Physik auf dem Computer I & II

PD Dr. S. Luding, Prof. Dr. H. J. Herrmann,
Dr. S. Schwarzer, Dipl. Phys. Matthias Müller,
Dr. H.-G. Matuttis, T. Karle und S. Manmana
Uni–Stuttgart

29. Oktober 1998

Inhaltsverzeichnis

Kapitel 1

Einführung

1.1 Was heißt "Physik auf dem Computer"

In dieser Vorlesung soll gezeigt werden, wie man physikalische Fragestellungen mit dem Computer angehen kann. Dies kann auf vielfältige Weise geschehen:

- *Schreibe die Diplomarbeit mit dem Computer*
  Alleine die Textverarbeitung ist nicht mehr aus dem wissenschaftlichen Leben wegzudenken – handgeschriebene Texte sind heutzutage nicht mehr akzeptabel (wie in Prä-Computer Zeiten).

- *Benütze den Computer als besseren Taschenrechner*
  Man macht Physik auf althergebrachte Weise und schaltet den Computer nur im Notfall ein. Je länger man mit dem Computer arbeitet, umso mehr Anwendungsmöglichkeiten wird man finden: z.B. **Grafik**, **Internet-Zugang** oder schnelle Kommunikation via **e-mail**.

- *Steuere Experimente mit dem Computer*
  Der Computer arbeitet auch nachts und ist viel ausdauernder als der fleissigste Student oder Wissenschaftler. Die riesige Datenmenge, die bei Großexperimenten anfällt ist von Menschen überhaupt nicht mehr zu bewältigen.

- ***Werte experimentelle Daten aus***
  Hat man einmal einen Datensatz (z.B. Flugbahn eines UFOs), so kann man unzählige Auswertungs-Operationen durchführen: Aus der Flugbahn kann man extrapolieren, wo das Flugobjekt herkam oder hinfliegt; Interpolation oder Glättung macht aus einer Punktewolke ein glatte Funktion, von der man dann Ableitungen berechnen kann, um die Beschleunigung zu bestimmen.

- ***Löse physikalische Probleme mit dem Computer***
  Nur die wenigsten Fragestellungen in der Physik sind analytisch "mit Bleistift und Papier" lösbar. Es werden jedoch viele Prozesse in der Natur durch komplizierte, nicht-lösbare Differentialgleichungen beschrieben. Der Computer hilft z.B. solche Gleichungen numerisch zu lösen (reales, nichtlineares Pendel) oder hilft bei der analytischen Lösung mit sog. "symbolischen" Programmiersprachen (z.B. MAPLE).

- ***Benutze den Computer für numerische "Experimente"***
  Zum Verständnis vieler physikalischer Probleme, kann man – im Gegensatz zum realen Experiment – das System stufenweise vereinfachen, bis nur noch die essentielle Physik übrigbleibt. Verhält sich das vereinfachte System ebenso wie das reale System, ist man dem Verständniss einen Schritt näher gekommen, da man unwichtige Effekte eliminiert hat. Man kann mit dem Computer sogar "unmögliche" Experimente durchführen. Wissenschaftler untersuchen z.B. Quanten-Phänomene, Sonnen oder sogar schwarze Löcher u.v.m., also Systeme, die experimentell nicht zugänglich sind.

In dieser Vorlesung werden wir uns vor allem auf die letzten drei Punkte konzentrieren. Zuvor jedoch noch einige grundlegende Bemerkungen und Erklärungen.

1.1.1 Vom Menschen zum Computer und zurück

Am Anfang steht eine Idee oder ein zu lösendes Problem. Der Forscher muß dieses nun in "Worte" fassen, die der Computer verstehen muß, um bei der Lösung helfen zu können. Eine geeignete "Sprache" nennt man **Programmiersprache** und die Befehle, die der Computer ausführen soll, **Programm**. Die Art und Weise, wie man das zu lösende Problem formuliert, nennt man **Algorithmus**. Dieser hängt a-priori nicht von der Programmiersprache ab, auch wenn bestimmte Sprachen für manche Algorithmen besser geeignet sein können als andere. Es gibt allerdings Abstufungen, was die Maschinennähe, bzw. Benutzerorientierung angeht. Besonders maschinennah sind Assemblersprachen, ein flexibler und allgemeinerer Kompromiss sind höhere Programmiersprachen (C/C++, PASCAL, FORTRAN) und besonders benutzernah (weil problemorientiert) sind Sprachen wie LISP und Interpreter wie MATHLAB oder MAPLE. Mit letzteren kann man bestimmte Aufgaben ganz einfach lösen, andere dagegen nur schwer.

