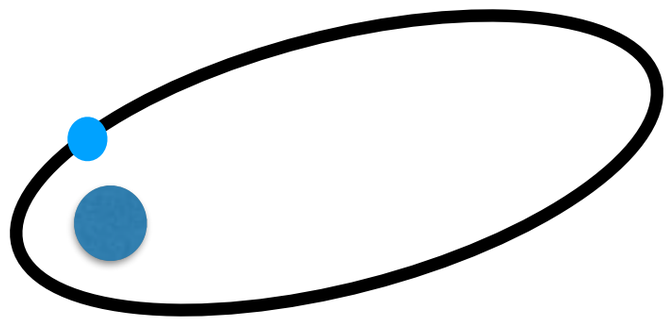
Schrittweitensteuerung

Bisher haben wir immer angenommen, dass die Schrittweite beim Lösen der DGL in jedem Schritt gleich ist.

Das ist oft nicht effizient, beispielsweise bei Planeten- oder Satellitenbewegungen ist die Geschwindigkeit beim Perihel oft deutlich größer. Wir erwarten, dass dort kleinere Schrittweiten erforderlich sind, als in der Nähe des Aphel.



Ziel ist nun die **optimale Schrittweite** automatisch zu ermitteln.

Ausgangspunkt ist ein **Verfahren der Ordnung *k*** und zwei Approximationen für $\mathbf{y}(t_n + h)$

Anzahl der Schritte	1	2
Schrittweite	h	$h/2$
nächste Näherung	$\mathbf{y}_{n+1,1}$	$\mathbf{y}_{n+1,2}$

Die exakte Lösung nach einem Schritt $\mathbf{y}(t + h)$ und die Approximation $\mathbf{y}_{n+1,1}$ unterscheiden sich durch

$$\mathbf{y}(t_n + h) - \mathbf{y}_{n+1,1} = h^{k+1}\Delta$$

wobei Δ in erster Approximation eine Konstante im Intervall [*t*, *t+h*] ist.

Wenn wir Δ kennen, erhalten wir eine recht gute Abschätzung des Fehlers.

Idee ist nun, die Lösung mit dem gleichen Verfahren in zwei halben Schritten zu erhalten.

Nach dem ersten halben Schritt haben wir

$$\mathbf{y}(t_n + h/2) = \mathbf{y}_{n+\frac{1}{2}} + \left(\frac{h}{2}\right)^{k+1} \Delta$$

Die Näherung $\mathbf{y}_{n+1,2}$ wird aus der Näherung $\mathbf{y}_{n+\frac{1}{2}}$ mithilfe des Verfahrens $\mathbf{T}$ berechnet:

$$\mathbf{y}_{n+1,2} = \mathbf{y}_{n+\frac{1}{2}} + \mathbf{T}(t_{n+1/2}, \mathbf{y}_{n+\frac{1}{2}})$$

und mithilfe der vorigen Gleichung bekommt man

$$\mathbf{y}_{n+1,2} = \mathbf{y}(t_n + h/2) - \left(\frac{h}{2}\right)^{k+1} \Delta + \mathbf{T}(t_{n+1/2}, \mathbf{y}_{n+\frac{1}{2}})$$

Die exakte Lösung $\mathbf{y}(t_n + h)$ kann auch anhand der Integrationsgleichung berechnet werden:

$$\mathbf{y}(t_n + h) = \mathbf{y}(t_n + h/2) + \mathbf{T}(t_{n+1/2}, \mathbf{y}(t_n + h/2)) + \left(\frac{h}{2}\right)^{k+1} \Delta$$

Dann setzen wir aus dieser Gleichung $\mathbf{y}(t_n + h/2)$ aus der vorigen Gleichung ein, und bekommen wir

$$\mathbf{y}(t_n + h) = \mathbf{y}_{n+1,2} + 2\left(\frac{h}{2}\right)^{k+1} \Delta + \underbrace{\mathbf{T}(t_{n+1/2}, \mathbf{y}(t_n + h/2)) - \mathbf{T}(t_{n+1/2}, \mathbf{y}_{n+\frac{1}{2}})}_{O(h^{k+2})}$$

$$\Longrightarrow \quad \mathbf{y}(t_n + h) - \mathbf{y}_{n+1,2} \approx 2\left(\frac{h}{2}\right)^{k+1} \Delta$$

