Maturaarbeit

Simulation mechanischer Bewegungen mit Hilfe des Computers

eingereicht bei
Jörg Donth

im Fach
Physik

an der
Kantonsschule Reussbühl

vorgelegt von
Philip Wiese

Luzern, Oktober 2014

Inhaltsverzeichnis

1. Vorwort

Das Ziel dieser Arbeit ist die Anwendung mathematischer Verfahren der numerischen Mathematik und deren Umsetzung anhand eines Computerprogramms zu zeigen und zu erklären. Weiter möchte ich meine Programmierkenntnisse im Bereich des Multithreading, also der parallelen Verarbeitung mehrerer Threads im selben Prozess, erweitern. Das Computerprogramm ist in der Programmiersprache Java verfasst und wurde mit der Entwicklungsumgebung Eclipse[1] entwickelt.

Die Begriffe *Simulation* und *Computersimulation* werden in dieser Arbeit als Synonyme benutzt, da heutzutage der Begriff *Simulation* meist mit *Computersimulation* gleichgesetzt wird. Streng genommen jedoch ist eine Simulation nur dann eine Computersimulation, wenn sie mit Hilfe eines Computers durchgeführt wird.

Im Verlauf dieser Arbeit werde ich die verschiedenen Anwendungsbereiche der Simulation aufzeigen und näher auf die mathematischen Grundlagen der numerischen Simulation eingehen. Zusätzlich zu meiner Arbeit habe ich eine eigene Computersimulation zur Berechnung mechanischer Modelle programmiert, welche ich MyMechSim genannt habe. Das Computerprogramm und der Quellcode befinden sich auf der Begleit-CD unter *Java und Java/Code.* Die Benutzeranleitung befindet sich im Anhang und auf der Begleit-CD unter *Java/Benutzeranleitung*.

Sämtliche in dieser Arbeit thematisierten Beispiele sind in MyMechSim unter dem Menüpunkt *Datei -> Beispiele* aufruf- und simulierbar. Weiter sind alle in dieser Arbeit verwendeten Bilder von mir persönlich erstellt worden.

1.1 Danksagung

An dieser Stelle möchte ich mich bei all denjenigen bedanken, die mich während dieser Arbeit unterstützt haben.

Ganz besonders danke ich meinem Betreuer Herrn J. Donth, der mir den Einstieg in die Welt der Numerik gezeigt und mir dabei geholfen hat, stets einen kühlen Kopf zu bewahren. Obwohl er anfangs von meiner Programmierlust etwas überrascht war, haben wir beide einen guten Weg gefunden, aus dem diese Arbeit entstanden ist. Vielen Dank für Ihre Unterstützung und Geduld.

Weiter möchte ich mich bei meiner Familie und Freunden bedanken, die mich stets unterstützt und motiviert haben. Mein Dank gilt auch den Lektoren, die diese Arbeit durchgelesen und auf Fehler korrigiert haben.

Daneben gilt mein Dank auch den zahlreichen Internetusern, die in den Internetforen aktiv tätig sind und mir bei der Bewältigung zahlreicher technischer Probleme geholfen haben.

[1] Eclipse www.eclipse.org/

2. Einleitung

Computersimulationen spielen seit Mitte des 20. Jahrhunderts in der Wissenschaft eine immer grösser werdende Rolle. Auch wenn wir uns dessen nicht bewusst sind, begleiten uns Simulationen durch unserer ganzes Leben. So werden die Kolben neuer Automotoren zuerst am Computer simuliert, bevor sie in Produktion gehen und die Flügel eines Flugzeuges entstehen am Computer und werden in Simulationen auf ihre Funktion getestet.

Die Idee einer Simulation, bei der mit Hilfe numerischer Mathematik Lösungen approximiert werden, ist so alt wie die Mathematik selbst. Bereits Archimedes kannte Verfahren für die numerische Integration von Flächen[2].

Der Begriff der Simulation im modernen Sinn entwickelte sich jedoch erst ab 1941 mit der Erfindung des modernen Computers[3] durch Konrad Zuse und der Entwicklung der von-Neumann-Architektur durch Johann von Neumann. Johann von Neumann gilt als einer der Gründerväter der Informatik. 1949 wurde mit ENIAC der erste vollelektronische Rechner entwickelt, welcher von den Amerikanern unter anderem zur Entwicklung der Wasserstoffbombe und der Berechnung ballistischer Flugbahnen eingesetzt wurde[4]. Der nächste bedeutende Schritt war 1946 die Bildung des ersten Computernetzwerkes der Welt[5], aus dem sich unser heutiges Internet entwickelte. Die Amerikaner gewannen den Wettlauf um die Mondlandung nicht, weil sie die besseren Raketen bauten, sondern weil sie mit ihren leichten Bordrechnern durch numerische Integration die Flugbahn berechnen konnten (Hinrichsen, 2014). Seither entwickelte sich die Informatik rasant weiter. Dank schnell wachsenden Speicherkapazitäten und schnelleren Prozessoren können immer komplexere Simulationen durchgeführt werden.

[2] Streifenmethode www.dom-gymnasium.de/mathpage/themen/numintegr/
[3] Z3 - Konrad Zuse www.hs.uni-hamburg.de/DE/GNT/hh/biogr/zuse.htm
[4] ENIAC www.inventors.about.com/od/estartinventions/a/Eniac.htm
[5] ARPANET www.planet-wissen.de/wissen_interaktiv/html-versionen/geschichte_des_computers/

3. Anwendungsbereiche

Die Begriffe *Simulation* und *Computersimulation* werden oft mit den Bereichen Physik und Computerspielen assoziiert. Simulationen werden heutzutage jedoch in fast allen Gebieten der Naturwissenschaften und darüber hinaus angewendet.

Der Zweck einer Simulation ist, mit Hilfe eines Modells die zukünftige Entwicklung eines Systems vorauszusagen. Modelle stellen ein abstraktes Abbild eines Systems dar, da die Wirklichkeit oft zu komplex ist.

Im Grunde genommen ist Simulation nur das numerische Lösen von Differentialgleichungen, von denen keine analytische Lösung existiert. Dank unseren Computern sind wir in der Lage, mit Hilfe von Simulationen mathematische Probleme zu approximieren, die wir nicht lösen können. Dadurch eröffnen sich vollkommen neue Möglichkeiten, Probleme zu analysieren und neue Lösungsansätze bzw. Zusammenhänge zu entwickeln.

«Simulationen dienen 1. dem Versuch einer konsistenten Beschreibung verschiedener Einzelbefunde, 2. der Überprüfung der Eigenschaften von Hypothesen und 3. der Vorhersage von noch nicht untersuchten Eigenschaften des betrachteten Systems» (Spektrum Akademischer Verlag, 1999).

Das obige Zitat stammt aus dem *Lexikon der Biologie* des Spektrum Akademischer Verlag und bezieht sich auf die Biologie. Jedoch kann diese Definition problemlos allgemein auf den Begriff der Simulation angewendet werden.

3.1 Naturwissenschaften

Das Ziel der Simulation in den Naturwissenschaften besteht darin, Voraussagen über die Entwicklung eines Modells zu machen und bestehende Hypothesen zu bestätigen. Die Anwendung von Simulationen ist vor allem in den Bereichen der Biologie, Theoretischen Chemie, Physik, Astronomie und Meteorologie weit verbreitet.

In der Astronomie wird unter anderem die Entstehung unseres Universums simuliert, um die Bildung der Galaxien und Entstehung von Planetensystemen zu verstehen. Dadurch ist man in der Lage, genaue theoretische Voraussagen über das Wachstum von Galaxien und superschweren Schwarzen Löchern zu treffen (Springel, 2005).

3.2 Ökonomie

Aus ökonomischer Sicht liegt der grosse Vorteil der Simulation darin, vor der Umsetzung einer neuen Strategie oder Methode, diese zuerst auf ihre Tauglichkeit überprüfen zu können, um dadurch nicht unnötige Ressourcen zu verschwenden. So unterstützt die Fraport AG, welche die Flughafenbetreiberin des Frankfurter Flughafens ist, ein Forschungsprojekt zur Simulation neuer Logistik- oder Materialflussprozesse. Ziel dieser Simulation ist die Vollautomatisierung der Transportwege ganz ohne Schienen und baulich vorgegebenen Vorrichtungen. Dadurch würde die Effizienz beim Gepäcktransport zusätzlich gesteigert, da zu früh abgegebenes Gepäck auf einer Abstellfläche parkiert werden kann und nur bei Bedarf bewegt wird (Fraport AG, 2014).

3.3 Sozialwissenschaften

«Computersimulation ist in den Sozialwissenschaften fast ebenso so alt wie in allen anderen Wissenschaften auch: Kaum waren die ersten Computer verfügbar, wurden sie auch schon von Sozialwissenschaftlern eingesetzt» (Herz, et al., 2000).

Mit Hilfe von Simulationen versucht man unter anderem die Entwicklung der Weltbevölkerung und deren Ressourcen vorauszusagen. Eine der bekanntesten Simulation in den Sozialwissenschaften ist John Conway's Game of Life[6]. Dieses System ist eine zweidimensionale Simulation zellulärer Automaten mit einfachen Bedingungen:

- Eine Zelle ist entweder mit einem Lebewesen besetzt oder nicht
- Wenn eine Zelle besetzt ist:
  - Die Zelle überlebt, wenn sie 2 oder 3 Nachbarn hat
  - Die Zelle stirbt, wenn sie 0, 1, 4, 5, 6, 7 oder 8 Nachbarn hat
- Wenn die Zelle tot ist
  - Entsteht ein neues Lebewesen, wenn sie genau 3 Nachbarn hat

In den 1970er Jahren beschäftigten sich viele Personen mit dieser Simulation. Dank immer schnelleren Computern erlangte die Simulation Game of Life nach der Jahrtausendwende neue Bekanntheit. Mittlerweilen wurden auch dreidimensionale Varianten des Games of Life entwickelt.