Hat man dem Computer einmal beigebracht was man von ihm erwartet, so wird er das entweder tun oder auch nicht. Er wird sich häufig beschweren, daß er die Sprache (**Syntax**) nicht oder falsch versteht. Neben solchen formalen Fehlern gibt es natürlich noch die logischen Fehler, die schon bei der Formulierung des Problems auftreten. Diese müssen natürlich vom Programmierer selbst aufgespürt werden; sie können häufig dadurch vermieden werden, daß man vor dem Programmieren sorgfältig nachdenkt. Mit den Beschwerden (und anderen Hilfsmitteln) kann man den Fehler im Programm lokalisieren und ausbügeln (**debugging**). Tut der Computer schließlich was man von ihm will, so gibt er z.B. Zahlen auf dem Bildschirm aus, die man dann als Grafik darstellen kann. Das Gesamtschema ist in Abb. 1.1 dargestellt.

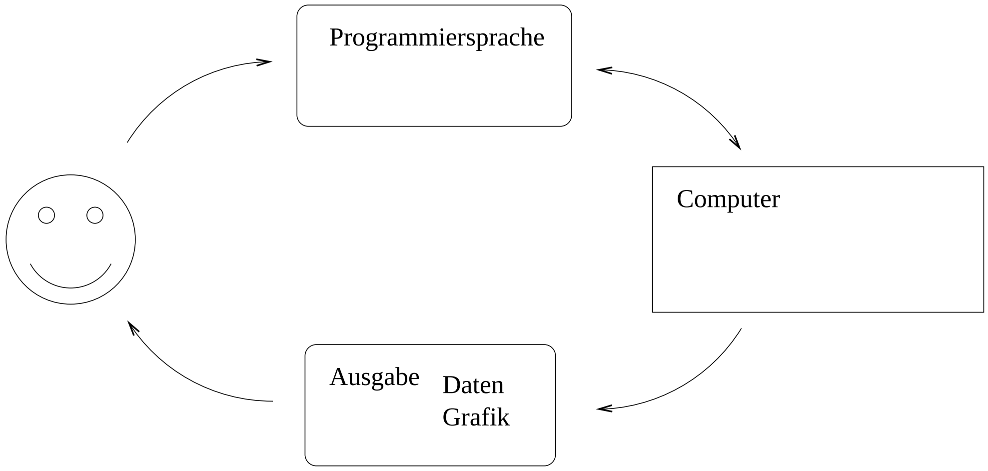

Abbildung 1.1: Schematische Darstellung der Interaktion zwischen Mensch und Computer

1.1.2 Das erste Beispiel

Wie oben erwähnt steht am Anfang das physikalische Problem. Der Einfachheit halber wählen wir einen harmonischen Oszillator, also eine Masse die an einer Feder schwingt. Ziel ist es – mit Hilfe des Computers – die Bewegung der Masse auf dem Bildschirm darzustellen.

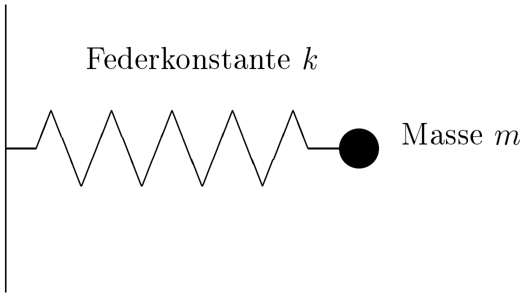

Abbildung 1.2: Harmonischer Oszillator

Zuerst muß man das physikalische Problem genau spezifizieren. Wir gehen davon aus, daß die Masse des schwingenden Körpers viel schwerer ist als die Feder (die Federmasse kann damit vernachlässigt werden). Weiterhin sollen keine externen Kräfte, wie Gravitation und Reibung wirken. Dann folgt aus Newtons zweitem Gesetz und der Hookschen Beziehung für die Federkraft:

$$m\ddot{x} = -kx \ . \tag{1.1}$$

Dabei ist k die Federkonstante, m die Masse, x die Auslenkung der Feder und $\ddot{x}$ die zweite Zeitableitung derselben. Die Lösung dieser Gleichung ist analytisch möglich, und enthält zwei frei verfügbare Konstanten (hier die Phase δ und die Amplitude A), da es sich um eine Differentialgleichung (DGL) zweiter Ordnung handelt,

$$x(t) = A\sin(\omega t + \delta) \ . \tag{1.2}$$

Die Schwingungsfrequenz $\omega^2 = \frac{k}{m}$ ist durch die physikalischen Parameter festgelegt. Setzt man die Anfangsbedingungen $x(0) = x_0$ und $\dot{x}(0) = v_0$ in die Gleichung mit $t = 0$ ein, ergeben sich die Amplitude

$$A = \frac{x_0}{\sin\delta} \ , \tag{1.3}$$

mit der Phase

$$\tan\delta = \frac{x_0\omega}{v_0} \ . \tag{1.4}$$

Alternativ kann man auch Anfangsort und -geschwindigkeit anstelle von Amplitude und Phase verwenden.

Bis jetzt wurde die Hilfe des Computers noch nicht benötigt. Außerdem wäre es jederzeit möglich die Lösung mit Hilfe eines Taschenrechners auf Millimeterpapier zu übertragen. Trotzdem wollen wir dieses Beispiel weiterführen um schrittweise die Vorgehensweise zu verstehen.

Um eine Funktion $x(t)$, Gleichung (1.2), in einem Programm verwenden zu können muß sie diskretisiert werden, d.h. man berechnet sie zu verschiedenen Zeitpunkten t_i, genauso, wie man es machen würde, um sie auf Millimeterpapier zu zeichnen. Der Algorithmus ist in diesem Fall denkbar trivial, er lautet nämlich: *Berechne die Funktion $x(t)$ an den Stützstellen t_0, t_1, t_2, ..., t_n.* Ziel ist es nun, ein Computerprogramm zu schreiben, das genau dies tut.

```
// Binde noetige Bibliotheken ein
#include<iostream>
#include<fstream>

// Definiere das Hauptprogramm
int main()
{
```

```
// Definiere das Feld x(t)
   double x[1000], t;   // Ausgabevariablen

// Anfangsbedingungen und Zustaende
   double A, delta;     // A=Amplitude, delta=Phasenwinkel
   double mass, kfeder; // mass=Masse, kfeder=Federkonstante
   double t_max, dt;    // t_max=Gesamtzeit, dt=Zeitintervall

// Parameter abfragen und einlesen
   cout << "Amplitude     ";  cin >> A;
   cout << "Phasenwinkel  ";  cin >> delta;
   cout << "Masse         ";  cin >> mass;
   cout << "Federkonstante";  cin >> kfeder;
   cout << "Gesamtzeit    ";  cin >> t_max;
   cout << "Zeitintervall ";  cin >> dt;
   double omega=sqrt(kfeder/mass);
// Nicht ANSI-Konform - besser waere:
// std::cout und std::cin anstelle von cout und cin

// Schleife von t=0 bis t=t_max
   t=0.0;
   for( int i=0; i<=t_max/dt; i++ ) // Schleife
   {
        x[i]=A*sin(omega*t+delta);  // Berechne die Funktion
        t=t+dt;                     // Gehe zur naechsten Zeit
   }

// Ausgabe
   ofstream outfile("harmon.dat");       // Oeffne Datei
   t=0.0;
   for( int i=0; i<=t_max/dt; i++ ) // Schleife
   {
      outfile << t << " " << x[i] << "\n";  // Ausgabezeile
      t=t+dt;                               // Gehe zur naechsten Zeit
   }
// Ende des Hauptprogramms
   return 0;
}
```

Die Bedeutung der vielen Symbole wird in den folgenden Kapiteln ausführlicher erklärt werden. Zum Schreiben des Programms benützt man einen Editor (Textverarbeitung) wie z.B. `xedit`, `emacs` oder `xemacs`. Das geschriebene Programm kann man dann als Datei `harmon.cc` abspeichern, wobei die Endung `.cc` anzeigt daß es sich um ein C++ Programm handelt. Der Computer kann mit dem Programm alleine noch nichts anfangen, man muß es erst in Maschinensprache übersetzen, d.h. man *compiliert* es. Dies geschieht mit dem

