Computerphysik I: Blatt 03

Aurel Müller-Schönau und Leon Oleschko

17. Juni 2022

a) Magnetpendel - Erste Schritte

Es soll ein Magnetpendel simuliert werden. Wir nehmen an, dass die Auslenkung klein ist, und beschränken uns auf die x-y-Ebene. Die auf die Stahlkugel wirkende magnetische Kraft soll von drei Magneten ausgehen und die form

$$\vec{F}_m = \sum_{i=1}^{N} S_i \cdot \frac{\vec{r} - \vec{r}_i}{|r - r_i|^3} \tag{1}$$

Die individuelle Magnetstärke S_i haben wir dabei selbst eingeführt, um das Programm für verschieden starke Magnete testen zu können. Sie wird aber im folgenden für alle Magnete 1 betragen. N ist die Anzahl der Magnete, $\vec{r}_i$ bezeichnet deren Position. Dabei haben wir uns im Programm bereits auf die x-y-Ebene beschränkt. Im Nenner kommt natürlich weiterhin auch die z-Komponente vor! Wie das praktisch aussieht, ist im Code im Anhang **??** zu sehen. Das Programm zeichnet den Pfad des Pendels für verschiedene Anfangspunkte, siehe dazu e). Im Programm werden die Koordinaten der Magneten in einem Array gespeichert, wobei die dritte Koordinate in wirklichkeit der Magnetstärke entspricht. Das Pendel befindet sich immer in der Höhe h über der Magnetebene.
Zusätzlich soll eine Schwerkraft

$$\vec{F}_f = -k\vec{r} \tag{2}$$

wirken, wobei wir $k = 1$ wählen. Weiterhin soll eine Reibungskraft

$$\vec{F_r} = -\gamma \dot{\vec{r}} \tag{3}$$

das Pendel letztendlich zum Stillstand bringen.

b)

Die Magnete wurden in einem regelmäßigen Dreieck im Abstand 1 um den Ursprung angeordnet. Die Koordinaten betragen also $(\cos(\alpha), \sin(\alpha))$ für $\alpha \in \{0, \frac{2\pi}{3}, \frac{4\pi}{3}\}$. Der Schwerpunkt liegt dann im Ursprung.

c)

Die Simulation wurde im *Leap-Frog-Verfahren* implementiert. Die Bewegungsgleichungen lauten dann

$$\vec{v}_{n+1} = \vec{v}_n + H \cdot \vec{F}_{gesamt}/m \tag{4}$$

$$\vec{r}_{n+1} = \vec{r}_n + H \cdot \vec{v}_{n+1} \tag{5}$$

Dabei wurde für die Beschleunigung $m = 1$ gewählt. Im Programm **??** taucht m deshalb gar nicht auf.

d)

Bevor die eigentliche Simulation beginnen konnte, wurde das Programm zunächst mit verschiedenen Anfangswerten getestet. So musste etwa ein Bug gefixt werden, bei dem das Pendel immer am Nullpunkt zum stehen kam. Es stellte sich heraus, dass das Pendel zu hoch über der Magnetebene war, und die magnetische Kraft somit klein gegenüber der Schwerkraft war. Schließlich konnte der Code auf zwei verschiedene Arten verwendet werden:

