Computergestütztes wissenschaftliches Rechnen
SoSe 2004

Alexander K. Hartmann, Universität Göttingen

21. April 2004

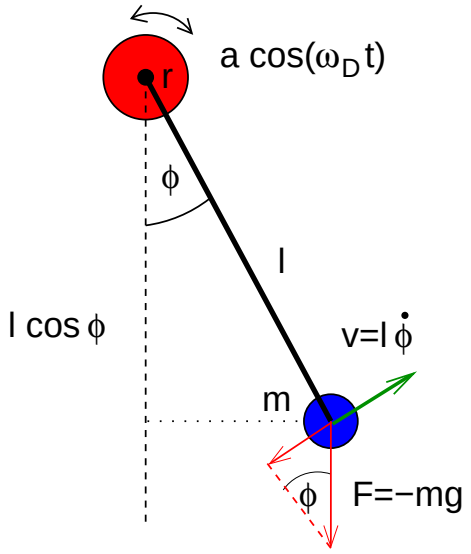
Hinweis zu Übungen:

- Cip-Pool Account besorgen `http://www.cip.physik.uni-goettingen.de/`
- Ab Donnerstag auf der Homepage von Thomas Pruschke `www.theorie.physik.uni-Goettingen.de/~pruschke`: C Kurs und 1. Übungsblatt.

2 Einfache Differentialgleichungen, Numerical Recipes

2.1 Das (geriebene und getriebene) Pendel

Modellsystem für diesen Abschnitt. Ziel: numerische Lösung mit *Numerical Recipes* Bibliothek.



Bewegungsgleichung: $ma = ml\ddot{\phi} = -mg\sin(\phi)$. Dimensionslose Bewegungsgleichung (Kräfte in Einheiten von mg und Zeit in Einheiten von $\sqrt{l/g}$ gemessen):

$$\ddot{\phi} = -\sin\phi \tag{1}$$

Erweiterung: Reibungskraft $-r\dot{\phi}$ und externe periodische Kraft $a\cos\omega_D t$ ergibt:

$$\ddot{\phi} = -\sin\phi - r\dot{\phi} + a\cos\omega_D t \tag{2}$$

2.2 Euler Methode

Wir wollen lösen: k gekoppelte einfache DGLs erster Ordnung für Funktionen $y^{(i)}(x)$ $(i = 1, \ldots, k)$:

$$\begin{aligned} \frac{d}{dx}y^{(1)} &= f^{(1)}(x, y^{(1)}, \ldots, y^{(k)}) \\ &\vdots \\ \frac{d}{dx}y^{(k)} &= f^{(k)}(x, y^{(1)}, \ldots, y^{(k)}) \end{aligned} \tag{3}$$

Umwandlung einer einfachen DGL k. Ordnung

$$\frac{d}{dx}^{n} y = f\left(x, y, \frac{d}{dx}y, \left(\frac{d}{dx}\right)^{2} y, \ldots, \left(\frac{d}{dx}\right)^{k-1} y\right) \tag{4}$$

in k DGLs erster Ordnung mit $y^{(1)} = y$, $y^{(2)} = \frac{d}{dx}y$, ..., $y^{(k)} = \left(\frac{d}{dx}\right)^{k-1} y$

$$\begin{aligned} \frac{d}{dx}y^{(1)} &= y^{(2)} \\ &\vdots \\ \frac{d}{dx}y^{(k)} &= f^{(k)}(x, y^{(1)}, \ldots, y^{(k)}) \end{aligned} \tag{5}$$

Für das Pendel $k = 2$ mit $t \equiv x$, $\phi \equiv y^{(1)}$, $\omega \equiv y^{(2)}$ und den Gleichungen

$$\begin{aligned} \dot{\phi} &= \omega \\ \dot{\omega} &= -r\omega - sin(\phi) + a\cos(w_D t) \end{aligned} \tag{6}$$

Zur Darstellung der Methode: $k = 1$, d.h.

$$y' = f(x, y) \tag{7}$$

Numerische Lösung: Integration vom Startwert $y_0 := y(x_0)$ durch *Diskretisierung* mit Schrittweite h: $x_n = x_0 + nh$. Sei $y_n = y(x_n)$.