[6] Game of Life Java-Applet www.denkoffen.de/Games/SpieldesLebens/
www.mathematische-basteleien.de/gameoflife.htm

4. Theoretische Grundlagen

In diesem Kapitel werden die theoretischen und mathematischen Funktionsweisen der verschiedenen Verfahren erläutert. Eine Simulation ist das numerische Lösen von Anfangswertproblemen, das heisst, das Lösen einer Differentialgleichung mit einer gegebenen Anfangsbedingung.

Zur Vereinfachung wird die Vektorschreibweise nur bis Gleichung (2) verwendet und ab Gleichung (3) nur der eindimensionale Bezug aufgezeigt. Die Kraftfunktion $\vec{F}(t)$ kann selbstverständlich beliebig viele Parameter enthalten. Einfachheitshalber wird in den folgenden Abschnitten nur die Zeit t als Parameter verwendet.

Das Grundprinzip der mechanischen Simulation ist das 2. Newtonsche Gesetz bzw. die Grundgleichung der Mechanik.

$$\vec{F} = \vec{a} \cdot m \tag{1}$$

$$\vec{a}(t) = \frac{\vec{F}(t)}{m} \tag{2}$$

Die Beschleunigung a kann als Ableitung der Geschwindigkeit v nach der Zeit t dargestellt werden.

$$a(t) = \dot{v} = \frac{dv}{dt} \tag{3}$$

$$dv = a(t) \cdot dt \tag{4}$$

Durch Integrieren beider Seiten der Gleichung erhalten wir:

$$v(t) = \int a(t') \cdot dt' = \int \frac{F(t')}{m} \cdot dt' \tag{5}$$

Analog zur Geschwindigkeit eines Körpers integrieren wir die Geschwindigkeitsfunktion $v(t)$:

$$v(t) = \dot{x} = \frac{dx}{dt} \tag{6}$$

$$dx = v(t) \cdot dt \tag{7}$$

$$s(t) = \int v(t') \cdot dt' \tag{8}$$

Da oft keine analytische Lösung dieses Problems existiert, muss man die Lösung durch numerisches Integrieren approximieren. Es existieren zahlreiche Methoden der numerischen Integration. Zu den bekanntesten zählen das explizite Euler-Verfahren (auch Euler-Cauchy-Verfahren), das Heun-Verfahren und das klassische Runge-Kutta-Verfahren (RGK 4. Ordnung)

Man kann zwischen Einschritt- und Mehrschrittverfahren unterscheiden. Die hier betrachteten Verfahren zählen alle zu den Einschrittverfahren. Das Grundprinzip dieser Verfahren ist die lineare Annäherung an den tatsächlichen Funktionsverlauf.

4.1 Explizites Euler-Verfahren

Das explizite Euler-Verfahren oder Euler-Cauchy-Verfahren zählt zu den einfachsten Einschrittverfahren. Das Ziel ist das Lösen der Differenzialgleichung der Form:

$$g'(t) = f(t, g(t)) \tag{1}$$

Bekannt ist die Funktion $\mathrm{f(t, g(t))}$ und der Funktionswert $g(t_0)$. Gesucht ist nun der Näherungswert von $g(t_0 + \Delta t)$. Mit Hilfe der Steigung, die der obigen Differentialgleichung (1) entnommen werden kann, ist es möglich eine lineare Hilfsfunktion $\mathrm{P(t)}$ zu ermitteln. Diese lineare Funktion lässt sich als "Lineare Fortsetzung" der Funktion g zum Zeitpunkt t_0 bis zum Zeitpunkt $t_0 + \Delta t$ intepretieren. Dieser Schritt wird so oft wie nötig wiederholt, bis der gewünschte Zeitpunkt erreicht ist.

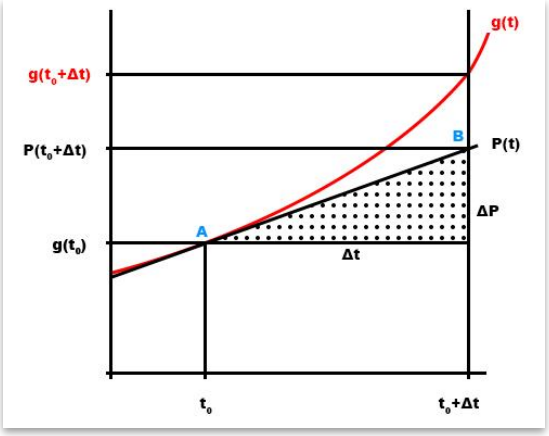


Abbildung 1: Explizites Euler-Verfahren

Die Allgemeine Definition des expliziten Euler-Verfahrens lautet:

$$\boldsymbol{g_{t+\Delta t} = g_t + \Delta t \cdot f(t, g(t))} \tag{2}$$

Konkret auf die Mechanik übersetzt ergibt sich für die Funktion $f(t, g(t))$:

$$f\big(t, g(t)\big) = \frac{F(t, v(t))}{m} \tag{3}$$

und somit für das explizite Euler-Verfahren:

$$v_{t+\Delta t} = v_t + \Delta t \, \frac{F(t, v(t))}{m} \tag{4}$$

Analog dazu kann nun die Position s des Objektes numerisch angenähert werden, wobei die Funktion $v(t)$ die Rolle von $\frac{F(t)}{m}$ übernimmt. Die Funktionswerte von $v(t)$ wurden im vorhergehenden Schritt bereits berechnet. Somit gilt:

$$f\big(t, g(t)\big) = v(t, s(t)) \tag{5}$$

und daher für das explizite Euler-Verfahren:

$$s_{t+\Delta t} = s_t + \Delta t \cdot v(t, s(t)) \tag{6}$$

Bereits in der Abbildung 1 ist zu erkennen, dass das Euler-Verfahren sehr ungenau ist. Die Abweichung vom tatsächlichen Funktionswert vergrössert sich kontinuierlich. Das Euler-Verfahren bei diskreten Systemen bietet somit nur bei kurzen Prognosezeiten ausreichende Genauigkeit. Wegen der Einfachheit des Verfahrens ist es jedoch sehr beliebt, um die Grundidee der numerischen Integration aufzuzeigen.

4.2 Heun-Verfahren

Mit sehr geringem Aufwand lässt sich das explizite Euler-Verfahren stark verbessern. Das Ergebnis ist das Heun-Verfahren. Der Unterschied zum Euler-Verfahren liegt in der Hilfsfunktion P(t). Beim Verfahren von Heun wird die Steigung der Hilfsfunktion P(t) durch das arithmetische Mittel zweier Steigungen errechnet. Dadurch fliesst in die Berechnung des Prognosewertes P(t) der weitere Verlauf der Funktion g ein, womit eine höhere Genauigkeit erreicht wird.

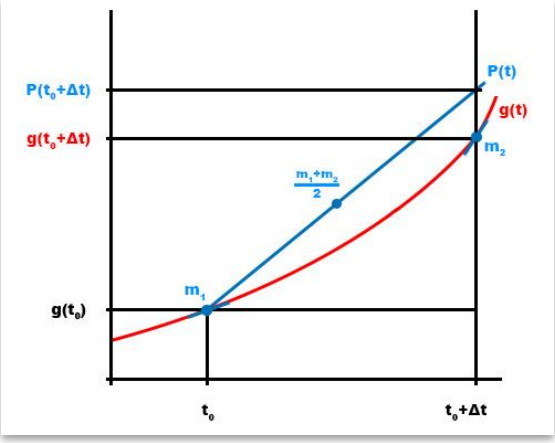


Abbildung 2: Heun-Verfahren

Die allgemeine Definition des expliziten Heun-Verfahren lautet:

$$f_0 = f\big(t, g(t)\big) \tag{1}$$

$$f_1 = f\big(t + \Delta t, g(t + \Delta t)\big) \tag{2}$$

$$\boldsymbol{g_{t+\Delta t} = g_t + \frac{\Delta t}{2}(f_0 + f_1)} \tag{3}$$

Auf die Mechanik übersetzt ergibt sich für die Geschwindigkeit v:

$$F_0 = F(t, v(t)) \tag{4}$$

$$F_1 = F(t + \Delta t, v(t + \Delta t)) \tag{5}$$

$$v_{t+\Delta t} = v_t + \frac{\Delta t}{2}\left(\frac{F_0 + F_1}{m}\right) \tag{6}$$

und weiter für die Position s:

$$v_0 = v(t, s(t)) \tag{4}$$

$$v_1 = v(t + \Delta t, s(t + \Delta t)) \tag{5}$$

$$s_{t+\Delta t} = s_t + \frac{\Delta t}{2}(v_0 + v_1) \tag{6}$$

Durch die Verwendung zweier Steigungen bzw. Differentialquotienten wird die Präzision des Heun-Verfahrens gegenüber dem Euler-Verfahren enorm gesteigert. Trotz der erhöhten Genauigkeit wird das Heun-Verfahren in der Praxis selten angewendet, da es im Vergleich zu höheren Verfahren zu ungenau ist. Der nächste Schritt ist die Verwendung weiterer Hilfssteigungen, um noch genauere Prognosewerte zu erhalten. Dies wird im klassischen Runge-Kutta-Verfahren bzw. dem Runge-Kutta Verfahren 4. Ordnung umgesetzt.

4.3 Klassisches Runge-Kutta-Verfahren

Das klassische Runge-Kutta-Verfahren bzw. Runge-Kutta-Verfahren 4. Ordnung verwendet ähnlich zum Heun-Verfahren Hilfssteigungen, um die Genauigkeit der Prognosewerte zu verbessern. Das gewichtete arithmetische Mittel der Hilfssteigungen ergibt die Steigung der Hilfsfunktion P. Durch die doppelte Gewichtung der beiden mittleren Steigungen wird die Genauigkeit des Prognosewertes gesteigert.

