Hochschule
Augsburg University of
Applied Sciences

**Programmieren 3**
**Wintersemester 2017/2018**
**Semester-Projekt**
**P. Rösch**

# Inhaltsverzeichnis


**1 Programmieren 3 – Überblick** **2**
- 1.1 Einordnung der Veranstaltung . . . 2
- 1.2 Didaktische Methodik . . . 2
- 1.3 Regeln und Hinweise . . . 3

**2 Werkzeuge und Programmiersprachen** **4**
- 2.1 Programmiersprache: Python . . . 4
- 2.2 Betriebssystem: Linux . . . 4
- 2.3 Dokumentation: Sphinx . . . 4
- 2.4 Software-Test: `doctest, unittest, nose` . . . 4
- 2.5 Entwicklungsumgebung: `spyder` . . . 5
- 2.6 Einheitlicher Programmierstil: `pylint` . . . 5
- 2.7 Installation der Software auf dem eigenen Rechner . . . 5

**3 Organisatorische Hilfsmittel** **6**
- 3.1 Wöchentliche Besprechung . . . 6
- 3.2 Definition von „Done“ . . . 6

**4 Mehrkörpersimulation – Aufgabenstellung** **7**

**5 Mehrkörpersimulation – Theoretische Grundlagen** **8**
- 5.1 Dynamik - Begriffe und Gleichungen . . . 8
- 5.2 Schrittweise Simulation . . . 8
- 5.3 Die Gravitationskraft . . . 8
- 5.4 Systeme aus vielen Körpern . . . 8

**6 Aufgaben, Teil 1 – 19.10.2016** **10**

**7 Aufgaben, Teil 2 – 09.11.2016** **12**

**8 Aufgaben, Teil 3 – 07.12.2016** **14**

**9 Aufgaben, Teil 4 – 21.12.2017** **15**

# 1 Programmieren 3 – Überblick

## 1.1 Einordnung der Veranstaltung

Nach der Erarbeitung wichtiger Grundlagen in den ersten beiden Semestern und vor der ersten Projektarbeit und dem Praxissemester ergeben sich folgende Anforderungen an die Lehrveranstaltung:

- Einordnung und praktische Anwendung der Inhalte aus den vergangenen Semestern aus Sicht der Software-Entwicklung.
- Vorbereitung von Projektarbeit und Praxissemester bezüglich der Software-Entwicklung.
- Konkrete Anwendung professioneller Methoden und Werkzeuge, die im Rahmen der Veranstaltung „Software Engineering 3“ aus Werkzeug-unabhängiger Sicht behandelt werden.

## 1.2 Didaktische Methodik

**Programmiersprache und Werkzeuge:** Im Rahmen der Vorlesung wird die Programmiersprache Python und für die Softwareentwicklung geeignete Werkzeuge vorgestellt. Studierende erhalten Jupyter-Notebooks für die interaktive Erarbeitung der Inhalte.

**Lösen einer komplexen Aufgabe im Team:** Die Entwicklung einer Mehrkörpersimulation stellt als „Semester-Projekt“ einen Rahmen für die Inhalte der Lehrveranstaltung dar.

**Einbeziehen vorgegebener Beispiele und theoretischer Grundlagen:** Um die selbständige Analyse von Quellen und den Transfer von Methoden aus Beispiel-Programme in eigene Applikationen zu fördern, werden physikalische und numerische Grundlagen in einem eigenen Kapitel zusammengefasst. Darüber hinaus werden undokumentierte, funktionierende Anwendungen zur Verfügung gestellt, in denen für das Projekt relevante Teilprobleme gelöst werden. Es ist eine wichtige und im Hinblick auf die spätere berufliche Tätigkeit sehr lohnende Aufgabe für die Studierenden, die Ressourcen zu identifizieren und zu nutzen, die für die erfolgreiche Lösung eines Teilproblems hilfreich sind.

**Einarbeitung in aktuelle Werkzeuge:** Um die Bearbeitung der Teilaufgaben zu unterstützen, wurden geeignete Werkzeuge ausgewählt, wobei auf deren freie Verfügbarkeit geachtet wurde. Die Werkzeuge werden in der Vorlesung präsentiert und dann unter Zuhilfenahme der Dokumentation im Praktikum verwendet, wobei ein Wiki zum internen Erfahrungsaustausch zur Verfügung steht.
Um die Installation auf dem eigenen Rechner zu vereinfachen, wird eine virtuelle Maschine (VirtualBox), die alle benötigten Werkzeuge umfasst, zur Verfügung gestellt.