Taylorreihenentwicklung:

$$y_{n\pm1} = y(y_n \pm h) = y_n \pm hy'_n + \frac{h^2}{2}y''_n \pm \frac{h^3}{6}y'''_n + \mathcal{O}(h^4) \tag{8}$$

wobei y' mit (7) bestimmt wird.
Euler Methode: bis 1. Ordnung (Fehler $\mathcal{O}(h^2)$)

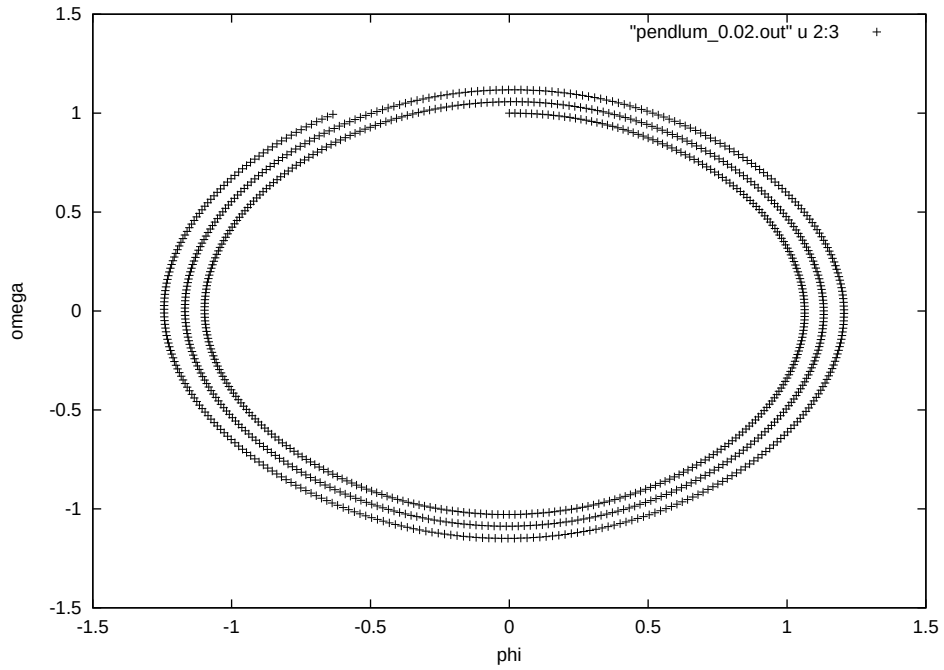
$$y_{n+1} = y_n + hf(x_n, y_n) \tag{9}$$

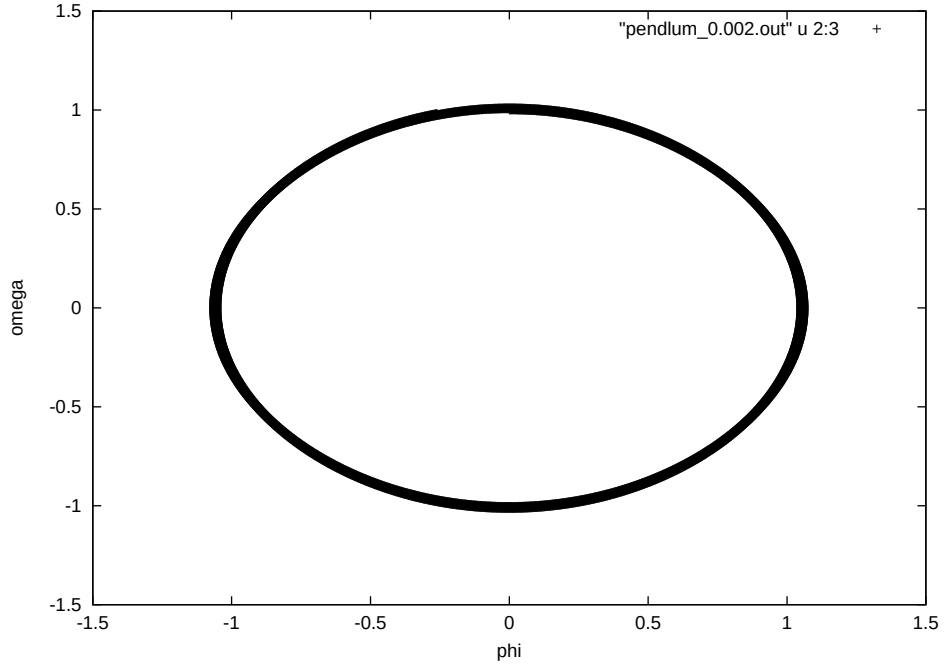
Anwendung auf ungetriebenes Pendel (`pendlum_euler.c`), ohne Reibung ($r = a = 0$):

```
/*** Integrates the pendlum using Euler method    ***/

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

double f1(double t, double phi, double omega)
{
  return(omega);
}

double f2(double t, double phi, double omega)
{
  return(-sin(phi));
}

int main(int argc, char **argv)
{
  double phi, omega;                    /* phase space variables */
  double phi_new, omega_new;           /* variables at next step */
  double h;                                        /* stepsize */
  int n, n_max;                           /* step, maximum step */
  double t;                                             /* time */

  sscanf(argv[1], "%lf", &h);       /* first argument = stepsize */
  sscanf(argv[2], "%d", &n_max); /*2nd argument = numb. of steps */
  phi=0; omega=1.0; t = 0;

  for(n=1; n<=n_max; n++)              /* iterate over all times */
  {
    printf("%f %f %f\n", t, phi, omega);        /* print results */
    phi_new = phi + h*f1(t,phi,omega);              /* integrate */
    omega_new = omega + h*f2(t,phi,omega);
    t += h;
    phi = phi_new; omega = omega_new;
  }
  return(0);
}
```