Kommando `g++ harmon.cc -o harmon`, das die *ausführbare Datei* `harmon` erzeugt. Diese wird durch das Kommando `harmon` aktiviert und fragt dann nacheinander die Amplitude, den Phasenwinkel, die Masse, die Federkonstante, die Gesamtzeit und das Zeitintervall der Stützstellen ab. Hat man diese eingegeben, so erzeugt das Programm die Datei `harmon.dat`, mit z.B. folgendem Inhalt (bei Verwendung von $A = 1$, $\delta = 0$, $m = 1$, $k = 10$, Ausgabedauer $t_{max} = 1.5$ und Zeitschritt $dt = 0.1$):

```
0   0
0.1  0.310984
0.2  0.591127
0.3  0.812649
0.4  0.953581
0.5  0.999947
0.6  0.947148
0.7  0.800422
0.8  0.574318
0.9  0.291259
1   -0.0206835
1.1 -0.330575
1.2 -0.607684
1.3 -0.824528
1.4 -0.959605
1.5 -0.999519
```

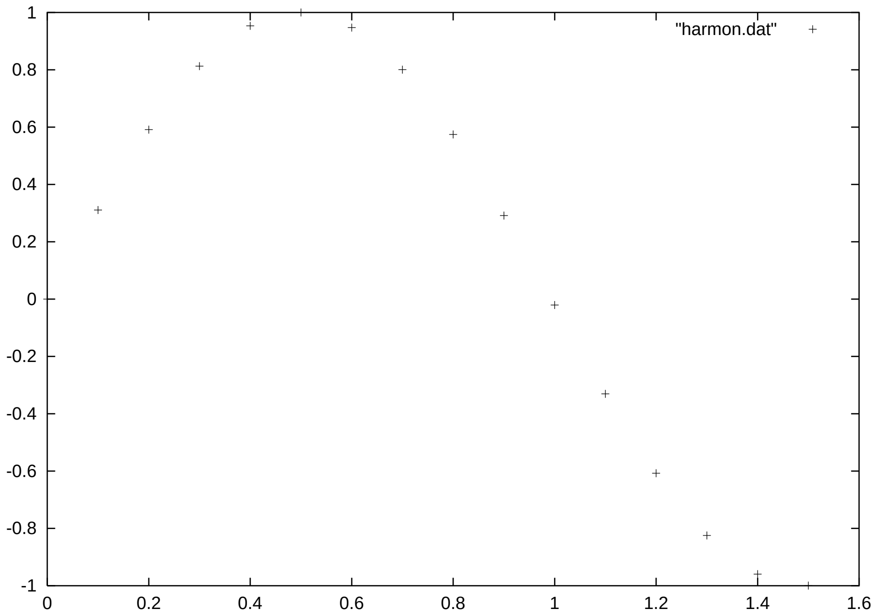

Abbildung 1.3: Grafische Darstellung der zweispaltigen Zahlenreihe auf dem Bildschirm. Die erste Spalte definiert die x-, die zweite die y-Koordinate der Punkte.

Diese Zahlenkolonne (2 Spalten) kann man nun mit dem Programm `gnuplot` auf den Bildschirm zeichnen. Nach Eingabe des Kommandos `gnuplot` erscheint eine Meldung des Programms und der sog. *Prompt*: '`>`'. Dann gibt man den Befehl
`plot "harmon.dat"` ein und wird die Grafik, wie in Abb. 1.3, auf dem Bildschirm erscheinen sehen. Dieses Beispiel kann zusammen mit anderen in der ersten Übung ausprobiert werden.

1.2 Programmier-Philosophie

Man kann verschiedene Programmiersprachen für ein Problem verwenden. Je nach Entwicklungsstufe und Spezialisierungsgrad sind die Sprachen mehr oder weniger benutzerfreundlich. Manche Sprachen eignen sich besonders für spezielle Aufgaben. Es ist im Prinzip egal, welche Sprache man verwendet, jedoch erspart die richtige Wahl unter Umständen Zeit und Mühe.

Eine Programmiersprache wie C++ hat die denkbar größte Komplexität und Flexibilität, und findet heutzutage die breiteste Anwendung bei Programmierern. Bei Stellenangeboten werden sehr häufig Kenntnisse in C++ gefordert. Außerdem sind viele immer wieder vorkommende Aufgaben mit existierenden *Tools* leicht zu lösen. Aus diesen Gründen wird im folgenden die Syntax von C++ verwendet.