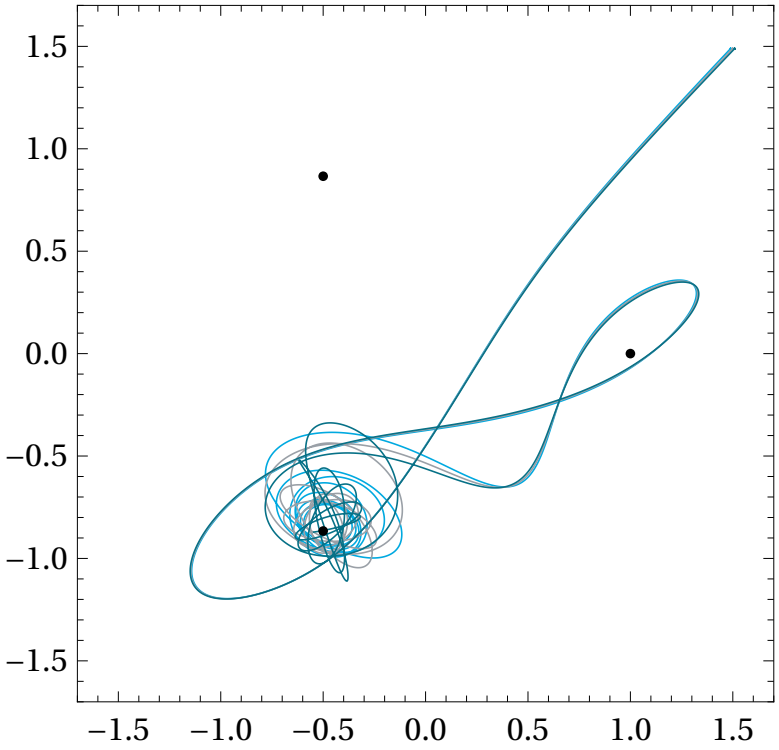

Abbildung 1: X-Y-Diagramm eines losgelassenen Pendels für drei geringfügig verschiedene Startpunkte.

1 e)

Im Programm **??** befindet sich das Pendel in einer Höhe von $h = 0.25$ Einheiten über der Magnetebene. Die Schwerkraftkonstante K beträgt 1, für die Dämpfung wurde $\gamma = 0.2$ gewählt. Dann wurden die Pfade dreier nahe beienanderliegender Startpunkte (keine Anfangsgeschwindigkeit) gezeichnet. Dies soll einen ersten Hinweis auf das chaotische Verhalten liefern. Die Anfangspunkte unterscheiden sich lediglich in der x-Koordinate um einen kleinen Wert $\pm\varepsilon$ mit $\varepsilon = 0.02$.
Das Ergebnis ist als Schaubild im Diagramm dargestellt. Gezeichnet wurden die x-y-Diagramme der drei verschiedenen Startpunkte. Es ist zu erkennen, dass sich trotz anfänglich fast identischer Pfade die Pendelbewegung mit fortschreitender Zeit deutlich anders entwickelt.

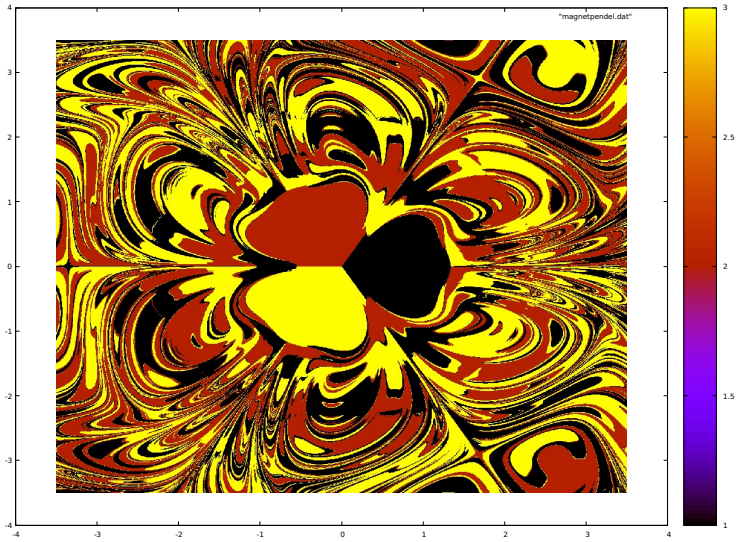

Abbildung 2: Schaubild zur Illustration des chaotischen Verhaltens des Magnetpendels. Jeder Pixel im Bild stellt einen anderen Startpunkt dar. Die Farbe des jeweiligen Pixels entspricht dem Magneten, der dem Endpunkt am nächsten liegt. Die Magnete selbst sind mit Kreuzen markiert, und liegen in den drei großen einfarbigen Bereichen.

f)

Die Simulation im Programm **??** ist identisch zu der im vorigen Teil, jedoch werden nun viele Punkte in der x-y-Ebene als Startpunkte gewählt. Für jeden Wert wird das Pendel ohne Anfangsgeschwindigkeit pendeln gelassen, bis es zum Stillstand gelangt, alternativ jedoch, falls 10000 Simulationsschritte erreicht sind. Dann wird der Startpunkt des Pendels farbkodiert entsprechend dem Magneten, der dem Endpunkt am nächsten liegt. Die Positionen der Magnete sind dabei deutlich als große einfarbige Bereiche zu erkennen, ein hier losgelassenes Pendel pendelt erst gar nicht los.
Im Schaubild **??** ist das Ergebnis zu bestaunen.