Die allgemeine Definition des expliziten Runge-Kutta-Verfahren lautet:

$$f_1 = f\big(t, g(t)\big) \tag{1}$$

$$f_2 = f\left(t + \frac{1}{2}\Delta t, g(t) + \frac{1}{2}\Delta t \cdot f_1\right) \tag{2}$$

$$f_3 = f\left(t + \frac{1}{2}\Delta t, g(t) + \frac{1}{2}\Delta t \cdot f_2\right) \tag{3}$$

$$f_4 = f(t + \Delta t, g(t) + \Delta t \cdot f_3) \tag{4}$$

$$\boldsymbol{g_{t+\Delta t} = g_t + \frac{1}{6}\Delta t(f_1 + 2f_2 + 2f_3 + f_4)} \tag{5}$$

Auf die Mechanik übersetzt ergibt sich:

$$F_1 = F\big(t, v(t)\big) \tag{6}$$

$$F_2 = F\left(t + \frac{\Delta t}{2}, v(t) + \frac{1}{2m}\Delta t \cdot F_1\right) \tag{7}$$

$$F_3 = F\left(t + \frac{\Delta t}{2}, v(t) + \frac{1}{2m}\Delta t \cdot F_2\right) \tag{8}$$

$$F_4 = F\left(t + \Delta t, v(t) + \frac{1}{m}\Delta t \cdot F_3\right) \tag{9}$$

$$v_{t+\Delta t} = v_t + \frac{\Delta t}{6}\left(\frac{F_0 + 2F_1 + 2F_2 + F_3}{m}\right) \tag{10}$$

und weiter für die Position s:

$$v_1 = v\big(t, s(t)\big) \tag{11}$$

$$v_2 = v\left(t + \frac{\Delta t}{2}, s(t) + \frac{1}{2}\Delta t \cdot v_1\right) \tag{12}$$

$$v_3 = v\left(t + \frac{\Delta t}{2}, s(t) + \frac{1}{2}\Delta t \cdot v_2\right) \tag{13}$$

$$v_4 = v(t + \Delta t, s(t) + \Delta t \cdot v_3) \tag{14}$$

$$s_{t+\Delta t} = s_t + \frac{\Delta t}{6}(v_1 + 2v_2 + 2v_3 + v_4) \tag{15}$$

Das Runge-Kutta-Verfahren 4. Ordnung gehört zu der Familie der Runge-Kutta-Verfahren. Die Runge-Kutta-Verfahren sind Einschrittverfahren zur numerischen Lösung von Anfangswertproblemen. Die Ordnung des Runge-Kutta-Verfahren kann theoretisch beliebig erhöht werden. Da die Genauigkeit im Verhältnis zur Rechenzeit bei sehr hohen Ordnungen immer ungünstiger wird, bedient man sich in der Praxis oft Mehrschrittverfahren. Mehrschrittverfahren nutzen bereits berechnete Prognosewerte, um eine noch genauere Näherung zu berechnen.

Das explizite Euler-Verfahren und das Heun-Verfahren sind beides Spezialfälle der Runge-Kutta-Verfahren.

5. Computerprogramm

In den folgenden Kapiteln soll die Funktionsweise des Computerprogramms MyMechSim aufgezeigt werden, welches sich auf der Begleit-CD unter *Java* befindet. Als Programmiersprache diente Java von Oracle[7] und die opensource[8] Entwicklungsumgebung Eclipse der Eclipse Foundation. Das Programm entstand in mehr als 80 Stunden Arbeit und besteht aus über 5'000 Zeilen Code. Zusätzlich wurde eine Benutzeranleitung erstellt, welche im Anhang und auf der Begleit-CD unter *Java/Benutzeranleitung* zu finden ist.

5.1 Warum Java?

Die Anzahl Programmiersprachen[9] wird auf 2'500 bis 8'000 geschätzt. Die verschiedenen Sprachen unterscheiden sich in ihrer Komplexität, Geschwindigkeit und Sicherheit, sowie in den unterstützten Konzepten und vielen weiteren Merkmalen. Zahlreiche entwickelte Sprachen werden heute nicht mehr gebraucht oder wurden bereits wieder überarbeitet. Java gehört jedoch zu den meist verwendeten Sprachen auf der Welt und liegt laut TIOBE[10] auf Platz 2 der beliebtesten Programmiersprachen [Stand August 2014].

Die Vorteile von Java sind:

- Plattformunabhängigkeit
- Einfach zu erlernen
- Multithreading
- Robust
- Objektorientiert
- Viele Bibliotheken

Java hat jedoch auch seine Nachteile. Dazu gehört, dass auf dem Zielrechner die Java Laufzeitumgebung installiert sein muss. Immer wieder stösst man zudem auf den Vorwurf, Java sei unsicher. Die Meinungen diesbezüglich gehen jedoch auch unter Experten weit auseinander.

5.1.1 Multithreading

Multithreading ist eines der wichtigsten Konzepte in den höheren Programmiersprachen und bezeichnet das gleichzeitige Abarbeiten mehrerer Befehlsstränge, sogenannten Threads, im gleichen Programm. Diese Gleichzeitigkeit ist jedoch nur scheinbar. In der Realität wird jedem Thread eine kurze Zeitspanne zur Verfügung gestellt und danach zum nächsten Thread gewechselt.

Eine mögliche Anwendung des Multithreading wäre, in einem Thread Berechnungen durchzuführen und einen zweiten Thread auf die Eingaben des Benutzers reagieren zu lassen. Dadurch erhält das Computerprogramm mehr Flexibilität und wirkt moderner.

---

[7] Oracle und Java www.oracle.com/de/technologies/java/overview/index.html
[8] Open source www.de.wikipedia.org/wiki/Open_Source
[9] Programmiersprachen www.informatik.uni-mainz.de/arbeitsgruppen/programmiersprachen-und-internet-technologie
[10] TIOBE www.tiobe.com/index.php/content/paperinfo/tpci/index.html

Auf weitere wichtige Konzepte wie Objektorientierung, Datenkapselung und ereignisorientierte Programmierung wird in dieser Arbeit nicht weiter eingegangen, da es den Rahmen einer Maturaarbeit sprengen würde.

5.2 Funktionsweise

5.2.1 Aufteilung in Threads

MyMechSim besteht aus vier grundlegenden Bausteinen bestehend aus Steuerung, Berechnung, Anzeige in der Tabelle und Anzeige im Graphen. Jeder dieser Bausteine ist in einem eigenen Thread realisiert, welche parallel zu einander ausgeführt werden. Der Grund für diese Aufteilung liegt im Geschwindigkeitsunterschied der einzelnen Bereiche. Es ist möglich, dass die Berechnung der Daten viel schneller erfolgt, als dass die Darstellung diese verarbeiten und visualisieren kann oder genau umgekehrt. Durch eine Aufteilung in diese vier Bereiche wird einerseits die Rechenzeit verkürzt und andererseits "friert" das Programm während der Berechnung nicht ein und kann daher immer noch auf Benutzereingaben reagieren.

Konkret wurde die Datenübermittlung mit Pipes, genauer mit Blocking Queues realisiert. Eine Pipe ist ein Konzept, mit der Daten von einem Thread A zu einem Thread B übermittelt werden können. Die Daten vom Thread A werden in der Pipe gespeichert. Die Speicherkapazität der Pipe vergrössert sich dabei dynamisch, in Abhängigkeit der Datenmenge, die gespeichert werden muss. Thread B kann nun die Daten in der Pipe auslesen. Eine weitere spezielle Eigenschaft der Pipes ist, dass ein Thread, der eine leere Pipe auslesen will, eingefroren wird und wartet, bis neue Daten verfügbar sind.

Bildlich gesprochen, kann man die Pipe als Kiesablagefläche betrachten und die Threads sind die Bagger, die den Kies nehmen und weiterverarbeiten. Wenn kein Kies mehr vorhanden ist, warten die Bagger, bis eine neue Ladung eintrifft und nehmen daraufhin ihre Arbeit wieder auf.

Die Threads *GUITable*, *Graphen* und *Simulation_Math* wurden als Daemon Threads konfiguriert. Dadurch beendet sich das Computerprogramm, sobald der Hauptthread, die Graphikoberfläche, beendet wird. Daher ist es auch möglich, in *GUITable* und *Graphen* eine Endlosschleife zu programmieren, die Daten aus der Pipe ausliest und diese verarbeitet.

Im Code sieht dies folgendermassen aus:

Simulation_Math.java

```
private void showAndPut(...) {
        for (int j=0;j<this.BewegteObjekte.size();j++) {
                (...)
                this.simulationToGraphPipe.put(daten);                    (1)
                if (objektSchowInTable.equals(objekt)){
                        this.simulationToTablePipe.put(daten);            (2)
                }
                (...)
        }
}
```

(1)&(2)	Rechnerdaten werden in die beiden Pipe eingespeist.

Wie oben ersichtlich ist, benötigt man je eine Pipe pro Thread, mit dem man kommunizieren möchte, da die Daten, die ein Thread aus einer Pipe ausliest, daraufhin aus der Pipe gelöscht werden.

GUITable.java

```
public void run() {
        (...)
        while(true) {                                                     (1)
                try {
                        objekt = pipe.take();                             (2)
                } catch (InterruptedException e) {}
                (...)
        }
}
```

(1)	Endlosschleife
(2)	Rechendaten werden aus der Pipe ausgelesen

Im Thread *Graphen* sieht die Endlosschleife gleich aus wie oben.

5.2.2 Berechnung

Die Berechnung wurde in der Klasse *BewObjekt* programmiert. Aus zeitlichen Gründen wurde nur das explizite Euler-Verfahren und das Heun-Verfahren implementiert. Dies wurde folgendermassen umgesetzt:

Beschleunigung ausrechnen

BewObjekt.java

```
public void NewKraft() throws ArithmeticException {
        setParameter(0);
        this.f_x = math.evaluate(this.glF_x);                    (1)
        this.f_y = math.evaluate(this.glF_y);
        this.f_z = math.evaluate(this.glF_z);
}
public void NewBeschleunigung() {
        if (masse!=0) {
                this.a_x = 1/masse * this.f_x;
                this.a_y = 1/masse * this.f_y;                   (2)
                this.a_z = 1/masse * this.f_z;
        }else{
                this.a_x = 0;
                this.a_y = 0;
                this.a_z = 0;
        }
}
```

(1)	Kraft zum Zeitpunkt t berechnen
(2)	Beschleunigung nach Newton ausrechnen. Falls die Masse 0 ist, Beschleunigung 0 setzen, um einen Programmabsturz zu verhindern.