Informationen zu den Befehlen: man-pages, z.B. `man sscanf` in der Shell eintippen.
Compilieren; `cc -o pendlum_euler pendlum_euler.c -lm`
(`-lm`: Einbinden der mathematischen Funktionen).
Simulation: `pendlum_euler 0.02 1000 > pendlum_0.02.out`
bzw. `pendlum_euler 0.002 10000 > pendlum_0.002.out`

Mit gnuplot anschauen, z.B.`gnuplot> plot "pendlum_0.02.out" u 2:3` ergibt





Großer Diskretisierungsfehler!

Rückblick zur Populationsdynamik:

$$x_{n+1} = 4rx_n(1 - x_n) = f(x_n) \tag{10}$$

Mit $\frac{dx}{dn} \approx x_{n+1} - x_n$ sowie kontinuierliche Fortsetzung: Gleichung ist Euler Integration der DGL

$$\frac{dx}{dn} = (4r - 1)x - 4rx^2 \tag{11}$$

mit Schrittweite $h = 1 \rightarrow$ großer Diskretisierungsfehler (kein numerischer Fehler!). Die richtige Lösung ist:

$$x(n) = \frac{4r - 1}{4r + Ke^{(1-4r)n}} \tag{12}$$

also $x(n) \to 1 - 1/4r =: x^*$ $(r > 1/4)$.Tatsächlich $f(x^*) = x^*$. Kleine Abweichungen: $f(x^* + \epsilon) \approx f(x^*) + f'(x^*)\epsilon = x^* + f'(x^*)\epsilon \to$ da $|f'(x^*)| > 1$ für $r > 3/4 \to$ abstoßender Fixpunkt.[1]

2.3 Runge-Kutta Verfahren

Ziele: höhere Genauigkeit in h. Grundidee: Verwendung der Ableitung an Zwischenpunkten $x_n + h/2$.

Taylorentwicklung bis $\mathcal{O}(h^2)$ von $y(x_n \pm h/2)$ ergibt nach y_n bzw. y_{n+1} aufgelöst

$$\begin{aligned} y_n &= y_{n+1/2} - \frac{h}{2}y'_{n+1/2} + \left(\frac{h}{2}\right)^2 \frac{1}{2}y''_{n+1/2} + \mathcal{O}(h^3) \\ y_{n+1} &= y_{n+1/2} + \frac{h}{2}y'_{n+1/2} + \left(\frac{h}{2}\right)^2 \frac{1}{2}y''_{n+1/2} + \mathcal{O}(h^3) \end{aligned}$$

Subtraktion der Gleichungen ergibt

$$y_{n+1} = y_n + hy'_{n+1/2} + \mathcal{O}(h^3) \tag{13}$$

Gebraucht wird $y_{n+1/2} = f(x_{n+1/2}, y_{n+1/2})$. Näherung: $y_{n+1/2} = f(x_{n+1/2}, y_n + h/2f(x_n, y_n)) + \mathcal{O}(h^2)$, [2] ergibt:

$$\begin{aligned} k_1 &= hf(x_n, y_n) \\ k_2 &= hf(x_n + h/2, y_n + k_1/2) \\ y_{n+1} &= y_n + k_2 \end{aligned}$$

Erhöhung der Zahl der Auswertungsschritte kann sich lohnen, weil man dann ggf. h nicht so klein machen muss.
Bis höhere Ordnung: Standard ist Fehler $\mathcal{O}(h^5)$.
Übungen im Cip-Pool: Ableitung/Implementierung der Methode.

Hier: Verwendung einer Bibliotheksfunktion `rk4()` von *Numerical Recipes* (NR), Kap. 16. Funktion in `nr.h` deklariert und in `rk4.c` implementiert. Man braucht bei NR immer `nrutil.h` und `nrutil.c`. Achtung: bei NR Arrays-Indizes laufen von 1 los → um eines länger reservieren, oder speziellen NR Routinen nehmen.