Appendix - Code

c)

Listing 1: Programm zur Simulation des Magnetpendels für drei jeweils um ε verschobene Startpunkte.

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

#define NMag 3 // Number of magnet
#define Nmax 20000 // Maximal number of simulation steps
#define H 0.001
#define GAMMA 0.2
#define K 1. // Schwerkraftst rke
#define h 0.25 // H he des Pendels

// parameters for the plot
const double start[2] = {1.5, 1.5};
const double epsilon = 1e-2;



double rMag[NMag][3]; // X- und Y-Koordinaten und Magnetst rke
const double Tmax = Nmax * H;
double r[2], v[2];
double kraft[2], diff[2], E; // temporary variables
FILE *file;

int main(){
    // calculate positions of the magnets and save to file "magnete.dat"
    file = fopen("./magnete.dat", "w+");
    // loop over all magnets
    for(int i = 0; i < NMag; i++){
        const double temp = 2*M_PI*i/NMag;

        rMag[i][0] = cos(temp);
        rMag[i][1] = sin(temp);
        rMag[i][2] = 1.;

        fprintf(file, "%g %g\n", rMag[i][0], rMag[i][1]); // plot "
    magnetpendel.dat" using 2:3 w l, "magnete.dat" lw 5
        //printf("x: %g \t\t y: %g \t\t St rke: %g \n", rmag[i][0], rmag
    [i][1], rmag[i][2]);
    }
```

```
37     fclose(file);
38
39   //
       ////////////////////////////////////////////////////////////////////////////////////////////////
40
41   // open output file
42   file = fopen("./path.dat", "w+");
43
44     // loop over x and y
45     for( double x = start[0]-epsilon; x <= start[0]+epsilon; x += epsilon
       ){
46         for( double y = start[1]-epsilon; y < start[1]+epsilon; y +=
       epsilon){
47
48       // set initial conditions
49             v[0] = 0;
50             v[1] = 0;
51             r[0] = x;
52             r[1] = y;
53
54       // loop over simulation Time
55             for(double t = 0; t < Tmax; t += H){
56
57         // reset the force
58                 kraft[0] = 0;
59                 kraft[1] = 0;
60         E = 0;
61
62         // calculate the Force from all Magnets
63         for(int i = 0; i < NMag; i++){
64                     diff[0] = rMag[i][0] - r[0];
65                     diff[1] = rMag[i][1] - r[1];
66
67                     kraft[0] += rMag[i][2] * diff[0]/pow(diff[0]*diff[0]
       + diff[1]*diff[1] + h*h, 1.5);
68                     kraft[1] += rMag[i][2] * diff[1]/pow(diff[0]*diff[0]
       + diff[1]*diff[1] + h*h, 1.5);
69
70           E -= 1* rMag[i][2]/pow(diff[0]*diff[0] + diff[1]*diff[1] + h*h,
        0.5);
71                 }
72         double Emag = E;
73
```

