Informatik für Physiker SoSe 18
Dreikörperproblem

Maximilian Jaffke (1721498)
Timo Göhring (1733514)

20. Juli 2018

Inhaltsverzeichnis

1 Motivation

Seit Jahrhunderten bekannt, bis heute keine vollständige analytische Lösung.

1.1 Worum es sich handelt

Das Dreikörperproblem beschreibt das Problem der Vorhersage eines Systemes, von dem man zu einem Zeitpunkt den Ort und die Impulse, also die Masse und Geschwindigkeit, der drei Körper kennt.

1.2 Physikalischer Hintergrund

Nach dem Newtonschen Gravitationsgesetz bewirken die Massen zweier Massepunkte eine Gravitationskraft aufeinander. Das Gravitationspotenzial eines Körpers nimmt dabei mit $\frac{1}{r}$ ab, wobei r der Abstand zum Massepunkt ist. Dies bewirkt, dass sich auch mehrere Massepunkte gegenseitig beeinflussen, theoretisch wird das Potenzial erst im Unendlichen null.
Die Kraft auf einen der Massepunkte, hier m_1 lässt sich nach dem Newtonschen Gravitationsgesetz wie folgt beschrieben, wobei in dem System n Körper mit Ortsvektor $\vec{r_n}$ sind und G die Gravitationskonstante.

$$\vec{F_1} = Gm_1 \sum_{i=2}^{n} m_i \frac{\vec{r_i} - \vec{r_1}}{|\vec{r_i} - \vec{r_1}|^3} = m_1 \vec{a_1} = m_1 \ddot{\vec{r_1}} \tag{1}$$

Für den n-ten Körper gilt zum Zeitpunkt t

$$\dot{\vec{v_n}}(t) = \vec{a_n}(\vec{r_n}(t)) \tag{2}$$

$$\dot{\vec{r_n}}(t) = \vec{v_n}(t)$$

1.3 Lösungsideen

Für zwei Massepunkte kann man aus der Energieerhaltung eine Lösung für die Bahngleichung gewinnen, was hier nicht weiter ausgeführt wird. Für mehr als zwei Massepunkte gibt es normalerweise keine analytische Lösung. Durch Vereinfachung des Problems kann man für einige Startparameter Lösungen erhalten.
Man kann zum Beispiel eine Masse in einem Dreikörpersystem als vernachlässigbar klein annehmen. Beispielsweise können Sonnen-Erde-Satelliten-Probleme gelöst werden, da der Satellit keinen nennenswerten Einfluss auf die anderen Objekte im System hat.
Für Dreikörpersysteme und allgemeiner auch n-Körpersysteme lässt sich aber immer die Kraft im System zu einem Zeitpunkt berechnen. Da die Kraft der Impulsänderung entspricht und die Massen im Idealfall konstant bleiben, ändert sich die Geschwindigkeit eines Körpers. Die Geschwindigkeit ist wiederum die Änderung des Ortes. Man kann das System also punktweise lösen. Da Computer eine immer höhere Rechenleistung aufweisen, kann man das Problem auf neue Weise untersuchen.

1.4 Aufgabenstellung

Es ist im Folgenden die Aufgabe eine Programm zu entwickeln, welches die Bahnkurve von sich gegenseitig beeinflussenden Körpern berechnen kann. Durch eine Ausgabe von Daten soll eine Visualisierung ermöglicht werden. Dazu werden Vereinfachungen getätigt. Die Planeten befinden sich im Weltraum, weshalb das System als reibungslos angenommen wird. Auch Effekte wie der Strahlungsdruck der Photonen eines sich gegebenenfalls im System befindlichen Sternes wird vernachlässigt.
Da primär Dreikörpersysteme untersucht werden sollen, wird die Betrachtung auf zwei Dimensionen vereinfacht, weil die drei Ortsvektoren entweder auf einer Geraden liegen oder maximal eine zweidimensionale Ebene aufspannen können. Dies vereinfacht zugleich eine Visualisierung des Ergebnisses.

