

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

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

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 1 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

$$\vec{F}_f = -k\vec{r} \quad (2)$$

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

$$\vec{F}_r = -\gamma \dot{\vec{r}} \quad (3)$$

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

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

$$(5)$$

$$\vec{r}_{n+1} = \vec{r}_n + H \cdot \vec{v}_{n+1} \quad (6)$$

Dabei wurde für die Beschleunigung $m = 1$ gewählt. Im Programm 1 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:

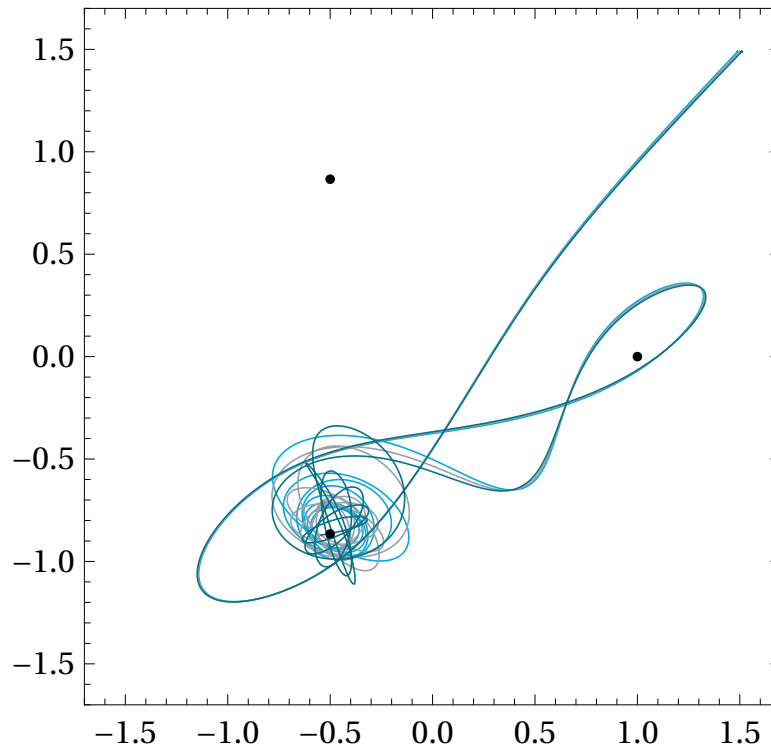


Abbildung 1: X-Y-Diagramm eines losgelassenen Pendels für drei geringfügig verschiedene Startpunkte.

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 beieinanderliegender 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.

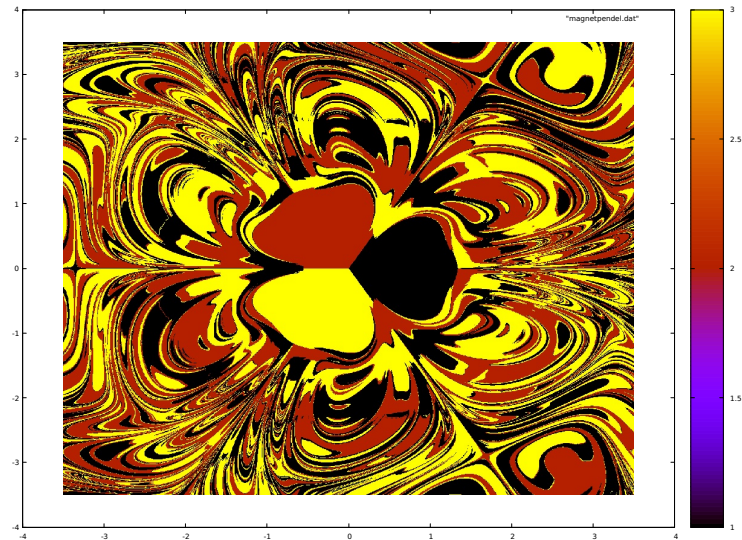


Abbildung 2: Schaubild zur Illustration des chaotischen Verhaltens des Magnetpendels. Jeder Pixel im Bild stellt einen anderen Startpunkt dar. Die Farbe des jeweiligen Pixels entspricht dem Magneten, der dem Endpunkt am nächsten liegt. Die Magnete selbst sind mit Kreuzen markiert, und liegen in den drei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

Appendix - Code

c)

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

```

1  #include <stdio.h>
2  #include <math.h>
3
4  #define NMag 3 // Number of magnet
5  #define Nmax 20000 // Maximal number of simulation steps
6  #define H 0.001
7  #define GAMMA 0.2
8  #define K 1. // Schwerkraftst rke
9  #define h 0.25 // H he des Pendels
10
11 // parameters for the plot
12 const double start[2] = {1.5, 1.5};
13 const double epsilon = 1e-2;
14
15
16
17 double rMag[NMag][3]; // X- und Y-Koordinaten und Magnetst rke
18 const double Tmax = Nmax * H;
19 double r[2], v[2];
20 double kraft[2], diff[2], E; // temporary variables
21 FILE *file;
22
23 int main(){
24     // calculate positions of the magnets and save to file "magnete.dat"
25     file = fopen("./magnete.dat", "w+");
26     // loop over all magnets
27     for(int i = 0; i < NMag; i++){
28         const double temp = 2*M_PI*i/NMag;
29
30         rMag[i][0] = cos(temp);
31         rMag[i][1] = sin(temp);
32         rMag[i][2] = 1.;
33
34         fprintf(file, "%g %g\n", rMag[i][0], rMag[i][1]); // plot "
magnetpendel.dat" using 2:3 w 1, "magnete.dat" lw 5
35         //printf("x: %g \t\t y: %g \t\t St rke: %g \n", rmag[i][0], rmag
[i][1], rmag[i][2]);
36     }

```

```

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     ){
47         for( double y = start[1]-epsilon; y < start[1]+epsilon; y +=
48         epsilon){
49
50             // set initial conditions
51             v[0] = 0;
52             v[1] = 0;
53             r[0] = x;
54             r[1] = y;
55
56             // loop over simulation Time
57             for(double t = 0; t < Tmax; t += H){
58
59                 // reset the force
60                 kraft[0] = 0;
61                 kraft[1] = 0;
62
63                 E = 0;
64
65                 // calculate the Force from all Magnets
66                 for(int i = 0; i < NMag; i++){
67                     diff[0] = rMag[i][0] - r[0];
68                     diff[1] = rMag[i][1] - r[1];
69
70                     kraft[0] += rMag[i][2] * diff[0]/pow(diff[0]*diff[0]
+ diff[1]*diff[1] + h*h, 1.5);
71                     kraft[1] += rMag[i][2] * diff[1]/pow(diff[0]*diff[0]
+ diff[1]*diff[1] + h*h, 1.5);
72
73                     E -= 1* rMag[i][2]/pow(diff[0]*diff[0] + diff[1]*diff[1] + h*h,
0.5);
74                 }
75                 double Emag = E;

```

```
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-
89         Emag);
90     }
91     } // end of y scan
92     fprintf(file, "\n");
93 } // end of x scan
94
95 fclose(file);
96 return 0;
97 }
```

f)