Eine mathematische Gleichung ist für den Computer nichts anderes als eine Zeichenkette. Um diese für den Computer verständlich zu machen, muss man sie zuerst mit einem Parser (engl. *to parse,* "analysieren") umwandeln. Der Parser wurde nicht selber entwickelt. sondern von einer Drittperson[11] bezogen, da die Entwicklung eines vollständigen Parsers zu komplex für diese Maturaarbeit wäre.

[11] Math Expression Evaluator www.softwaremonkey.org/code/MathEval

Explizites Euler-Verfahren

BewObjekt.java

```
public void NewGeschwindigkeit(double h) {                    (1)
        this.v_x_old = this.v_x;                              (2)
        this.v_y_old = this.v_y;
        this.v_z_old = this.v_z;

        switch (this.GetVerfahren)) {
                case "Euler":
                        this.v_x = this.v_x + this.a_x * h;   (3)
                        this.v_y = this.v_y + this.a_y * h;
                        this.v_z = this.v_z + this.a_z * h;
                        break;
        (...)
        }
}
```

(1)	Parameter h für die Schrittweite
(2)	Die aktuelle Geschwindigkeit wird für die Berechnung der Position zwischengespeichert
(3)	Berechnung der neuen Geschwindigkeit mit dem Euler-Verfahren

BewObjekt.java

```
public void NewPosition(double h) {
        switch (this.GetVerfahren()) {
        case "Euler":
                this.p_x = this.p_x + this.v_x_old * h;       (1)
                this.p_y = this.p_y + this.v_y_old * h;
                this.p_z = this.p_z + this.v_z_old * h;
                break;
        (...)
        }
}
```

(1)	Berechnung der neuen Position mit der alten Geschwindigkeit mit dem Euler-Verfahren

Heun-Verfahren

```
BewObjekt.java
public void NewGeschwindigkeit(double h) {
        (...)
        switch (this.GetVerfahren()) {
                (...)
                case "Heun":
                        double g0X = this.a_x;                          (1)
                        double g0Y = this.a_y;
                        double g0Z = this.a_z;
                        double g1 = 0;
                        setParameter(h);                                (2)

                        g1 = 1/masse * math.evaluate(this.glF_x);       (3)
                        this.v_x = this.v_x + h/2 * (g0X + g1);         (4)

                        g1 = 1/masse * math.evaluate(this.glF_y);
                        this.v_y = this.v_y + h/2 * (g0Y + g1);

                        g1 = 1/masse * math.evaluate(this.glF_z);
                        this.v_z = this.v_z + h/2 * (g0Z + g1);
                        break;
                (...)
        }
}
```

(1)	Kraft zum Zeitpunkt t
(2)	Parameter der Kraftgleichung werden auf den Zeitpunkt $t + h$ gesetzt
(3)	Kraft zum Zeitpunkt $t + h$
(4)	Berechnung der neuen Geschwindigkeit mit dem Heun-Verfahren

```
BewObjekt.java
public void NewPosition(double h) {
        switch (this.GetVerfahren()) {
                (...)
                case "Heun":
                        double g0 = 0;
                        double g1 = 0;

                        g0 = this.v_x_old;                              (1)
                        g1 = this.v_x;                                  (2)
                        this.p_x = this.p_x + h/2 * (g0 + g1);          (3)

        Analog zum obigen Code werden die y und z Koordinaten berechnet
        (...)
        }
}
```

(1)	Geschwindigkeit zum Zeitpunkt t
(2)	Geschwindigkeit zum Zeitpunkt $t + h$ (bereits berechnet)
(3)	Berechnung der neuen Position mit dem Heun-Verfahren

Das Herzstück des Programms besteht mehr oder weniger aus diesen wenigen Zeilen Code. Dadurch ist es relativ einfach, weitere Verfahren zu implementieren. Dies wurde aus zeitlichen Gründen jedoch nicht getan. Der Rest des Quellcodes dient der Grafikoberfläche und der Datenverarbeitung, wobei die Grafik den grösseren Teil darstellt.

6. Vergleich der Verfahren

Um den Fehler eines Verfahrens zu betrachten, muss man zwischen Diskretisierungs- und Rundungsfehler unterscheiden. Der Diskretisierungsfehler gibt an, wie weit der erhaltene Wert für einen Integrationsschritt vom theoretisch exakten Wert abweicht. Rundungsfehler entstehen bei der Berechnung mit Hilfe des Computers und können nicht ganz vermieden werden. Es existieren jedoch spezielle Programmiersprachen und Prozessoren, um diesen Fehler zu minimieren. Diese Methoden standen leider nicht zur Verfügung oder sind zu komplex.

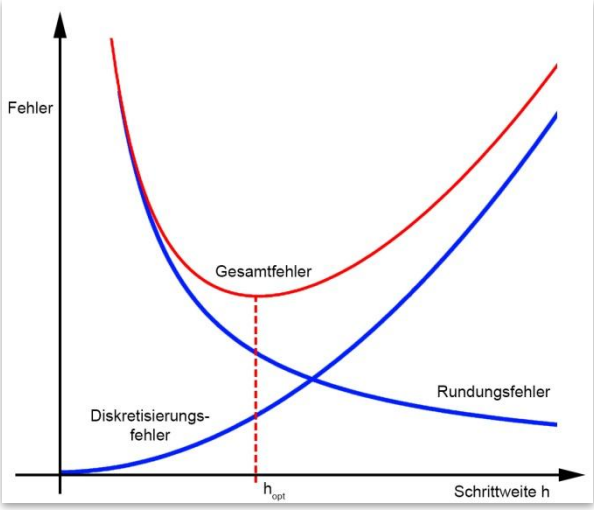


Abbildung 3: Gesamtfehler

Wie in Abbildung 3 ersichtlich ist, führt eine Verkleinerung der Schrittweite h zu einer Abnahme des Diskretisierungsfehlers, jedoch zu einer Zunahme des Rundungsfehlers. Dadurch ergibt sich eine optimale Schrittweite h_{opt}, bei der die Summe der Diskretisierungs- und Rundungsfehler minimal ist.

Um die Verfahren mit Hilfe meines Computerprogramms zu vergleichen, werden Systeme simuliert, die auch analytisch exakt gelöst werden können. Dies ist nötig, um die Grösse des Gesamtfehlers zu bestimmen.

6.1 Freie ungedämpfte Schwingung

6.1.1 Analytische Lösung

Das erste Anwendungsbeispiel beinhaltet die freie ungedämpfte Schwingung eines Federpendels. Die Kraftgleichung des Federpendels lautet:

$$F = D \cdot y \tag{1}$$

Der Proportionalitätsfaktor D ist die Federkonstante und y bezeichnet den Weg. Die Lösung der Schwingungsgleichung lautet:

$$y(t) = A \cdot sin(w_0 \cdot t + \varphi) \tag{2}$$

$$w_0 = \sqrt{\frac{D}{m}} = \textit{ungedämpfte Eigenkreisfrequenz und } \varphi = \textit{Anfangswinkel} \tag{3}$$

Der Faktor A ist die Amplitude der Schwingung. In unserem System ist die Amplitude A gleich der Anfangsauslenkung a. Da das Pendel nicht beim Nullpunkt zu schwingen beginnt, sondern bei der Anfangsauslenkung a, verschiebt sich die Schwingung um $\frac{\pi}{2}$.

$$y(t) = A \cdot sin\left(w_0 \cdot t + \frac{\pi}{2}\right) = A \cdot cos(w_0 \cdot t) \tag{4}$$

6.1.2 Simulation und Fehlerbetrachtung

In Tabelle 1 sind die Parameter der Simulation abgebildet. In Tabelle 2 ist ersichtlich, dass das Heun-Verfahren dem expliziten Euler-Verfahren klar überlegen ist.

Simulationsparameter		Objektparameter	
Startzeit [s]	*0*	Fx [N]	*-0.5*Px*
Endzeit [s]	*20*	Px [m]	*5*
Iterationen	*1'000'000*	M [kg}	*1*
Zeitschritte [s]	*0.00002*	Fy = Fz = Vx = Vy = Px = Py	*0*

Tabelle 1

Der Einfluss der Rundungsfehler lässt sich leider nicht darstellen, da das Programm mit dem Datentyp *double* rechnet. Der Datentyp *double* hat eine Auflösung von 64bit. Somit können Zahlen zwischen ca. $-4.9 \cdot 10^{-324}$ bis $1.8 \cdot 10^{308}$ in einem *double* gespeichert werden. Da die Anzahl Iterationen und die Endzeit aus Gründen der Rechenzeit und dem Aufbau des Programms nicht beliebig klein gewählt werden können, ist es nicht möglich, die Rundungsfehler sichtbar zu machen.

Zeit	Exakte Lösung [m]	explizites Euler-Verfahren [m]	Heun Verfahren [m]	Fehler explizites Euler-Verfahren [m]	Fehler Heun-Verfahren [m]
00	5	5	5	0.00E+00	0.00E+00
04	-4.756815641	-4.756910778	-4.756815641	-9.51E-05	-3.64E-11
08	4.050918016	4.051080054	4.050918016	1.62E-04	1.38E-10
12	-2.95097243	-2.951149491	-2.95097243	-1.77E-04	-2.86E-10
16	1.563974708	1.564099827	1.563974708	1.25E-04	4.48E-10

Tabelle 2

6.2 Freie gedämpfte Schwingung

6.2.1 Analytische Lösung

Bei der freien gedämpften Schwingung existieren je nach Grösse der Masse m, dem Reibungskoeffizienten r und der Federkonstante D drei verschiedene Fälle.

