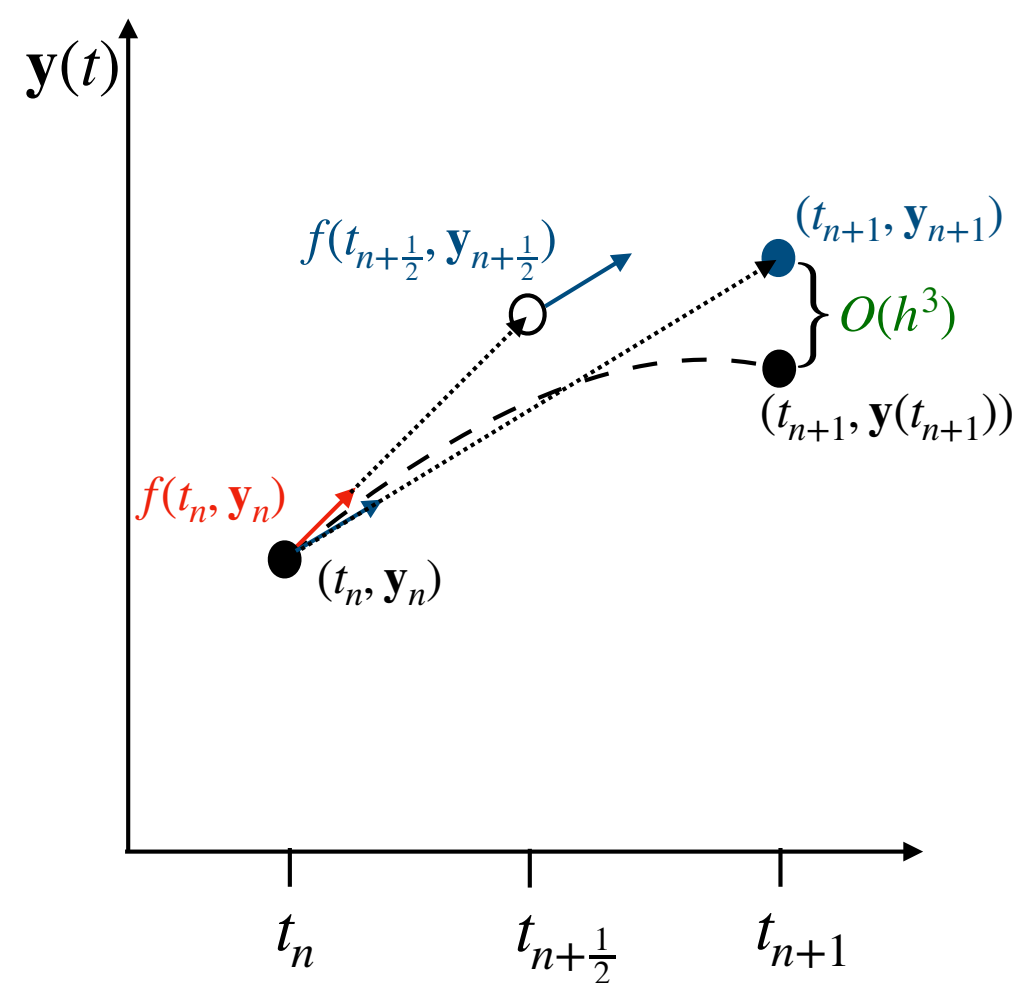
$$y_{n+\frac{1}{2}}^{(\mu)} = y_n^{(\mu)} + \frac{h}{2} f^{(\mu)}(t_n, \mathbf{y}_n) + O(h^2) \equiv y_n^{(\mu)} + \frac{\mathbf{k}^{(\mu)}}{2} + O(h^2)$$

erhält man dann

$$f_{n+\frac{1}{2}}^{(\mu)} = f^{(\mu)}(t_{n+\frac{1}{2}}, \mathbf{y}_{n+\frac{1}{2}}) = f^{(\mu)}(t_{n+\frac{1}{2}}, \mathbf{y}_n + \frac{\mathbf{k}}{2}) + O(h^2)$$

**Euler-Cauchy-Verfahren**

$$y_{n+1} = y_n + h \mathbf{f}(t_n, y_n)$$

**verbessertes Euler-Verfahren (RK2)**

$$\begin{aligned} y_{n+1} &= y_n + h \mathbf{f}\left(t_{n+\frac{1}{2}}, y_{n+\frac{1}{2}}\right) \\ &= y_n + h \mathbf{f}\left(t_{n+\frac{1}{2}}, y_n + \frac{\mathbf{k}}{2}\right) \end{aligned}$$