Unabhängig von der Syntax (dem Vokabular) der Programmiersprache C++ kann man jedoch Aussagen zum Programmierstil machen. Wenn das Ziel ist, ein Problem schnellstmöglich zu lösen, wird man zwar kurzfristige Erfolge erzielen, langfristig aber feststellen, daß viele Nachteile gegen eine "quick and dirty" Methode sprechen. Der Programmcode ist schon nach wenigen Tagen unverständlich geworden, weil man nicht von der Möglichkeit gebraucht macht, **Kommentare** einzubauen. Deshalb ist das Programm nahezu unverständlich für andere und ist dadurch nicht wiederverwendbar. Man kann die schnelle Methode also als "write-only" bezeichnen.

Ziel soll es jedoch sein "write-and-read" Programme zu schreiben, die man auch im nächsten Monat noch verstehen und benutzen kann (wenn man es für eine neue Übungsaufgabe benötigt). Dazu gehört eine konsistente Schreibweise, die man sich leider erst durch Erfahrung angewöhnt. Deshalb bietet es sich an, sich ein System auszusuchen und dieses beizubehalten. Ein optisch übersichtliches Programm hilft bei Änderungen und beim Verstehen ebenso wie häufige kurze Kommentare. Man sollte auch Dinge kommentieren, die im Moment klar erscheinen, denn schon bald kann das nicht mehr der Fall sein.

1.3 Geschichte der Programmierung mit C++

C wurde in den 70er Jahren von K. Thompson, B. Kernighan und D. Ritchie zusammen mit dem Betriebssystem UNIX eingeführt. Es war als typisierte Sprache zur strukturierten Programmierung konzipiert. Der ANSI Standard wurde 1988 als ANSI-C festgelegt.

C++ ist als Erweiterung von C zu sehen, die zwischen 1983 und 1985 von B. Stroustrup propagiert und eingeführt wurde. Neben dem Ziel, eine objektorientierte Sprache zu erschaffen, war auch eine Abwärtskompatibilität zu C erwünscht. Deshalb ist C als Teilmenge von C++ anzusehen. Der ANSI Standard für C++ wurde erst im Juni 1998 festgelegt. Dieses Skript hält sich noch nicht ganz daran, da auch die meissten benutzten Computer noch nicht ANSI-C++ "sprechen". Im Rahmen dieser Vorlesung werden wir C++ als ein "besseres" C verwenden und einige dubiose Eigenschaften von C vermeiden.

Gründe für die Verwendung von C++ sind Systemunabhängigkeit, Erweiterbarkeit, Wiederverwendbarkeit und Robustheit. Die Bedeutung dieser Begriffe wird aber erst klar, wenn man länger mit der Sprache arbeitet.

Ziel dieser Vorlesung ist es nicht, perfekt C++ zu programmieren! Dazu gibt es Spezialkurse und Bücher, auf die unten verwiesen wird. Stattdessen soll erlernt und vertieft werden, wie man physikalische Fragestellungen erkennt, in ein Programm formuliert und schließlich mit Hilfe des Computers beantwortet.

1.4 Anwendungsbeispiele

Im Laufe der Vorlesung wird nach einer
• Einführung in die Programmierung (Syntax C++),
• die numerische Differentiation, Integration und Interpolation,
• die Lösung von Differentialgleichungen (Pendel, Planeten),
• der dichtlineare Oszillator, Chaos, dann
• Zufallszahlen, Monte-Carlo Methoden, Gittermodelle und
• Fouriertransformation (FFT)
behandelt.

1.5 Literatur

Als weiterführende Literatur empfehlen wir die *Manuals des Rechenzentrums* für C, C++ und UNIX, die sich durch ein hervorragendes Preis-Leistungs Verhältnis auszeichnen. Ein Nachschlagewerk für numerische Methoden ist das Buch *Numerical Recipes* von William H. Press, Saul A. Teukolsky, William T. Vetterling und Brian P. Flannery, Cambridge University Press. Daneben gibt es mehrere – häufig englischsprachige – Werke, wie z.B.:
◦ S. E. Koonin, Physik auf dem Computer, Oldenburg Verlag
◦ H. Gould, I. Tobochnik, Introduction to Computer Simulation Methods, Addison Wesley
◦ D. W. Heermann, Computer Simulation Methods, Springer
◦ W. Kinzel, G. Reents, Physics by Computer, Springer