2 Methodik

Numerisch gibt es meist mehrere Ansätze, zwei werden für das Dreikörperproblem betrachtet.

2.1 Problem

Wie bereits im Punkt 1.3 beschrieben, lässt sich das System für beliebige Fälle fast immer nur punktweise lösen. Unter den Vereinfachungsannahmen ist nur die Beschleunigung, die auf die Körper wirkt, zu gegebener Position bestimmbar.

2.2 Explizites Euler-Verfahren

Für eine gewöhnlich Differentialgleichung erster Ordnung

$$y' = f(t, y) \tag{3}$$

kann man t in möglichst kleine Schritte Δt aufteilen und dann y punktweise betrachten mit

$$t_{n+1} = t_n + \Delta t \tag{4}$$

$$y_{n+1} = y_n + \Delta t f'(t_n, y_n)$$

Dabei kann jede Raumdimension in dem Problem einzeln betrachtet werden, welche zum Schluss überlagert dargestellt werden können.
Es wird also mit Hilfe des Gravitationsgesetzes die Beschleunigung bestimmt und dann mit der Methode die Geschwindigkeit, welche wiederum genutzt werden kann, um mit nochmaliger Anwendung der Methode einen neuen Ort zu liefern.

2.3 Verlet-Algorithmus

Die Idee des Verlet-Algorithmus ist es, die Position $x(t)$ einmal vorwärts in der Zeit $(t+\Delta t)$ und einmal rückwärts $(t - \Delta t)$ nach Taylor zu entwickeln. Dies ergibt

$$x(t + \Delta t) = x(t) + v(t)\Delta t + \frac{1}{2}a(t)\Delta t^2 + \frac{1}{6}b(t)\Delta t^3 + \mathcal{O}(\Delta t^4) \tag{5}$$

$$x(t - \Delta t) = x(t) - v(t)\Delta t + \frac{1}{2}a(t)\Delta t^2 - \frac{1}{6}b(t)\Delta t^3 + \mathcal{O}(\Delta t^4).$$

Eine Addition und Umformung beider Gleichung ergibt

$$x(t + \Delta t) = 2x(t) - x(t - \Delta t) + a(t)\Delta t^2 + \mathcal{O}(\Delta t^4). \tag{6}$$

Der Restterm $\mathcal{O}(\Delta t^4)$ wird bei der Berechnung dann vernachlässigt. Für eine Iteration mit Δt bedeutet dies dann

$$x_{n+1} = 2x_n - x_{n-1} + a_n \Delta t^2 \tag{7}$$

Für dieses Verfahren wird nicht die Geschwindigkeitsberechnung benötigt, was weniger Rechenschritte zu Folge hat.
Auch hier können die Raumdimensionen einzeln betrachtet werden und anschließend überlagert dargestellt werden.

3 Dokumentation

Hier werden die wichtigsten Bausteine des Programmes beschrieben und erläutert.

3.1 IDE und Compiler

Das Programm wurde in CLion erstellt und mit Hilfe von mingw-w64 compiliert.

3.2 Klasse Gravk

Die Klasse der Gravitationskörper ist dazu da, um jedem Objekt im System seine Eigenschaften, also Ort und Geschwindigkeit zu geben. Für diese Größen sind verschiedene Methoden definiert. Vier zweielementige Arrays des Typen double speichern die x-Koordinate, y-Koordinate, Geschwindigkeit in x-Richtung und in y-Richtung. Ein als mass benannter double speichert die Masse eines Körpers.

3.2.1 void Gravk::setAll(double a, double b, double c, double d, double h)

Die Funktion setzt die übergebenen Werte in den Arrays als Startwerte fest und setzt auch die Masse eines Objektes.