Unter der Annahme, dass Δ in beiden Fällen gleich ist, findet man

$$\left|\mathbf{y}_{n+1,1} - \mathbf{y}_{n+1,2}\right| \approx \frac{2^k - 1}{2^k} h^{k+1} |\Delta|$$

oder

$$|\Delta| = \left|\mathbf{y}_{n+1,1} - \mathbf{y}_{n+1,2}\right| \frac{2^k}{h^{k+1}(2^k - 1)}$$

und das kann man nutzen, um den Fehler ε_h bei einem Schritt abzuschätzen

$$\varepsilon_h \equiv \left|\mathbf{y}(t_n + h) - \mathbf{y}_{n+1,1}\right| \approx h^{k+1} |\Delta| \approx \frac{2^k}{2^k - 1} \left|\mathbf{y}_{n+1,1} - \mathbf{y}_{n+1,2}\right|$$

Einer gewünschten Genauigkeit ε_{min} entspricht die Schrittweite $h_{max} = \left(\varepsilon_{min}/|\Delta|\right)^{\frac{1}{k+1}}$, für die man gerade noch den Fehler klein genug hält.

Da $|\Delta| \approx \varepsilon_h / h^{k+1}$, kann h_{max} abgeschätzt werden:

$$h_{max} = h \left|\frac{\varepsilon_{min}}{\varepsilon_h}\right|^{\frac{1}{k+1}} \Longrightarrow \frac{h_{max}}{h} = \left|\frac{\varepsilon_{min}}{\frac{2^k}{2^k - 1}\left|\mathbf{y}_{n+1,1} - \mathbf{y}_{n+1,2}\right|}\right|^{\frac{1}{k+1}}$$

Das kann man in einem Code leicht zur **Schrittweitensteuerung** nutzen

- benutze derzeitiges *h* um $\mathbf{y}_{n+1,1}$ und $\mathbf{y}_{n+1,2}$ zu erhalten
- schätze daraus h_{max}
- wenn $h < h_{max}$ wird $\mathbf{y}_{n+1,1}$ akzeptiert und führe nächsten Schritt mit $h=h_{max}$ aus
- wenn $h > h_{max}$ wird $\mathbf{y}_{n+1,1}$ *nicht akzeptiert* und führe den selben Schritt mit kleineren *h* (z.B. $h=h_{max}$) aus

Beispielprogramm eines einfachen DGL Lösers mit Schrittweitensteuerung

```
/* Datei: beispiel-4.3.c Datum: 03.05.2016 */

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

/* 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)
*/
```

wie in vorherigen Code wird die DGL in derhs definiert

```
void derhs(int neq,double t,double *y,double *f)
{
  /* sicherstellen, dass Anzahl der Gleichungssystem   wie erwartet */
  if(neq!=2)
    {
      printf("neq passt nicht!\n");
      abort();
    }
  /* es wird angenommen, dass die aufrufende Funktion den Speicherplatz f
     bereitstellt hier   dy/dt = (y_1(t),-t*(1-cos(t)^2)*y_0(t)) */
  f[0]=y[1];
  f[1]=-t*(1-cos(t)*cos(t))*y[0];
}
```

$$\mathbf{y} \equiv \begin{pmatrix} y \\ \dot{y} \end{pmatrix} = \begin{pmatrix} y^{(1)} = \mathbf{y}[0] \\ y^{(2)} = \mathbf{y}[1] \end{pmatrix} \quad \Longrightarrow \quad \dot{\mathbf{y}} = \begin{pmatrix} \dot{y} = \mathbf{y}[1] \\ \ddot{y} \end{pmatrix} \equiv \begin{pmatrix} f^{(1)} = \mathbf{f}[0] \\ f^{(2)} = \mathbf{f}[1] \end{pmatrix}$$

$$\ddot{y}(t) = -\,t(1-\cos^2(t))\; y(t)$$

zur Vereinfachung nutzt der Code die Funktion für einen Schritt nach dem Runge-Kutta Verfahren 2. Ordnung

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

```
double delta(double *y1,double *y2,int neq)
{
  double sum;
  int i;

  sum=0.0;
  for(i=0;i<neq;i++)
    {
      sum+=fabs((y1[i]-y2[i]));
    }

  return(sum);

}
```

Funktion definiert den Abstand zweier Lösungen.