## 1.3 Regeln und Hinweise

**Regeln:**

- Die Aufgaben sollen in Gruppen zu jeweils fünf Studierenden bearbeitet werden.
- Aufgaben können auch *vor* den angegebenen Terminen vorgeführt werden, eine verspätete Abgabe „kostet“ einen Joker (siehe Wiki).
- Die Lösungen können auf einem beliebigen Rechner erstellt werden, müssen jedoch auf einem der Rechner in M2.02 unter Linux vorgeführt werden.
- Ergebnisse werden akzeptiert (Individuelles Testat), falls **jedes Gruppenmitglied** folgende Dokumente vorlegen kann:
  1. Handschriftliche Aufzeichnungen, aus denen der Weg vom Problem zur Lösung sowie Funktionsweise der erstellten Software unabhängig vom Code nachvollziehbar erklärt werden kann. Alternativ kann auch ein eigenes Jupyter-Notebook für die prototypische Implementierung verwendet werden.
  2. Dokumentierter, lauffähiger Source-Code, der Zeile für Zeile erklärt werden kann und mindestens 8 Punkte bei der Überprüfung mit `pylint` erzielt (nur bei Programmier-Aufgaben).
- Für verteilte Anwendungen sollen die Rechner im Labor M2.02 verwendet werden.

**Hinweise:**

- Die Bearbeitung der Aufgaben ist ein wichtiger Teil der Klausur-Vorbereitung.
- Sämtliche Aufgaben sind mit (aus Sicht des Dozenten) „angemessenem“ Aufwand lösbar. Falls Sie sich an einer Teilaufgabe längere Zeit „festbeißen“, sollten Sie sich einen Tipp von Kommilitonen oder dem Dozenten (auch per EMail) holen.
- Vorlagen und Beispiel-Programme finden Sie im subversion-repository.
- Für die erfolgreiche Bearbeitung einer Aufgabe ist es hilfreich, die Aufgabenstellung vorher *genau* zu lesen. Im Rahmen der Vorlesung und über das wiki können (und sollen) Sie auch konkrete Fragen zu den Übungen stellen.
- Es kann sinnvoll sein, die Aufgaben für einen Abgabe-Termin nicht in der Reihenfolge zu bearbeiten, die der Nummerierung entspricht.

# 2 Werkzeuge und Programmiersprachen

In diesem Abschnitt wird motiviert, warum für diese Lehrveranstaltung bestimmte Werkzeuge und Programmiersprachen verwendet werden.

## 2.1 Programmiersprache: Python

Als Vertreter der objektorientierten Skriptsprachen verbindet Python (http://www.python.org) eine klare, elegante Syntax mit der Möglichkeit, leistungsfähige Erweiterungen einzubinden. C-Funktionen und -Bibliotheken können von Python aus z.B. mittels `ctypes` genutzt werden. Außerdem kann Python-Code mit `cython` (http://www.cython.org) automatisch in effizienten C-Code umgewandelt werden. Jupyter-Notebooks (http://www.jupyter.org) können für die interaktive Erstellung von Prototypen verwendet werden und erlauben es, Text, Formeln, Bilder und Python-Code zu kombinieren und auf einfache Art und Weise parallele und verteilte Anwendungen zu erstellen.

## 2.2 Betriebssystem: Linux

Linux ist frei verfügbar und erlaubt praktische Übungen zu allen Aspekten, die in dieser Lehrveranstaltung behandelt werden. Da ausschließlich freie Software verwendet wird, konnte ein Image für die Virtualisierungs-Plattform `VirtualBox` (http://www.virtualbox.org) erstellt und den Studierenden zur Verfügung gestellt werden. Damit ist es möglich, Linux als Gast auch auf einem Windows-Rechner zu nutzen, ohne die eigentliche Linux-Installation selbst durchführen zu müssen.

## 2.3 Dokumentation: Sphinx

Sphinx (http://sphinx-doc.org) erlaubt die automatische Erstellung einer verlinkten Software-Dokumentation aus Docstrings von Funktionen, Klassen, Modulen und Paketen. Es werden unterschiedliche Ausgabe-Formate wie html, pdf oder ePub unterstützt.

## 2.4 Software-Test: `doctest, unittest, nose`

Außer den bereits bekannten Unit-Tests unterstützt Python Tests über „ausführbare Kommentare“. Diese `doctests` können mit Unit-Tests kombiniert und automatisch ausgeführt werden. Über den Python-Interpreter sind interaktive Tests von Pyhon-Funktionen und Klassen möglich. Das Werkzeug `nose` durchsucht einen Verzeichnisbaum nach Tests und führt diese aus, siehe https://nose.readthedocs.org/en/latest.

## 2.5 Entwicklungsumgebung: `spyder`

`spyder` ist eine relativ schlanke, speziell auf Python abgestimmte, frei verfügbare Entwicklungsumgebung, siehe `http://pythonhosted.org/spyder`.