```
74          // add the force due to gravity
75                  kraft[0] += - K * r[0] - GAMMA * v[0];
76                  kraft[1] += - K * r[1] - GAMMA * v[1];
77
78          E += 1/2 * K * (r[0]*r[0] + r[1]*r[1]);
79                  //E += 1/2 * (v[0]*v[0] + v[1]*v[1]);
80
81                  // Leap Frog step
82                  v[0] += H * kraft[0];
83                  v[1] += H * kraft[1];
84                  r[0] += H * v[0];
85                  r[1] += H * v[1];
86
87          // print path
88          fprintf(file, "%g %g %g %g %g %g \n", t, r[0], r[1], E, Emag, E-
    Emag);
89        }
90          }// end of y scan
91          fprintf(file, "\n");
92
93      }// end of x scan
94
95      fclose(file);
96      return 0;
97  }
```

f)

Listing 2: Simulation des Magnetpendels zum erstellen eines Schaubildes. Die Farbe im Bild kodiert den nächstgelegenen Magneten am Ende der Simulationszeit für den jeweiligen Startpunkt.

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

#define NMag 3 // Number of magnet
#define Nmax 10000 // Maximal number of simulation steps
#define H 0.005
#define GAMMA 0.2
#define K 1. // Schwerkraftst rke
#define h 0.25 // H he des Pendels
#define kmin 0.5
#define vmin 0.5

// parameters for the plot
#define PW 3.5 // plot width
#define PP 600 // plot points



double rMag[NMag][3]; // X- und Y-Koordinaten und Magnetst rke
const double Tmax = Nmax * H;
double r[2], v[2];
double kraft[2], diff[2]; // temporary variables
FILE *file;

int main(){
    // calculate positions of the magnets and save to file "magnete.dat"
    file = fopen("./magnete.dat", "w+");
    // loop over all magnets
    for(int i = 0; i < NMag; i++){
        const double temp = 2*M_PI*i/NMag;

        rMag[i][0] = cos(temp);
        rMag[i][1] = sin(temp);
        rMag[i][2] = 1;

        fprintf(file, "%g %g\n", rMag[i][0], rMag[i][1]); // plot "
    magnetpendel.dat" using 2:3 w l, "magnete.dat" lw 5
```

```
38          //printf("x: %g \t\t y: %g \t\t St rke: %g \n", rmag[i][0], rmag
    [i][1], rmag[i][2]);
39      }
40      fclose(file);
41
42  //
    ////////////////////////////////////////////////////////////////////////////////////////////////////
43
44  // open output file
45  file = fopen("./magnetpendel.dat", "w+");
46
47      // loop over x and y
48      for( double x = -PW; x < PW; x += PW/PP){
49          for( double y = -PW; y < PW; y += PW/PP){
50
51        // set initial conditions
52              v[0] = 0;
53              v[1] = 0;
54              r[0] = x;
55              r[1] = y;
56
57        // loop over simulation Time
58              for(double t = 0; t < Tmax; t += H){
59
60          // reset the force
61                  kraft[0] = 0;
62                  kraft[1] = 0;
63
64          // calculate the Force from all Magnets
65          for(int i = 0; i < NMag; i++){
66                      diff[0] = rMag[i][0] - r[0];
67                      diff[1] = rMag[i][1] - r[1];
68
69                      kraft[0] += rMag[i][2] * diff[0]/pow(diff[0]*diff[0]
    + diff[1]*diff[1] + h*h, 1.5);
70                      kraft[1] += rMag[i][2] * diff[1]/pow(diff[0]*diff[0]
    + diff[1]*diff[1] + h*h, 1.5);
71
72                  }
73
74          // add the force due to gravity
75                  kraft[0] += - K * r[0] - GAMMA * v[0];
76                  kraft[1] += - K * r[1] - GAMMA * v[1];
```

```
77
78                  // Leap Frog step
79                  v[0] += H * kraft[0];
80                  v[1] += H * kraft[1];
81                  r[0] += H * v[0];
82                  r[1] += H * v[1];
83
84                  // check for stagnation
85                  if(fabs(v[0]) < vmin)
86                      if(fabs(v[1]) < vmin)
87                          if(fabs(kraft[0]) < kmin)
88                              if(fabs(kraft[1]) < kmin)
89                                  break; // Simulation abbrechen, wenn sich
     nichts mehr   ndert
90
91
92
93          // print path for debugging
94                  //printf("%g %g %g\n", t, r[0], r[1]);
95
96              }
97
98        // print for debugging
99              //printf("v0: %g    v1: %g    k0: %g    k1: %g\n", fabs(v[0])
    , fabs(v[1]), fabs(kraft[0]), fabs(kraft[1]));
100
101       // find the closest magnet
102             double temp = 1e5; // closest distance to magnet
103             int maxi; // index of strongest magnet
104             for(int i = 0; i < NMag; i++){
105       // calculate distance vector
106                 diff[0] = rMag[i][0] - r[0];
107                 diff[1] = rMag[i][1] - r[1];
108
109       // compare distance squares
110                 if(diff[0]*diff[0] + diff[1]*diff[1] < temp){
111                     temp = diff[0]*diff[0] + diff[1]*diff[1];
112                     maxi = i;
113                 }
114             }
115
116       // print to file
117             fprintf(file, "%g %g %d %g\n", x, y, maxi+1, sqrt(temp) );
118
```

```
119         }// end of y scan
120         fprintf(file, "\n");
121
122     }// end of x scan
123
124     fclose(file);
125     return 0;
126 }
```