Hauptprogramm (ohne Hauptschleife)

```
int main()
{
  double h,t0,y0,y1,tend,tstep,epsmin,epsh,hmax;  /* Schrittweite, startpunkt, Startwert
                                Endpunkt, Schritt fuer Ausgabe, minimal Genauigkeit */
  int neq=2,i;                    /* feste Vorgabe der Anzahl der Gleichungen */
  double *y,*f,*k,*ys1,*ys2;          /* Zeiger auf Speicherplaetze, die double enthalten */
  double t,tprint,eps=1.0E-4,hmin;
  int nstep;
  /* Eingabe der Parameter */
  printf("#Bitte geben Sie h,t0,y0,y1,tend, tstep und Genauigkeit ein: \n");
  scanf(" %le %le %le %le %le %le %le",&h,&t0,&y0,&y1,&tend,&tstep,&epsmin);

  y=malloc(sizeof(double)*neq);  /* malloc reserviert Speicher fuer Feld mit y Werten und
Hilfsfeld */
  ys1=malloc(sizeof(double)*neq);
  ys2=malloc(sizeof(double)*neq);
  k=malloc(sizeof(double)*neq);
  f=malloc(sizeof(double)*neq);

  printf("\n #  %20s %20s %20s %20s\n","t","y","y'","h");

  ...

  printf("Schritte die benoetigt wurden: %d \n",nstep*3);
  printf("Schritte bei konstantem h= %20.5le:   %d \n",hmin,(int)((tend-t0)/hmin));
  printf("Gewinn  %20.5le \n",(tend-t0)/hmin/3.0/nstep);


  free(k);free(y);free(ys1);free(ys2);free(f);
}
```

Deklarationen und Eingabe der Parameter

Speicherbereiche für übliche Hilfsgrößen und $y^{(1)}$ und $y^{(2)}$

Programm gibt Tabelle mit Ergebnis und Anzahl der Schritte (mit/ohne Schrittweitensteuerung) und Effizienzgewinn aus

```
int main()
{  ...

  y[0]=y0; y[1]=y1;
  tprint=t0;
  nstep=0; hmin=h;

  for(t=t0;t<=tend;)  /* Erhoehung von t spaeter */
    { if(t-tprint>=-eps) /* Naechsten Ausgabepunkt erreicht?*/
      { printf("   %20.5le %20.5le %20.5le %20.5le \n",t,y[0],y[1], h );
        tprint+=tstep;  /* Ausgabe und naechsten Punkt bestimmen */ }

      for(i=0;i<neq;i++) { ys1[i]=y[i]; ys2[i]=y[i];}

      rkstep(neq,h,t,ys1,f,k,&(derhs));     /* Dgl.schritt ausfuehren mit Schrittweite h*/
      rkstep(neq,h/2.,t,ys2,f,k,&(derhs)); /* Dgl.schritt zweimal ausfuehren mit Schrittweite h/2*/
      rkstep(neq,h/2.,t+h/2.,ys2,f,k,&(derhs));

      nstep++;   /* ein paar Werte um die Effizienz zu bestimmen */
      hmin=fmin(hmin,h/2.);

      epsh=delta(ys1,ys2,neq)*pow(2.0,2.0)/(pow(2.0,2.0)-1.0);
      hmax=h*pow(epsmin/epsh,1.0/(2.0+1.0));

      if(h<=hmax)
       { t+=h;       /* Schritt akzeptieren: t erhöhen und y
         h=hmax*0.95;
         for(i=0;i<neq;i++){y[i]=ys1[i];} }
      else
        { h=hmax*0.95; /* Schritt wiederholen: t  und y ni
     }
  ...
}
```