Schwache Dämpfung	$\delta < w_0$	$\delta = \frac{r}{2m^2}$ *und* $W = \sqrt{\frac{D}{m} - \frac{r^2}{4m^4}}$
aperiodischer Grenzfall	$\delta = w_0$	
Kriechfall	$\delta > w_0$	

Tabelle 3

Die allgemeine Lösung der Schwingungsgleichung im Fall der schwachen Dämpfung lautet:

$$y(t) = A \cdot e^{-\delta t} \cdot (cos(w \cdot t) + \frac{\delta}{w} sin(w \cdot t)) \quad (1)$$

6.2.2 Simulation und Fehlerbetrachtung

Simulationsparameter		Objektparameter	
Startzeit [s]	*0*	Fx [N]	*-0.5*Px-0.15*Vx*
Endzeit [s]	*10*	Px [m]	*1*
Iterationen	*1'000'000*	M [kg}	*1*
Zeitschritte [s]	*0.00001*	Fy = Fz = Vx = Vy = Vz = Py = Pz	*0*

Tabelle 4

Auch in diesem Beispiel ist die Überlegenheit des Heun-Verfahrens gegenüber dem Euler-Verfahren klar ersichtlich.

Im Simulationsgraphen lässt sich sehr schön die Natur der gedämpften Schwingung erkennen:

Zeit	Exakte Lösung [m]	explizites Euler-Verfahren [m]	Heun Verfahren [m]	Fehler explizites Euler-Verfahren [m]	Fehler Heun-Verfahren [m]
00	1	1	1	0.00E+00	0.00E+00
10	0.380541402	0.380580405	0.380541402	3.90E-05	-1.59E-10
20	0.040410097	0.040408332	0.040410097	-1.76E-06	-1.45E-10
30	-0.056924643	-0.056952588	-0.056924643	-2.79E-05	-4.10E-11
40	-0.048439321	-0.048464338	-0.048439321	-2.50E-05	2.86E-11
50	-0.020844577	-0.020855842	-0.020844577	-1.13E-05	4.05E-11

Tabelle 5

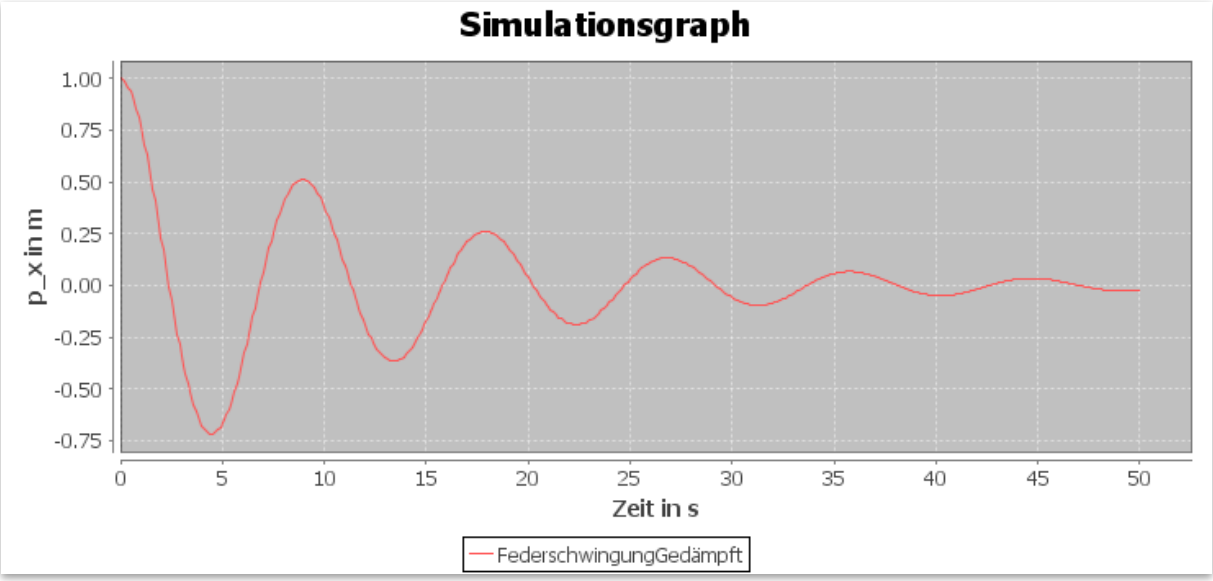


Abbildung 4: Freie gedämpfte Schwingung

7. Beispiele

7.1 Erzwungene Schwingung

Ein sehr schönes Anwendungsbeispiel ist die erzwungene Schwingung, in Abbildung 5 zu erkennen. Bei der erzwungenen Schwingung wird ein schwingungsfähiges System von aussen periodisch angeregt. Nach einem Einschwingvorgang geht das System in eine stationäre erzwungene Schwingung[12] über.

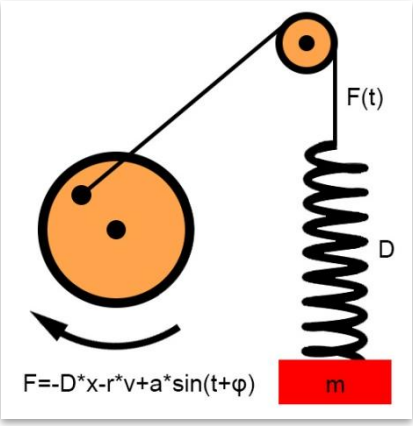


Abbildung 5

Die Amplitude der Schwingung hängt von der Grösse der Dämpfung und der Frequenz der Anregung ab. Dabei gilt: Je näher die Anregungsfrequenz an der Eigenfrequenz des Systemes liegt (mit einer leichten Verschiebung) und je kleiner die Dämpfung ist, desto grösser ist die Amplitude der Schwingung. Ohne Dämpfung divergiert die Amplitude bei der richtigen Anregunsfrequenz ins Unendliche.

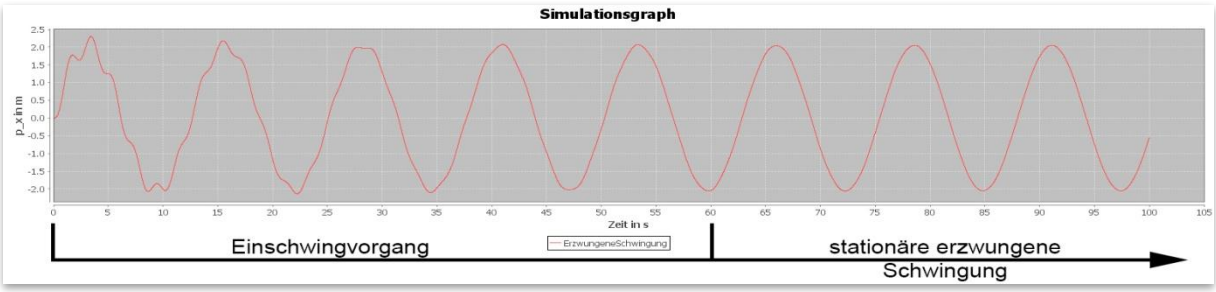


Abbildung 6: Erzwungene Schwingung

In Abbildung 7 ist ein Beispiel ohne Dämpfung dargestellt. Die Anregungsfrequenz liegt hier in der Nähe der Eigenfrequenz des Systems. Die Amplitude der Schwingung divergiert daher ins Unendliche.

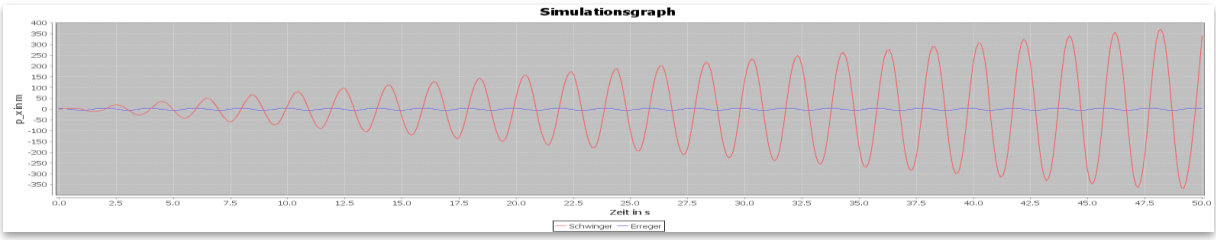


Abbildung 7: Divergente erzwungene Schwingung

Die erzwungene Schwingung hat eine grosses Bedeutung in unserem Alltagsleben. Das beste Beispiel hierfür ist die Federung eines Autos. Das Auto ist dabei der Schwinger und wird von den Bodenwellen, die nie ganz vermieden werden können, angeregt. Dies ist auch der Grund für die starke Dämpfung der Autofeder, da sich das Auto sonst von der Strasse abheben würde, was fatale Folgen hätte. Das Ziel dieser Simulationen ist es, den Einschwingvorgang möglichst kurz und die Amplitude der stationären erzwungenen Schwingung möglichst klein zu halten (s. Abbildung 6).

[12] Erzwungene Schwingung www.de.wikipedia.org/wiki/Erzwungene_Schwingung

7.2 Nichtlineare gedämpfte Schwingung

Ein schönes Beispiel für eine nichtlineare Schwingung ist die Konstruktion in Abbildung 8.

Abhängig von den Parametern $m, D, b, r\ und\ a$ hat das System eine oder drei reelle Gleichgewichtslagen. Wenn drei Gleichgewichtslagen existieren, sind zwei ein stabiles und eine ein instabiles Gleichgewicht. Dies lässt sich mit einer einfachen Überlegung nachvollziehen (s. Abbildung 9).