Eingesetzt ist das Integral findet man damit das

## Runge-Kutta Verfahren 2. Ordnung (verbessertes Euler-Verfahren)

$$\mathbf{k} = h \mathbf{f}(t_n, \mathbf{y}_n) = h \mathbf{f}_n$$

$$y_{n+1}^{(\mu)} = y_n^{(\mu)} + h f^{(\mu)} \left( t_{n+\frac{1}{2}}, \mathbf{y}_n + \frac{\mathbf{k}}{2} \right) + O(h^3)$$

Überprüfen kann man die Ordnung durch den Vergleich mit der allgemeinen Taylor-Entwicklung um  $(t_n, \mathbf{y}_n)$ . Für das Runge-Kutta Verfahren findet man

$$y_{n+1}^{(\mu)} = y_n^{(\mu)} + h f_n^{(\mu)} + h \left( \frac{h}{2} \partial_t f_n^{(\mu)} + \left( \partial_\nu f_n^{(\mu)} \right) \frac{k^{(\nu)}}{2} \right) + O(h^3)$$

$$= y_n^{(\mu)} + h f_n^{(\mu)} + \frac{h^2}{2} \left( \partial_t f_n^{(\mu)} + \left( \partial_\nu f_n^{(\mu)} \right) f_n^{(\nu)} \right) + O(h^3)$$

Das stimmt mit dem allgemeinen Ausdruck bis zur Ordnung  $O(h^2)$  überein. 

## Schritt nach dem Runge-Kutta Verfahren 2. Ordnung

```
/* Datei: beispiel-4.2.c Datum: 24.4.2012 */
```

```
/* ... siehe beispiel-4.1.c */
```

```
/* Schritt nach dem Runge-Kutta-Verfahren 2. Ordnung, Mittelpunkt oder modifiziertes Euler-Verfahren
```

```
neq: Anzahl der Gleichungen
```

```
h : Schrittweite
```

```
t : "Zeit" beim Start
```

```
y : Loesung bei t , (Aufruf)  
    Loesung bei t+h, (Rueckgabe)  
    Feld der Laenge neq
```

```
f : Hilfsfeld der Laenge neq
```

```
k : Hilfsfeld der Laenge neq
```

```
derhs : Zeiger auf Funktion, die rechte Seite bestimmt
```

```
*/
```

```
void rkstep(int neq, double h, double t, double *y, double *f, double *k,  
            void (*derhs)(int, double *, double *))
```

```
{ int i;
```

```
    (*derhs)(neq,t,y,k); /* rechte Seite bestimmen f(t,y(t)) und bei k speichern */
```

```
    for(i=0;i<neq;i++) /* k verwenden fuer neues Argument von f k = y_n + k/2 */
```

```
    {  
        k[i]=0.5*h*k[i]+y[i];  
    }
```

```
    (*derhs)(neq,t+0.5*h,k,f); /* rechte Seite bestimmen f(t+h/2,y(t)+k/2) und bei f speichern */
```

```
    for(i=0;i<neq;i++) /* Schritt modifiziertes Eulerverfahren:  $y(t+h) = y(t) + h * f(t+h/2, y(t)+k/2)$  */
```

```
    {  
        y[i]+=h*f[i];  
    }
```

```
}
```

**“rkstep” startet mit  $y$  bei  $t$  und führt Schritt nach  $t+h$  durch  
 $f$  und  $k$  sind Hilfsfelder**



## Schritt nach dem Runge-Kutta Verfahren 2. Ordnung

```
/* Datei: beispiel-4.2.c Datum: 24.4.2012 */
```

```
/* ... siehe beispiel-4.1.c */
```

```
/* Schritt nach dem Runge-Kutta-Verfahren 2. Ordnung, Mittelpunkt oder modifiziertes Euler-Verfahren
```

```
neq: Anzahl der Gleichungen
```

```
h : Schrittweite
```

```
t : "Zeit" beim Start
```

```
y : Loesung bei t , (Aufruf)  
    Loesung bei t+h, (Rueckgabe)  
    Feld der Laenge neq
```

```
f : Hilfsfeld der Laenge neq
```

```
k : Hilfsfeld der Laenge neq
```

```
derhs : Zeiger auf Funktion, die rechte Seite bestimmt
```

```
*/
```

```
void rkstep(int neq, double h, double t, double *y, double *f, double *k,  
            void (*derhs)(int, double *, double *, double*))
```

```
{ int i;
```

```
(*derhs)(neq,t,y,k); /* rechte Seite bestimmen f(t,y(t)) und bei k speichern */
```

```
for(i=0;i<neq;i++) /* k verwenden fuer neues Argument von f k = y_n + k/2 */
```

```
{  
    k[i]=0.5*h*k[i]+y[i];  
}
```

```
(*derhs)(neq,t+0.5*h,k,f); /* rechte Seite bestimmen f(t+h/2,y(t)+k/2) und bei f speichern */
```

```
for(i=0;i<neq;i++) /* Schritt modifiziertes Eulerverfahren: y(t+h) = y(t) + h * f(t+h/2,y(t)+k/2) */
```

```
{  
    y[i]+=h*f[i];  
}
```

```
}
```