3.2.2 double Gravk::get...(unsigned t)

Diese Funktionen geben den Wert des t-ten Arrayelementes zurück, notwendig um die x- und y-Komponente der Ortsvektoren zu erhalten. getMass gibt hingegen ohne einen Übergabeparameter nur die Masse des Objektes zurück.

3.2.3 double Gravk::distance(Gravk a)

Berechnet wird die euklidische Norm zwischen dem funktionsaufrufenden Körper und des übergebenen Körpers mit Hilfe der get-Funktionen. Ausgerechnet wird ein Skalar, welches als double zurückgegeben wird.

3.2.4 double Gravk::Forcex(Gravk **body, int bcount)

Hier wird die Gesamtkraft auf den übergebenen Körper berechnet, indem die einzelnen Kräfte zwischen zwei Körper mit dem Newtonschen Gravitationsgesetz berechnet werden und addiert werden. Diese Funktion gibt es jeweils für y- und y-Komponente.
Voraussetzung zur Berechnung des Wertes ist es, dass die Entfernung zwischen den beiden Körpern größer als $1000m$ ist, damit ausgeschlossen werden kann, dass nicht die Kraft eines Körpers zu sich selbst bestimmt wird, da dieser Betrag nichts zur Gesamtkraft beiträgt, aber durch Null dividiert werden müsste.

3.2.5 void Gravk::setVel(Gravk **body, int bcount, unsigned dt)

Diese Funktion greift auf die Force-Funktionen zurück, um mit dem Euler-Verfahren die neue Geschwindigkeit mit mit Intervalllänge dt zu bestimmen und überschreibt den vorherigen Wert.

3.2.6 void newPos(unsigned dt)

Für den aufrufenden Körper wird die neue Position ebenfalls durch das Euler-Verfahren berechnet und anschließend wird der alte Wert mit dem Neuen überschrieben.

3.2.7 void firstPos(Gravk** body, int bcount, unsigned dt)

Da der Verlet-Algorithmus die letzten beiden Positionsiterationen benötigt, muss zu den Startparametern noch einer erster Schritt gewonnen werden. Dies wird mit $x_1 = x_0 + v_0 dt + \frac{1}{2} a dt^2$ berechnet.

3.2.8 void verPos(Gravk** body, int bcount, unsigned dt)

Berechnet aus den letzten beiden Positionen und der Beschleunigung, die aus der Force-Funktion folgt den neuen Positionswert und verschiebt die Werte im Array, um öfter hintereinander ausgeführt werden zu können.

3.2.9 void print()

Gibt in der Konsole Startparameter für Euler, und x_{n-1} bzw. Startparameter für Verlet aus.

3.3 main.cpp

Hier liegt der Kern des Programmes, die Abfrage der Eingabe.

3.3.1 Eingabeparameter

Zu Beginn werden die Eingabeparameter hierarchisch abgefragt. Hierbei wird ein Fehler ausgegeben, falls das Lösungsverfahren nicht richtig ausgewählt wird. Wenn man hingegen zum Beispiel den Iterationsparameter zu Null wählt, so wird das Programm ausgeführt, wobei nichts geschieht, da die Lösungsverfahren kein Mal angewendet werden.

3.3.2 Berechnung und Datenausgabe

In einer for-Schleife werden im Falle des Euler-Verfahrens die setVel und newPos Funktionen aufgerufen und für die Anzahl der Iterationen wiederholt. Im Falle des Verlet-Algorithmus wird nur verPos aufgerufen. Da die Iterationen schnell berechenbar sind und die Zeitschritte möglichst klein gewählt werden können, wird in eine Ausgabedatei nur eine reduzierte Anzahl an Positionen und Geschwindigkeiten gespeichert.