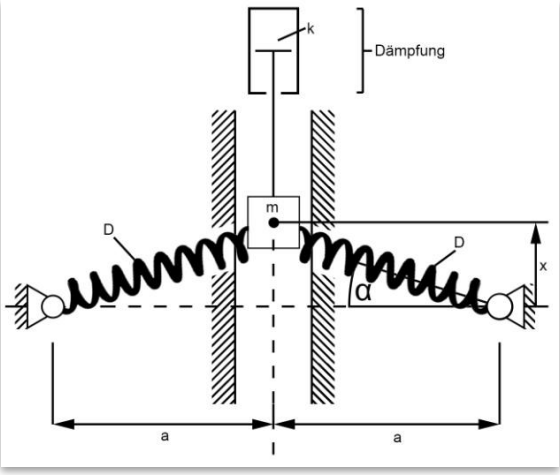


Abbildung 8: Nichtlineare Schwingung

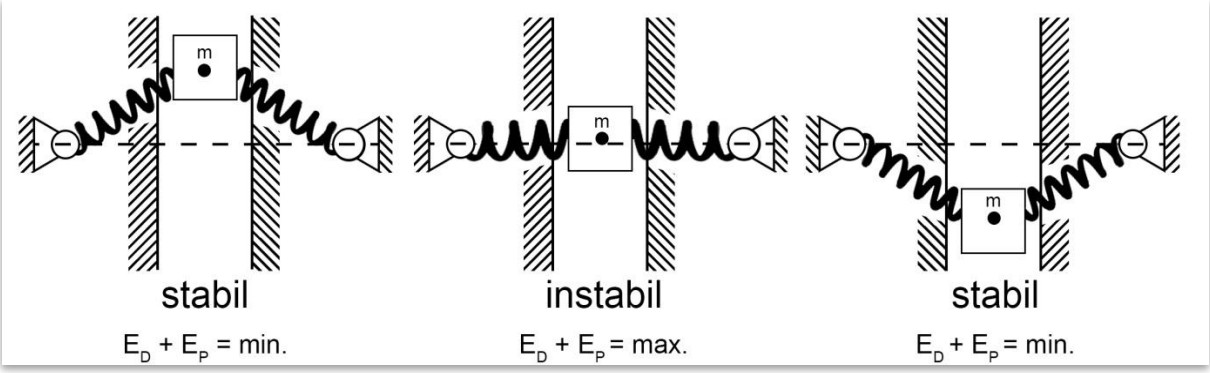


Abbildung 9: Gleichgewichtslagen

Wegen der Dämpfung wird der Schwinger nach einer gewissen Zeit fast immer in eine der beiden stabilen Ruhelagen gelangen.

Der Kräfteansatz diese Systems ist wie folgt:

$$F = -m \cdot g - 2 \cdot F_D \cdot \sin\alpha - r \cdot v \tag{1}$$

$$F_D = D\left(\sqrt{a^2 + x^2} - b\right)\ \ und\ \ \sin\alpha = \frac{x}{\sqrt{a^2+x^2}} \tag{2}$$

$$F = -m \cdot g - 2 \cdot D\left(\sqrt{a^2 + x^2} - b\right) \cdot \frac{x}{\sqrt{a^2 + x^2}} - r \cdot v \tag{3}$$

$b = Länge\ der\ entspannten\ Feder,\ D = Federkonstante,\ r = Reibungskoeffizient$

7.2.1 Simulation

Parameter					
a [m]	*2*	g [N/kg]	*9.81*	D [N/m}	*490.5*
b [m]	*2.4*	m [kg]	*1*	r [kg/s]	*0.5*

Tabelle 6

Simulationsparameter		Objektparameter	
Startzeit [s]	*0*	Fx [N]	*-9.81-2*490.5*(sqrt(2^2+Px^2)-2.4) *Px/(sqrt(2^2+Px^2))-0.5*Vx*
Endzeit [s]	*8*	Px [m]	*3*
Iterationen	*1'000'000*	M [kg}	*1*
Zeitschritte [s]	*0.000008*	Fy = Fz = Vx = Vy = Vz = Py = Pz	*0*

Tabelle 7

Mit den Parametern in Tabelle 6 hat das System die drei Gleichgewichtslagen x_1=-1.359, x_2=0.501 und x_3=1.293. Eine Simulation dieses Systems ergibt folgenden Graphen:

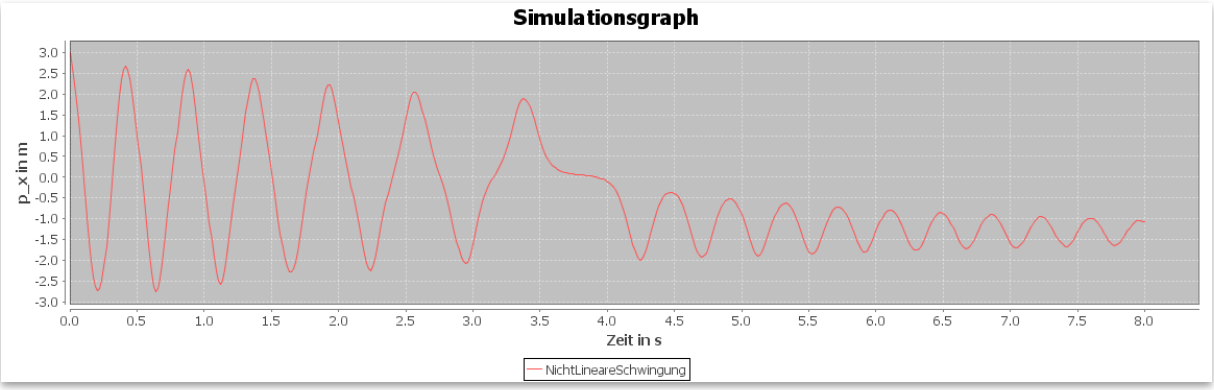


Abbildung 10: Nichtlineare gedämpfte Schwingung

In Abbildung 10 erkennt man bei ca. 3.7 Sekunden die Eigenheit des instabilen Gleichgewichtes. Der Schwinger nähert sich dem instabiles Gleichgewicht an. Da der Schwinger aber noch eine Restgeschwindigkeit hat, verlässt er sie wieder. Die Restenergie reicht jedoch nicht mehr aus, um das mittlere instabile Gleichgewicht zu durchqueren. Nach 3.7 Sekunden schwingt der Schwinger nur noch im unteren Bereich und nähert sich langsam dem unteren stabilen Gleichgewicht an.

Eine kleine Änderung der Anfangsbedingungen kann ein vollkommen anderes Endergebnis hervorbringen. Dies kann jedoch auch durch Fehler in der Simulation geschehen, die sich nie ganz vermeiden lassen. In der folgenden Simulation wurde der Reibungskoeffizient r um 0.01 auf 5.01 $\frac{\mathrm{kg}}{\mathrm{s}}$ geändert. Diese kleine Änderung der Anfangsbedingung hat zur Folge, dass der Schwinger nun dem oberen stabilen Gleichgewicht zustrebt. In solchen kritischen Fällen muss der Wahrheitsgehalt der Simulation kritisch hinterfragt werden.

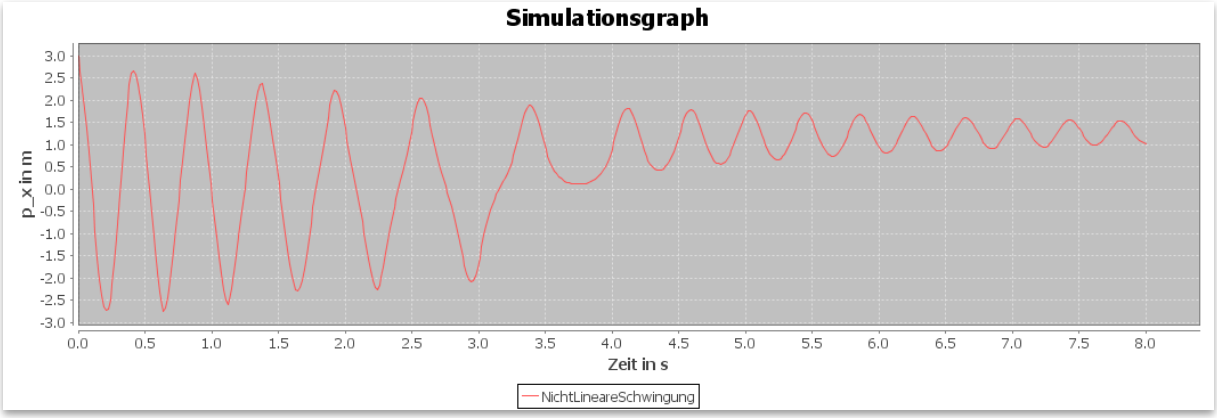


Abbildung 11: Nichtlineare gedämpfte Schwingung

7.2.2 Energiebetrachtung

Wenn die Reibungskraft weggelassen wird, gilt der Energieerhaltungssatz. Er besagt, dass in einem abgeschlossenen System die Summe aller Energien konstant ist. Die Energie unseres Systems setzt sich aus potentieller und kinetischer Energie zusammen.

$$E_G = E_{P,Masse} + E_{K,Masse} + E_{P,Feder} = konst. \tag{1}$$

$$E_P = mgx + \frac{1}{2}mv^2 + \frac{1}{2}D(\sqrt{a^2 + x^2} - b)^2 = konst. \tag{2}$$

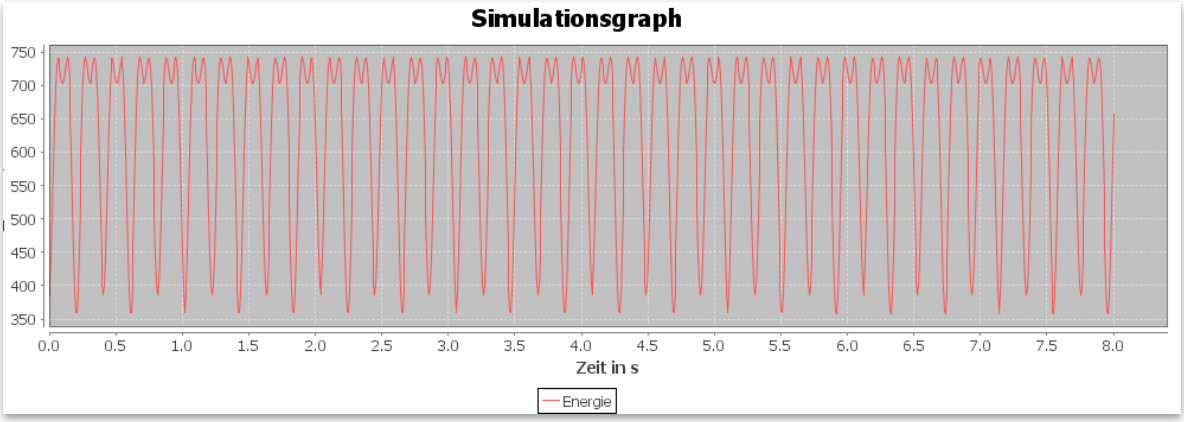


Abbildung 12: Energiebetrachtung

Da die Simulation nicht vollkommen exakt ist, ist die Gesamtenergie des Systems nicht konstant, sondern unterliegt gewissen Schwankungen. Erhaltungssätze der Physik werden oft verwendet, um die Zuverlässigkeit einer Simulation zu untersuchen.