**“rkstep” startet mit  $y$  bei  $t$  und führt Schritt nach  $t+h$  durch  
 $f$  und  $k$  sind Hilfsfelder**

$$\mathbf{k} = h \mathbf{f}(t_n, \mathbf{y}_n) = h \mathbf{f}_n$$

## Schritt nach dem Runge-Kutta Verfahren 2. Ordnung

```
/* Datei: beispiel-4.2.c Datum: 24.4.2012 */
```

```
/* ... siehe beispiel-4.1.c */
```

```
/* Schritt nach dem Runge-Kutta-Verfahren 2. Ordnung, Mittelpunkt oder modifiziertes Euler-Verfahren
```

```
neq: Anzahl der Gleichungen
```

```
h : Schrittweite
```

```
t : "Zeit" beim Start
```

```
y : Loesung bei t , (Aufruf)  
    Loesung bei t+h, (Rueckgabe)  
    Feld der Laenge neq
```

```
f : Hilfsfeld der Laenge neq
```

```
k : Hilfsfeld der Laenge neq
```

```
derhs : Zeiger auf Funktion, die rechte Seite bestimmt
```

```
*/
```

```
void rkstep(int neq, double h, double t, double *y, double *f, double *k,  
            void (*derhs)(int, double *, double *))
```

```
{ int i;
```

```
(*derhs)(neq,t,y,k); /* rechte Seite bestimmen f(t,y(t)) und bei k speichern */
```

```
for(i=0;i<neq;i++) /* k verwenden fuer neues Argument von f k = y_n + k/2 */
```

```
{  
    k[i]=0.5*h*k[i]+y[i];  
}
```

```
(*derhs)(neq,t+0.5*h,k,f); /* rechte Seite bestimmen f(t+h/2,y(t)+k/2) und bei f speichern */
```

```
for(i=0;i<neq;i++) /* Schritt modifiziertes Eulerverfahren: y(t+h) = y(t) + h * f(t+h/2,y(t)+k/2) */
```

```
{  
    y[i]+=h*f[i];  
}
```

```
}
```

**“rkstep” startet mit  $y$  bei  $t$  und führt Schritt nach  $t+h$  durch  
 $f$  und  $k$  sind Hilfsfelder**

$$\mathbf{k} = h \mathbf{f}(t_n, \mathbf{y}_n) = h \mathbf{f}_n$$

$$y_{n+1}^{(\mu)} = y_n^{(\mu)} + h f^{(\mu)}\left(t_{n+\frac{1}{2}}, y_n + \frac{\mathbf{k}}{2}\right) + O(h^3)$$

```

int main()
{ ...
  double *y,*f,*k;          /* Zeiger auf
  ...
  /* malloc reserviert Speicher fuer Feld mit y Werten und Hilfsfeld */
  y=malloc(sizeof(double)*neq);
  k=malloc(sizeof(double)*neq);
  f=malloc(sizeof(double)*neq);
  ...
  for(t=t0;t<=tend;t+=h)
  {
    exact=exp(-t*t*0.5);      /* known exact value */
    diff=fabs(exact-y[0])/exact; /* and rel. error */

    if(t-tprint>=-eps) /* Naechsten Ausgabepunkt erreicht?*/
    {
      printf("    %20.5le %20.5le %20.5le %20.5le \n",t,exact,y[0],diff);
      tprint+=tstep; /* Ausgabe und naechsten Punkt bestimmen */
    }

    /* Dgl.schritt ausfuehren */
    rkstep(neq,h,t,y,f,k,&(derhs));

  }

  free(k); free(y); free(f);
}