3.4 Visualisierung

Die graphische Ausgabe ist dem Benutzer selbst überlassen. Wir haben uns für eine Ausgabe mit Hilfe von Python entschieden. Verwendet wird Matplotlib, damit werden die einzelnen Zeitpunkte des Systems als Frames gespeichert und mit dem Programm FFmpeg als MP4 Video gespeichert. Die Anzahl der Körper ist hierbei egal, da wir die Größe des Arrays abfragen und diesen Wert nutzen, um die richtige Anzahl an Körpern zu plotten. Hierdurch mussten wir leider auf Label verzichten, da dies nicht mit der Flexibilität für verschiedene Systeme vereinbar ist.
Es ist technisch möglich, ein Python-Skript in das Programm einzufügen, da bei unserer Umsetzung aber auch FFmpeg genutzt wird, könnte dies für einen Benutzer dieses Programmes zusätzliche Schwierigkeiten mit sich ziehen. Das Visualisieren ist dem Benutzer selbst überlassen, da er die Ausgabedatei nur in ein Programm einlesen müsste.

4 Ergebnisse

Im Folgenden werden die wichtigsten Ergebnisse zusammengefasst und erläutert.

4.1 Programm

Das Programm hat eine konsolenbasierte Eingabe und benötigt für zehn Millionen Iterationen für drei Körper etwas weniger als 30 Sekunden.

4.2 Zweikörperproblem

Konkret wurde das Sonne-Mond System betrachtet. Die Simulation ergibt physikalisch Sinn, da der Impuls durch die Startwerte eine Bewegung des Gesamtsystems zur Folge hat. Bei wenigen Umläufen des Mondes sind noch keinen sichtbaren Effekte durch numerische Fehler sichtbar.

4.3 Dreikörperproblem

Das Sonne-Erde-Mond wirkt in der Simulation stabil und verhält sich wie erwartet.
Andere Systeme zeigen ein chaotisches Verhalten und werfen gegebenenfalls Körper aus dem System (Beispiel 3 Sonnenmassen), falls sich verschiedene Körper zu nahe kommen sollte und stark beschleunigen. Damit Energie und Impuls erhalten bleiben können, entfernen sie sich wie vermutet immer weiter. Falls man für ein chaotisches System die Startparameter um wenige Prozent ändert, so sieht man es in der Visualisierung normalerweise nicht mit bloßen Auge. Nach etwas Laufzeit kann man an einigen Stellen in der Visualisierung des Systems eine Abweichung erkennen.

4.4 Mehrkörperproblem

Eine Eingabe von mehr als drei Körpern funktioniert, man muss nur aufpassen, dass die Datei am richtigen Ort ist und nur Zahlen enthält und keine Potenzen, da sonst die Werte nicht richtig eingelesen werden können.
Mehrkörpersysteme verhalten sich für fast alle Anfangsparameter chaotisch. In symmetrischen Fällen kann es ein stabiles System geben (Beispiel 4 Körper).

4.5 Chaotisches Verhalten

Wie bereits beschrieben, hat eine kleine Veränderung der Startparameter größere Auswirkungen auf das System. Das Gesamtsystem ist in seiner Bewegung auch nicht vorhersehbar. Man kann von den Startparametern auf keine Bahn eines Körpers schließen.
Es spielt auch das Verfahren eine Rolle, ein System kann mit gleichen Startparametern und gleichen Iterationen und Schrittweiten (Beispiel 6) verschiedene Ausgänge haben. Der Fehler, der bei der Berechnung gemacht wird, ist unterschiedlich. Diese kleine Veränderung ändert das Gesamtbild des Systems.

5 Diskussion der Fehler

In diesem Abschnitt geht es um die Probleme und Schwierigkeiten das physikalische Dreikörperproblem numerisch mit einem Computerprogramm zu lösen.