8. Fazit

Das Ziel, eine mechanische Simulation mit Hilfe des Computers zu erstellen, wurde erreicht. Mit Hilfe meines Computerprogramms und dieser Arbeit ist es möglich, die beiden implementierten Verfahren zu verstehen und miteinander zu vergleichen. Das Programm ist selbstverständlich keine professionelle Anwendung für den Alltagsgebrauch. Jedoch ist es möglich, einzelne Systeme zu simulieren und diese graphisch darzustellen.

Während der Programmierphase habe ich sehr viele Bereiche des objektorientierten Programmierens kennengelernt und eine sehr grosse Sicherheit in der Programmiersprache Java erlangt. Obwohl ich den Grossteil der Zeit mit dem Programmieren zugebracht habe, bereiteten mir die theoretischen Grundlagen am meisten Schwierigkeiten. Dies liegt daran, dass wir erst in der 6. Klasse die Differentialgleichungen und Numerik behandeln und ich sehr viel alleine erarbeiten musste. Nach dieser Arbeit habe ich einen relativ kleinen Einblick in den grossen Bereich der Numerik erhalten und verstehe jetzt die grundlegendsten Konzepte und Verfahren. Mit dieser Arbeit ist mir auch klar geworden, warum eine Simulation nur eine Näherung ist und ihr Wahrheitsgehalt stets kritisch betrachtet werden muss.

Die Mechanik hat sich mir durch diese Arbeit als höchst interessantes und teilweise überraschendes Gebiet offenbart. Es ist erstaunlich, welche komplexen Bewegungen sich aus einfachen Kraftgleichungen ergeben können.

Das von mir erstellte Computerprogramm könnte noch stark erweitert und verbessert werden. Zudem sind noch nicht alle Fehler vollständig behoben und es kann immer noch zu Abstürzen kommen. Momentan sind jedoch alle wichtigen Funktionen implementiert und unter normalen Umständen auch ohne Probleme zu gebrauchen.

9. Literaturverzeichnis

Dankert, Jürgen. Technische Mechanik. [Online] [Zitat vom: 24. August 2014.] http://www.tm-aktuell.de/TM5/NLGedSchw/NLGedSchw.html.

Dankert, Jürgen und Dankert, Helga. 2011. *Technische Mechanik.* Wiesbaden : Vieweg+Teubner Verlag, 2011. S. 733-735. ISBN: 978-3-8348-1375-6.

Faires, J. Douglas und Burden, Richard L. 2000. *Numerische Methoden.* s.l. : Spektrum Akademischer Verlag, 2000. ISBN: 978-3-8274-0596-8.

Fraport AG. 2014. Fraport AG. [Online] 19. Februar 2014. [Zitat vom: 28. April 2014.] http://www.fraport.de/de/presse/newsroom/pressemitteilungen/forschungsobjekt-reisegepaeck--neue-wege-in-der-logistik.html.

Frotscher, Ralf und Pflug, Hans-Joachim. 2010. *Skript zum Kurs "Multithreading und Grafikprogrammierung in Java un C#" für Mathematisch-Technische Softwareentwickler.* [PDF] Aachen : RWTH Aachen, 2010.

Herz, Dietmar und Blätte, Andreas. 2000. *Simulation und Planspiel in den Sozialwissenschaften: eine Bestandsaufnahme der internationalen Diskussion.* Münster : LIT Verlag, 2000. ISBN: 978-3-8258-4752-4.

Hinrichsen, Prof. Dr. Haye. 2014. *Computational Physics.* [PDF] Würzburg : Universität Würzburg, 6. Februar 2014.

Kohorst, Helmut, Portscheller, Ph. und Goldkuhle, P. Helmut Kohorst. [Online] [Zitat vom: 29. 5 2014.] http://kohorst-lemgo.de.

Spektrum Akademischer Verlag. 1999. Spektrum.de. [Online] 1999. http://www.spektrum.de/lexikon/biologie/computersimulation/15019.

Springel, Volker. 2005. Max-Planck-Gesellschaft. [Online] 2005. [Zitat vom: 28. April 2014.] http://www.mpg.de/864208/forschungsSchwerpunkt1?c=166500.

Wenzel, Horst. 1991. *Mathematik für Ingenieure, Naturwissenschaftler, Ökonomen und sonstige anwendungsorientierte Berufe. Band 7/1: Gewöhnliche Differentialgleichungen 1.* Leipzig : Teubner Verlagsgesellschaft, 1991. ISBN: 978-3871444234.

A. Anhang

Benutzerhandbuch MyMechSim

Autor:	Philip Wiese
Stand:	30. Sept 2014
Version Anleitung:	0.3.0
Version Anwendung:	1.1.0
Kontakt:	philip-wiese@hotmail.com

Inhalt

1. Einleitung

MyMechSim ist eine Java Anwendung zur Simulation mechanischer Bewegungen. MyMechSim entstand im Zuge meiner Maturaarbeit, welche sich mit der Simulation mechanischer Bewegungen mit Hilfe des Computers befasst.

Die aktuellste Version kann von meiner Dropbox heruntergeladen werden.

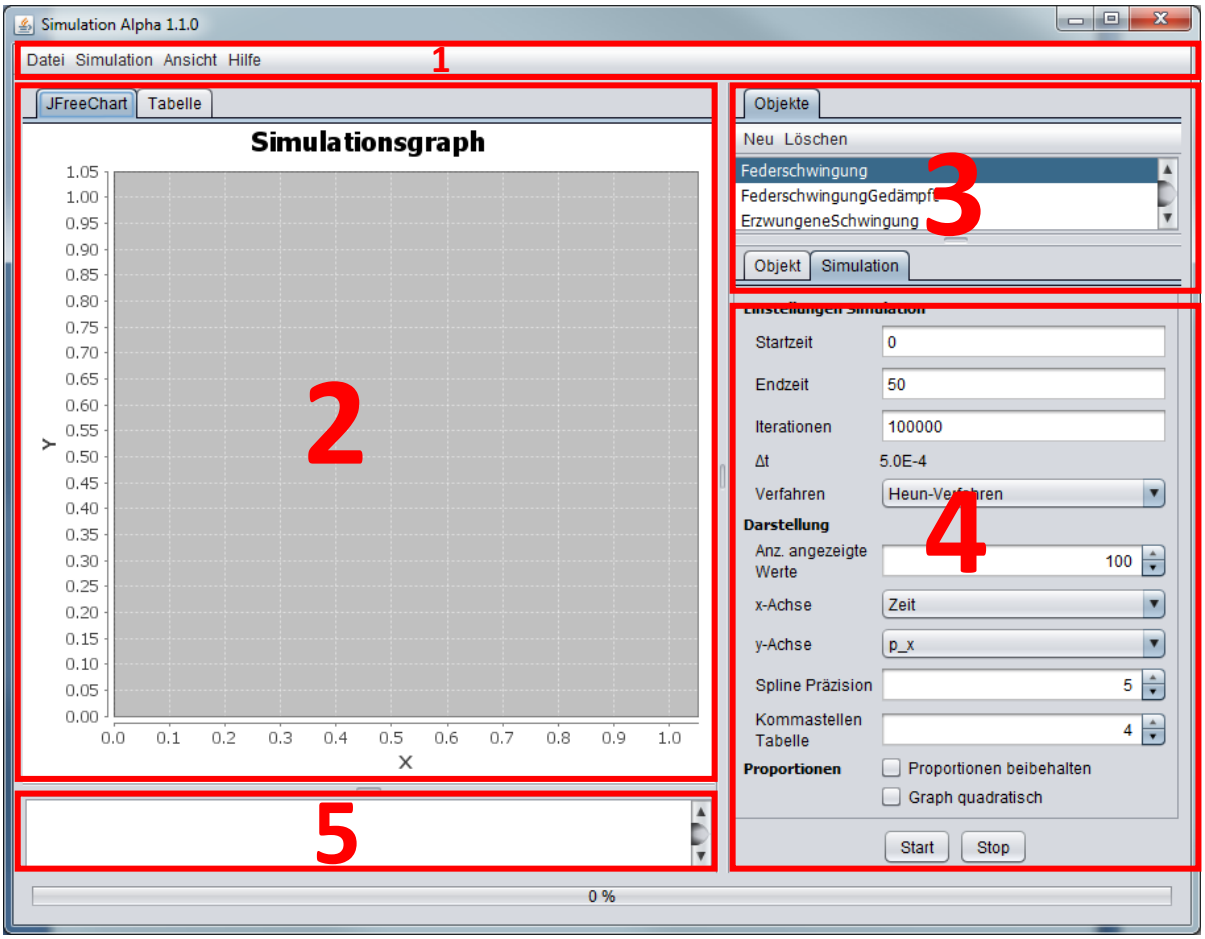
Download: https://www.dropbox.com/sh/5xh1wiebe3e3v2v/8mXYFyW7-C

2. Voraussetzungen

1. Windows® Vista (oder neuer)
2. Java 1.7.0 (oder neuer)

3. Simulation

3.1 Hauptfenster



1. **Menü** (siehe 3.2 Menü)
2. **Graph und Tabelle**: Stellt das Ergebnis graphisch dar und gibt die Werte in einer Tabelle aus. Mit Tab kann zwischen dem Graphen und der Tabelle gewechselt werden.
3. **Objektliste:** Enthält eine Liste aller Objekte in der Simulation.
4. **Einstellungen:** Einstellungen zur Simulation und zu den einzelnen Objekten.
5. **Infobox:** Ausgabe von Informationen und Fehlern.