```

## Hauptprogramm mit Ausgabe fast identisch zum Euler-Cauchy Verfahren

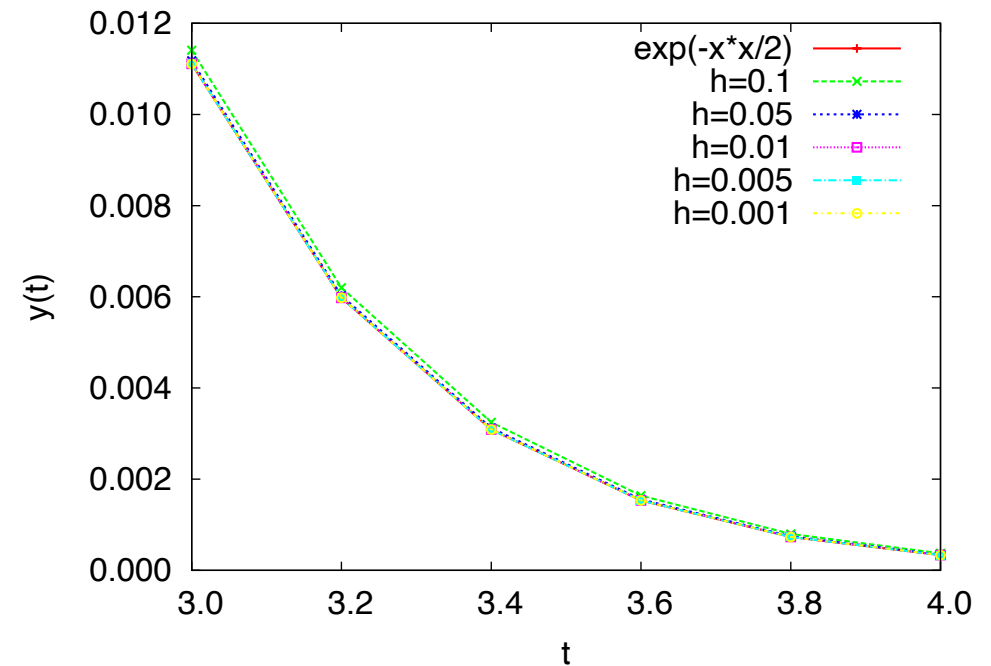
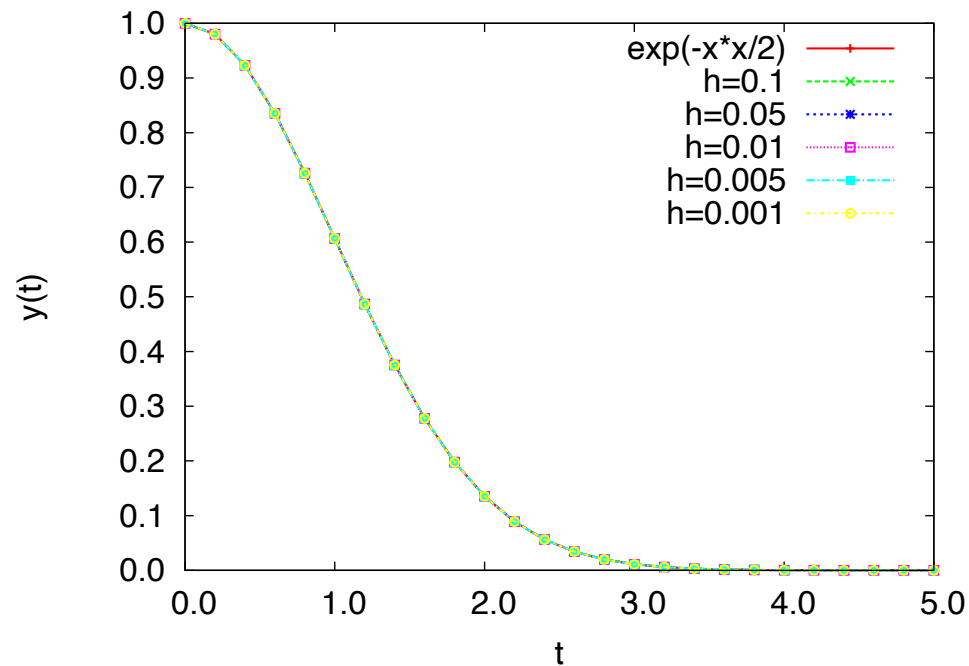
wende hier Runge-Kutta 2. Ordnung an

```

/* Ergebnis: Bitte geben Sie h,t0,y0,tend und tstep ein:
0.01 0.0 1.0 4.0 1.0

```

t	exact	dgl	diff
0.00000e+00	1.00000e+00	1.00000e+00	0.00000e+00
1.00000e+00	6.06531e-01	6.06526e-01	8.39207e-06
2.00000e+00	1.35335e-01	1.35338e-01	1.67996e-05
3.00000e+00	1.11090e-02	1.11115e-02	2.28885e-04
4.00000e+00	3.35463e-04	3.35760e-04	8.87585e-04 */



- beachte recht gute Genauigkeit bei **großen  $t$**
- Konvergenz für verschiedene  $h = 0.1, \dots, 0.001$  deutlich schneller als bei Euler-Cauchy
- andere Wahl der Koeffizienten bei gleicher Ordnung ist möglich
- und es gibt höherer Ordnungen