Routine

```
void rk4(float y[], float dydx[], int k, float x, float h, float yout[],
        void (*derivs)(float, float [], float []));
```

`y[]`: Array von Variablen (Wert an Stelle `x`)
`dydx[]`: Array von Ableitungen (Wert an Stelle `x`)
`k`: Anzahl der Variablen
`x`: x-Wert
`h`: Schrittweite

[1] Genauere Analyse: Für r ein wenig größer als 3/4: stabile Zweierzyklen $f(f(x^*_{1,2})) = x^*_{1,2}$, dann Viererzyklen etc. Ab $r \approx 0.89$: chaotisches Verhalten.

[2] Fehler $\mathcal{O}(h^2)$, da $f(x_{n+1/2}, y_n + h/2f(x_n, y_n) + \mathcal{O}(h^2)) = f(x_{n+1/2}, y_n) + [h/2f(x_n, y_n) + \mathcal{O}(h^2)]\frac{\partial f}{\partial y} + \mathcal{O}(h^2)$

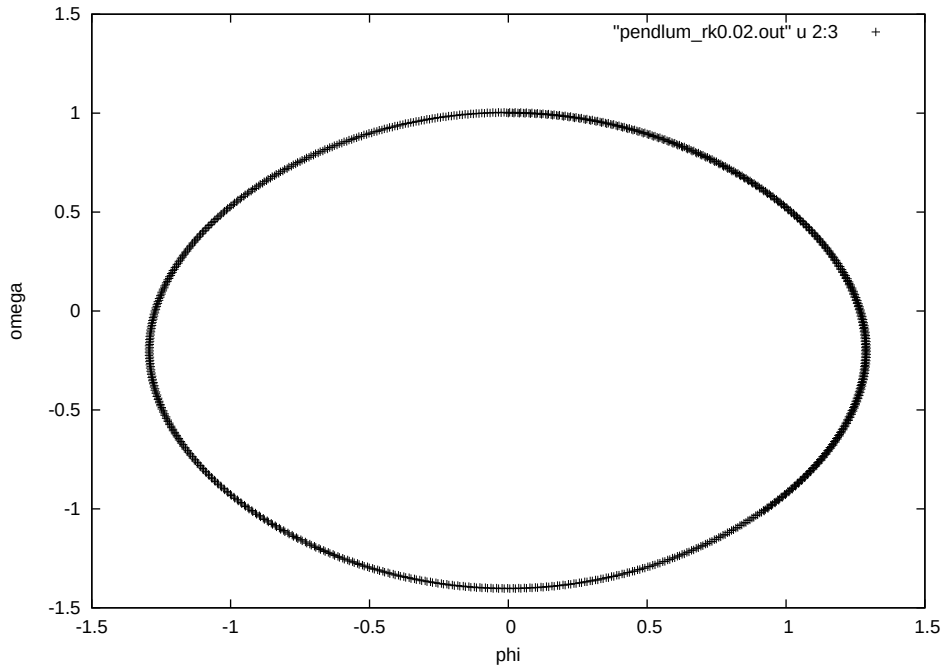
`yout[]`: Ergebnis (kann auch `y` sein)
`derivs`: Pointer auf Funktion die Ableitungen ausrechnet

Programm `pendlum_rk.c`:

```
/*** Integrates the pendlum using Runge-Kutta method ***/
/*** rk4 from numerical recipes                       ***/

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

#define select select_nr            /* otherwise: twice defined */
#include "nr.h"
#include "nrutil.h"

/** sets f[]=derivates at point x of variables y[] **/
/** y[1]=phi, y[2]=omega                           **/
void derivs(float x, float *y, float *f)
{
  f[1] = y[2];
  f[2] = -sin(y[1]);
}

int main(int argc, char **argv)
{
  float y[3];         /* phase space variables: unused,phi,omega */
  float f[3];                                    /* derivatives */
  float h;                                          /* stepsize */
  int n, n_max;                               /* step, maximum step */
  float t;                                              /* time */

  sscanf(argv[1], "%f", &h);        /* first argument = stepsize */
  sscanf(argv[2], "%d", &n_max); /*2nd argument = numb. of steps */
  y[1]=0; y[2]=1.0; t = 0;

  derivs(t, y, f);
  for(n=1; n<=n_max; n++)              /* iterate over all times */
  {
    printf("%f %f %f\n", t, y[1], y[2]);        /* print results */
    rk4(y, f, 2, t, h, y, derivs);           /* call Runge-Kutta */
    t += h;
  }
  return(0);
}
```

Compile: `cc -o pendlum_rk pendlum_rk.c nrutil.c rk4.c -lm`

Mit Stepsize 0.02 über 1000 Schritte ergibt im Phasenplot:



Verbesserungen: adaptive Schrittweitenanpassung, steife DGLs