3.2 Menu

Datei

Neu: Löscht den Graphen und alle Werte aus der Tabelle und setzt die Simulation auf die Grundeinstellungen zurück:

Startzeit	0
Endzeit	10
Iterationen	1'000'000
Verfahren	Heun Verfahren
Anz. angezeigte Werte	20
x-Achse	Zeit
y-Achse	p_x

Öffnen: Öffnen einer XML-Simulationsdatei, welche Einstellungen und Objekte beinhaltet.

Speichern: Speichern einer XML-Simulationsdatei, welche Einstellungen und Objekte beinhaltet.

Exportieren: Speichert die Tabelle im CSV Format, welches von den meisten Tabellenkalkulationsprogrammen gelesen werden kann.

Beispiele: Zeigt eine Liste mit anschaulichen Simulationsbeispielen, die mit einem Klick geöffnet werden können.

Schliessen: Beendet die Anwendung **ohne vorher zu speichern.**

Simulation

Start Starten der Simulation.

Stopp Stoppen der Simulation.

Ansicht

Infobox Ein- oder Ausblendung der Infobox.

Hilfe Zeigt Informationen zur Anwendung, u.a. ein Changelog.

3.3 Datenausgabe

Die Ausgabe der Simulationsergebnisse ist in zwei Bereiche eingeteilt, den Graphen und die Tabelle.

Graph

Der Graph stellt die Daten der ausgewählten Objekte graphisch dar. In den Simulationseinstellungen (siehe Einstellungen) kann die Art der Darstellung verändert werden.

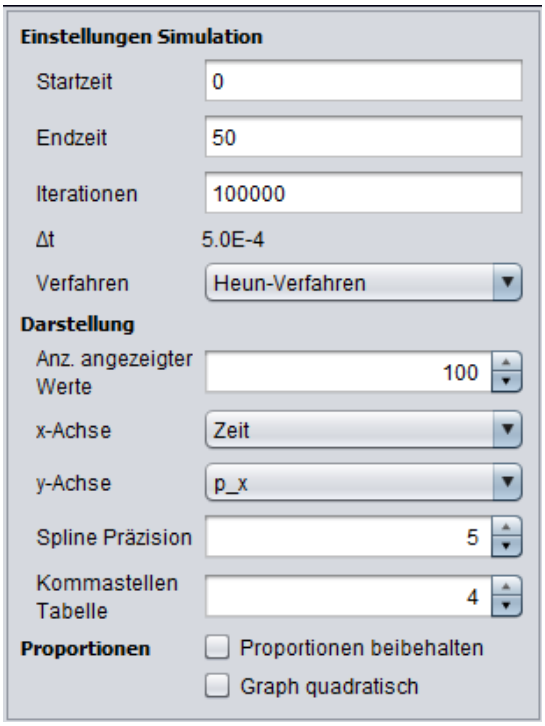
Tabelle

Die Tabelle stellt die Daten eines bestimmten Objektes dar. Dazu gehören Zeit, Beschleunigung, Geschwindigkeit und Position. In den Objekteinstellungen kann ausgewählt werden, ob dieses Objekt in der Tabelle angezeigt werden soll. Es kann jedoch immer nur ein Objekt angezeigt werden.

3.4 Einstellungen

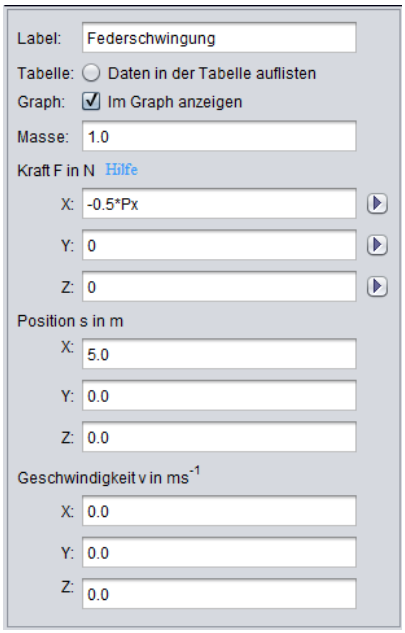
Wichtig: Nach jeder Werteänderung muss *Enter* gedrückt werden.

3.4.1 Simulation



Startzeit	Startpunkt der Simulation in Sekunden
Endzeit	Endzeit der Simulation in Sekunden
Iterationen	Anzahl Rechenschritte
Δt	Zeitspanne zwischen den einzelnen Berechnungen
Verfahren	Auswahl des Verfahrens (siehe Maturaarbeit, Kapitel Verfahren)
Anz. angezeigter Werte	Anzahl Punkte, die im Graph und der Tabelle angezeigt werden sollen. Realistisch sind 20-1000 Punkte
x-Achse	Auswahl der x-Achse
y-Achse	Auswahl der y-Achse
Spline Präzision	Präzision des Splines
Kommastallen Tabelle	Anzahl angezeigter Kommastellen in der Tabelle
Proportionen beibehalten	Gleicher Abschnitt auf beiden Achsen darstellen (wird erst nach dem Rechenvorgang ausgeführt)
Graph quadratisch	Graph quadratisch darstellen

3.4.2 Objekte



Label	Den eindeutigen Namen des Objektes. Erlaubte Zeichen: A-Z,a-z,0-9,Umlaute
Tabelle	Wenn dieses Feld aktiviert ist, wird nur dieses Objekt in der Tabelle angezeigt. Es kann immer nur ein Objekt angezeigt werden
Graph	Objekt im Graphen ein- bzw. ausblenden
Masse	Die Masse in kg
Kraft	Eine Funktion für die Kraft. Für eine erweiterte Hilfe mit der Maus über *Hilfe* fahren
Position	Anfangsposition des Objektes
Geschwindigkeit	Anfangsgeschwindigkeit des Objektes

3.5 Kraftfunktionen

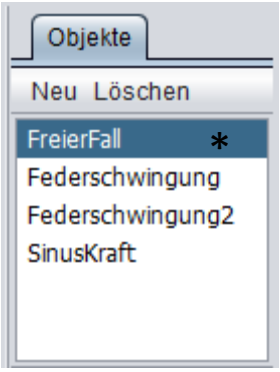
Konstanten	Schreibweise	Wert
Pi	PI	*3.141592653589793*
Eulersche Zahl	E	*2.718281828459045*
Eulers Konstante	Euler	*0.577215664901533*
Log von 2 zur Basis e	LN2	*0.693147180559945*
Log von 10 zur Basis e	LN10	*2.302585092994046*
Log von e zur Basis 2	LOG2E	*1.442695040888963*
Log von e zur Basis 10	LOG10E	*0.434294481903252*
Phi	PHI	*1.618033988749895*

Parameter	Schreibweise	Beispiel
Zeit	t	*t*30*
Masse	m	*t-m*
Position	Px,Py,Pz	*Px-log(Pi)*
Geschwindigkeit	Vx,Vy,Vz	*sin(Vz)*
Beschleunigung	Ax,Ay,Az	*sin(Kugel2Ax)*
Anderes Objekt	<Objektname><Parameter>	*FreierFallVx*cos(Kugel-t)*

Funktionen	Beschreibung	
abs	Betrag	*-cos(t)*abs(Px)*
acos	inverser Cosinus	*acos(Px/Py)*
asin	inverser Sinus	*asin(Px/Py)*
atan	inverser Tangens	*atan(Py/Px)*
cbrt	Dritte Wurzel des Argumentes	*cbrt(8+Vx)*
ceil	Kleinste ganze Zahl grösser oder gleich dem Argument	*ceil(sin(t))*

cos	Cosinus	*-cos(t)*abs(Px)*
cosh	Hyperbolischer Cosinus	*cosh(Px/Py)*
exp	Exponentialfunktion	*exp(-t)*
floor	Grösste ganze Zahl kleiner oder gleich dem Argument	*floor(t)*
log	natürlicher Logarithmus	*log(10*t+10)*
log10	dekadischer Logarithmus	*log10(Vx+10)*
max	Maximum der beiden Argumente	*max(KugelAx,5)*
min	Minimum der beiden Argumente	*min(0.5,sin(t))*
random	Zufallszahl zwischen 0 und 1	*10*random()*
round	Runden auf ganze Zahlen mit Tendenz zum Aufrunden (0.5 -> 1)	*round(Px)*
signum	Vorzeichen des Arguments	*-signum(Px)*sin(Px)*
sin	Sinus	*sin(Vx/Vy)*
sinh	Hyperbolischer Sinus	*sinh(t)*
sqrt	Wurzelfunktion	*sqrt(Px^2+Py^2)*
tan	Tangens	*tan(Px^2+Py^2)*
tanh	Hyperbolischer Tangens	*tanh(Px^2+Py^2)*
toDegrees	Konvertierung in Grad	*sin(toDegrees(Px))*
toRadians	Konvertierung in Bogenmass	*sin(to*Radians*(Px))*

3.6 Objektliste



Die Objektliste enthält eine Liste mit allen in der Simulation enthaltenen Objekten. Das aktive Objekt wird blau hinterlegt und in den Objekteinstellungen dargestellt.

Im Menüreiter kann das aktive Objekt gelöscht bzw. ein neues Objekt erstellt werden.

* aktives Objekt

3.7 Infobox

Die Infobox enthält Informationen und zeigt Fehler an.

4. Links

MyMechSim: https://www.dropbox.com/sh/5xh1wiebe3e3v2v/8mXYFyW7-C

Java: http://www.java.com/de/download/windows_xpi.jsp?locale=de

B. Deklaration

Ich erkläre hiermit,

- dass ich die vorliegende Arbeit selbstständig verfasst und nur die angegebenen Quellen benutzt habe,
- dass ich auf eine eventuelle Mithilfe Dritter in der Arbeit ausdrücklich hinweise,
- dass ich vorgängig die Schulleitung und die betreuende Lehrperson informiere, wenn ich
  - diese Maturaarbeit bzw. Teile oder Zusammenfassungen davon veröffentlichen werde
  - Kopien dieser Arbeit zur weiteren Verbreitung an Dritte aushändigen werde.

Ort / Datum / Unterschrift