5.1 Physikalische und mathematische Fehler

Die Systeme werden physikalisch vereinfacht betrachtet, trotzdem zeigen die Systeme, zum Beispiel Sonne-Erde-Mond, ein zu erwartendes Verhalten, deswegen scheinen die physikalischen Vereinfachung durchaus sinnvoll. Zwar ist die gravitative Wechselwirkung die Schwächste, aber in den Größenordnung von Planeten die dominierende.
Die Genauigkeit des Euler-Verfahren und des Verlet-Algorithmus sind beide von der Schrittweite einer

Iteration abhängig. Das Verlet-Verfahren hat den Vorteil aus einer Taylorentwicklung zu entstehen und keine Berechnung der Geschwindigkeit zu benötigen.
Es zeigt sich, dass Schrittweiten von einer Sekunde für Problem mit deutlich höheren Größenordnungen schon sehr genau sind.
Zudem muss beachtet werden, dass der Fehler und die damit resultierenden Abweichungen das System zu einem komplett anderen Ausgang bewegen kann. Vom Fehler hängt somit auch der Verlauf der Körperbewegung ab.

5.2 Schwierigkeiten

Physikalische Schwierigkeiten gibt es weniger. Das Newtonsche Gravitationsgesetz ist recht einfach umsetzbar, es neben Skalaren nur eine Vektoraddition gibt. Hingegen für Startparameter von Systemen muss man sich bewusst machen, welcher Körper sich wie relativ zum System bewegt. Auch bei der Startparameterübergabe muss man sich dies bewusst sein.
Die numerischen Verfahren sind einfach in der Idee und müssen in erster Linie nur passend mit for-Schleifen umgesetzt werden.
Am meisten Schwierigkeiten bereitete uns die Umsetzung in C++. Neben einfachen Syntaxfehlern muss man sich immer bewusst sein, was man physikalisch tut, um keine wertlosen Daten zu erhalten. Als am Schwersten hat sich Folgendes erwiesen.
Jeder Körper wird als Gravitationsklasse gespeichert und die Körper müssen abzählbar werden, um sie sinnvoll nutzen zu könne, vor allem, wenn man mehr als drei Körper betrachten möchte. So müssen diese zum Beispiel wiederum in einem Array gespeichert werden.
Besonders wichtig ist auch die Geschwindigkeit des Programmes. Da es von Vorteil ist, möglichst kleine Schritte zu wählen, müssen mehr Iterationen ausgeführt werden, damit man die gleiche Betrachtungsdauer für ein System erhält. Unser erster Ansatz war, std::vector zu nutzen, um alle berechneten Daten zu speichern. Das n-te Vektorelement ist also der n-te Iterationsschritt. Das belegt zum einen immer mehr Speicher, zum anderen verlangsamt dies die Berechnung neuer Daten. Ein Umstieg auf ein Array mit zwei Elementen, in dem der (n-1)-te Schritt immer wieder überschrieben wird und ab und zu ausgegeben wird, beschleunigt das Programm sehr. Im Bereich der höheren Iterationsschritte von $>$100.000 können mit Hilfe der Arrays 1000-mal mehr Schritte in kürzerer Zeit durchgeführt werden.

5.3 Fazit

Wir sind mit dem Programm insgesamt zufrieden. Die Berechnungsgeschwindigkeit finden wir besonders beachtenswert, weil unsere erste Methode keine besonders genaue Simulation ermöglichen konnte und auch schon einige Systeme damit nicht in sinnvollen Zeiträumen simulierbar waren.
Bekannte Systeme ,wie zum Beispiel Sonne-Erde-Mond, verhalten sich wie erwartet. Die Gravitationskraft wird richtig berechnet und funktioniert auch für mehr als drei Körper automatisch durch Schleifen.
Wir haben Einblicke in C++ erhalten und haben auch auch im Hinblick auf Debugging und Fehlersuche per Hand einiges gelernt. Das Programm ist gegen grobe Eingabefehler resistent (zu große Eingabeparameter könnten für int ein Problem darstellen).
Auch ist der physikalische Aspekt des Problems sehr interessant, man sieht und versteht die Erhaltung von Größen und sieht immer wieder unerwartetes und wird manchmal sogar überrascht.