Hauptschleife des Hauptprogramms

Anfangswerte, Schrittanzahl und minimale Schrittweite

for Schleife: t wird nicht automatisch erhöht

Ausgabe in definiertem Abstand

derzeitige Lösung nach $y^{(1)}$ und $y^{(2)}$

Runge-Kutta Schritt mit *h* und *h*/2

Statistik …

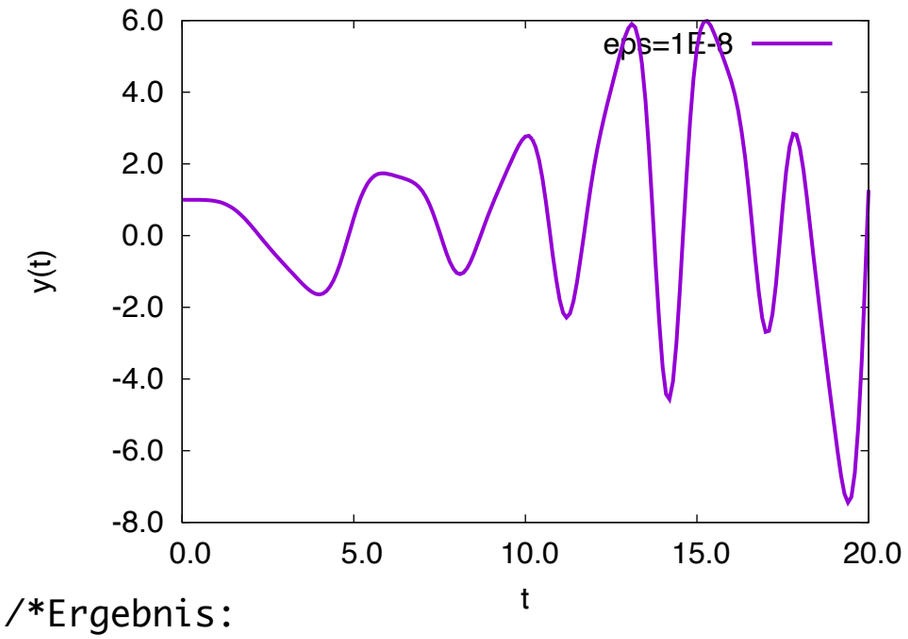
$$\frac{h_{max}}{h} = \left| \frac{\varepsilon_{min}}{\varepsilon_h} \right|^{\frac{1}{k+1}}$$

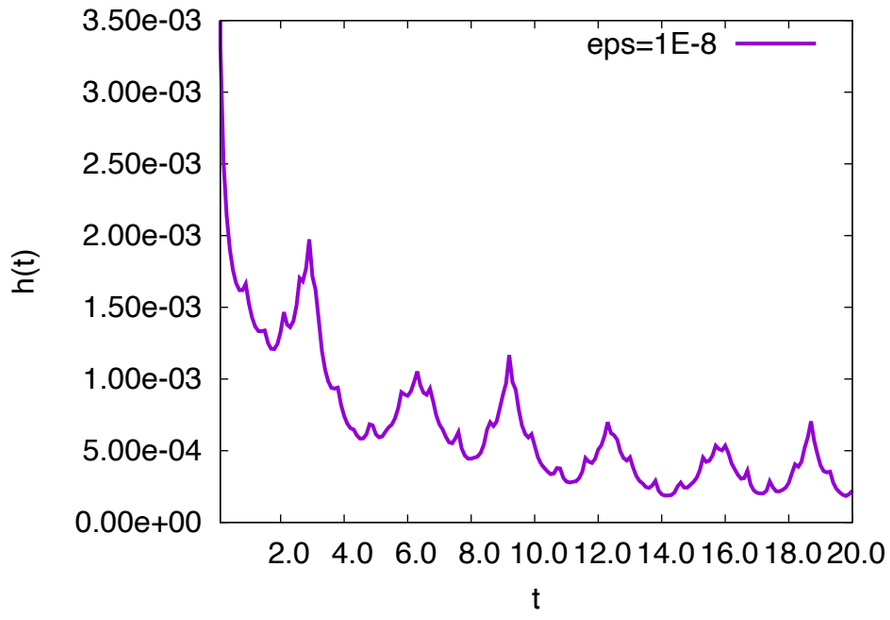
Abschätzung der Genauigkeit und des notwendigen *h*

Schritt akzeptieren: wähle *h* etwas kleiner als notwendig

Schritt nicht genau genug: wähle *h* etwas kleiner als notwendig und wiederhole bei gleichem *t*

Beispiel: $\ddot{y}(t) = -t(1-\cos^2(t))\, y(t) \quad \text{mit} \quad y(0) = 1 \quad \text{und} \quad \dot{y}(0) = 0$






```
/*Ergebnis:
#Bitte geben Sie h,t0,y0,y1,tend, tstep und Genauigkeit ein:
0.1 0 1 0 100 20 1e-8

 #                    t                    y                   y'                   h
            0.00000e+00          1.00000e+00          0.00000e+00          5.00000e-02
            2.00003e+01          1.27875e+00          2.43044e+01          2.21121e-04
            4.00000e+01          1.90823e+01         -3.43570e+01          8.54449e-05
            5.99999e+01          3.60999e+02          4.02597e+02          6.89620e-05
            7.99999e+01          7.48781e+02          3.73996e+03          1.08900e-05
            9.99999e+01         -5.05591e+04         -3.37706e+05          5.00699e-06
Schritte die benoetigt wurden: 6673974
Schritte bei konstantem h=          2.27099e-06:   44033684
Gewinn           6.59782e+00
 */
```

Einfaches Beispiel variiert Schrittweite zur Fehlerabschätzung
Funktionsauswertungen können hier noch optimiert werden

Vergleich verschiedener Ordnung: **Runge-Kutta-Fehlberg** oder **Cash-Karp-Verfahren**