## 2.6 Einheitlicher Programmierstil: `pylint`

Es ist sehr sinnvoll, sich innerhalb eines Projekts für einen einheitlichen Programmierstil („coding conventions“) zu einigen. Das Werkzeug `pylint` erlaubt die automatische Überprüfung von Code. Als Konvention hat sich der Google Python Style Guide durchgesetzt.

## 2.7 Installation der Software auf dem eigenen Rechner

Falls Sie nicht nur in den Laboren im M-Gebäude, sondern auch auf dem eigenen Rechner arbeiten wollen, können Sie die virtuelle Debian-Maschine `pyBox` für die `VirtualBox` einsetzen. Dies bietet folgende Vorteile:

- Die benötigten Python-Bibliotheken sind installiert.
- Die Entwicklungsumgebung starten Sie durch Eingabe von `spyder&` in einem Terminal-Fenster oder über das entsprechende Symbol.
- Die Umgebung entspricht dem System, das Sie in den Laboren im M-Gebäude vorfinden.

# 3 Organisatorische Hilfsmittel

Es werden einige Methoden der agilen Softwareentwicklung eingeführt und verwendet.

## 3.1 Wöchentliche Besprechung

Die ersten Minuten des Praktikums werden für eine Bestandsaufnahme verwendet.

## 3.2 Definition von „Done“

Folgende Bedingungen müssen erfüllt sein, damit ein Task als erledigt („Done“) gilt:

1. Vollständige Dokumentation (`sphinx` bei Programmier-Tasks, ansonsten Text-Dokumente).
2. Für Programmier-Tasks wurden automatisierte Tests erstellt, andere Ergebnisse wurden auf Plausibilität überprüft.
3. Das Ergebnis wurde von zwei nicht in die Bearbeitung direkt involvierte Mitgliedern des Teams evaluiert (Review).
4. Python-Code erzielt einen `pylint`-Score von mindestens 8. Abweichungen davon sind zu begründen.

# 4 Mehrkörpersimulation – Aufgabenstellung

Im Laufe des Semesters soll die Simulation einer Menge von Körpern, die über die Gravitationskraft wechselwirken, entwickelt werden. Folgende Anforderungen sind vorgegeben:

1. Ablauf einer Simulation:

   (a) Vorab werden geeignete Programme auf den verfügbaren Rechnern gestartet, die eine parallelisierte, verteilte Berechnung erlauben.

   (b) Der Anwender spezifiziert über eine grafische Benutzerschnittstelle auf einem Desktop-Rechner die für die Verteilung notwendigen Daten, wie viele Körper simuliert werden sollen, in welchen Bereichen Massen und Radien der Körper liegen und welche Ausdehnung der Raumbereich hat, in dem die Körper verteilt werden sollen. Darüber hinaus werden die Masse des Körpers (oder des schwarzen Lochs) im Zentrum des Raumbereichs sowie die Schrittweite für die eigentliche Simulation festgelegt.

   (c) Die geforderte Anzahl von Körpern wird im angegebenen Raumbereich zufällig verteilt, wobei die ebenfalls zufällig gewählten Radien und Massen in den vom Benutzer gewählten Bereichen liegen. Die Geschwindigkeiten der Körper werden automatisch so initialisiert, dass eine relativ stabile Konfiguration entsteht.

   (d) Die zeitliche Entwicklung des Systems wird schrittweise simuliert, wobei alle verfügbaren Rechner (und Prozessor-Kerne) zu verwenden sind. Die Ergebnisse dürfen nicht von der Zahl der eingesetzten Prozessoren (Kerne) abhängen.

   (e) Innerhalb der grafischen Benutzerschnittstelle des Arbeitsplatzrechners (Client) wird der aktuelle Zustand des Systems grafisch dargestellt (3D-Animation) und der momentane Gesamtimpuls angezeigt.

   (f) Der Benutzer beendet die Simulation über ein geeignetes Bedienelement der grafischen Benutzerschnittstelle.

2. Die komplette Software ist zu dokumentieren, wobei als Formate `pdf` und `html` gefordert sind.

3. Für alle Teile der Software sind geeignete Tests zu erstellen, die möglichst automatisch ablaufen.

# 5 Mehrkörpersimulation – Theoretische Grundlagen

## 5.1 Dynamik - Begriffe und Gleichungen

**Position $\vec{r}(t)$:** Ort, an dem sich der Körper zum Zeitpunkt $t$ befindet.

**Geschwindigkeit $\vec{v}(t) = \dot{\vec{r}}(t)$:** Zeitliche Änderung der Position.

**Beschleunigung $\vec{a}(t) = \dot{\vec{v}}(t) = \ddot{\vec{r}}(t)$:** Zeitliche Änderung der Geschwindigkeit.

**Kraft $\vec{F}(t)$:** Nach Newton gilt $\vec{F} = m\,\vec{a}$, wobei $m$ die Masse des Körpers bezeichnet.

## 5.2 Schrittweise Simulation

Analog zur aus dem Mathematik bekannten Taylor-Entwicklung lässt sich die Position $\vec{r}$ eines Körpers zum Zeitpunkt $(t + \Delta t)$ näherungsweise berechnen, wenn die Position und deren zeitliche Ableitungen zum Zeitpunkt $t$ bekannt sind:

$$\vec{r}(t + \Delta t) \approx \vec{r}(t) + \Delta t\, \dot{\vec{r}}(t) + \frac{\Delta t^2}{2}\, \ddot{\vec{r}}(t) \tag{1}$$

Dabei bezeichnen Punkte zeitliche Ableitungen.

## 5.3 Die Gravitationskraft

Die Gravitationskraft, mit der eine Punktmasse $m_1$, die sich an der Position $\vec{r}_1$ befindet, von einer Punktmasse $m_2$, die sich an der Position $\vec{r}_2$ befindet, angezogen wird, ist

$$\vec{F}_{G,1} = G\frac{m_1 m_2}{|\vec{r}_2 - \vec{r}_1|^3}(\vec{r}_2 - \vec{r}_1), \quad G = 6.672 \cdot 10^{-11}\ \frac{\mathrm{Nm}^2}{\mathrm{kg}^2} \tag{2}$$

## 5.4 Systeme aus vielen Körpern

Folgende Größen sind für Beschreibung von Systemen aus $N$ Körpern relevant:

1. Gesamtmasse $M$ der beteiligten Körper:

$$M = \sum_{i=1}^{N} m_i \tag{3}$$

2. Position des Massen-Schwerpunkts $\vec{r}_s$:

$$\vec{r}_s = \frac{1}{M}\sum_{i=1}^{N} m_i \vec{r}_i \tag{4}$$

3. Position des Massen-Schwerpunkts ohne Berücksichtigung des Körpers $l$:

$$\vec{r}_{s,l} = \frac{1}{M - m_l} \sum_{i=1, i\neq l}^{N} m_i \vec{r}_i \tag{5}$$

4. Gesamt-Impuls $\vec{p}_{\text{gesamt}}$ des Systems:

$$\vec{p}_{\text{gesamt}} = \sum_{i=1}^{N} m_i \vec{v}_i \tag{6}$$

5. Der durch $(\vec{q}_{\min}, \vec{q}_{\max})$ spezifizierte kleinste Quader, der alle Körper enthält und entlang der Koordinaten-Achsen ausgerichtet ist.

$$\vec{q}_{\min} = \begin{pmatrix} \min_i(r_{i,x}) \\ \min_i(r_{i,y}) \\ \min_i(r_{i,z}) \end{pmatrix}, \ \vec{q}_{\max} = \begin{pmatrix} \max_i(r_{i,x}) \\ \max_i(r_{i,y}) \\ \max_i(r_{i,z}) \end{pmatrix}, \ \vec{r}_i = \begin{pmatrix} r_{i,x} \\ r_{i,y} \\ r_{i,z} \end{pmatrix} \tag{7}$$

Für eine stabile, näherungsweise kreisförmige Bewegung eines Körpers um eine Massenverteilung muss die Bahn durch die Gravitationskraft entsprechend gekrümmt werden. Daraus lässt sich folgende Gleichung für den Betrag der Geschwindigkeit der Punktmasse $i$ herleiten, siehe Wikipedia-Artikel

$$|\vec{v}_i| = \frac{M - m_i}{M} \sqrt{\frac{GM}{r}}, \quad r = |\vec{r}_i - \vec{r}_{s,i}| \tag{8}$$

Diese Gleichung gilt streng genommen nur, wenn der Abstand des Körpers zur Massenverteilung groß gegen deren Durchmesser ist.
Beobachtungen zeigen, dass Galaxien und Planetensysteme meist scheibenförmig sind, d.h. es gibt eine Raumrichtung, in der das Gebilde nur wenig ausgedehnt ist. Für die folgenden Betrachtungen sei dies die $z$-Richtung. Weiterhin wird beobachtet, dass sich die Objekte mit einem einheitlichen Drehsinn (z.B. aus der positiven $z$-Richtung betrachtet im Uhrzeigersinn) innerhalb der oben diskutierten Scheibe um das Zentrum, an der sich meist eine große Masse (Stern, schwarzes Loch) befindet, bewegen. Die Geschwindigkeiten stehen also senkrecht zur $z$-Achse und im Falle einer Kreisbahn auch senkrecht auf der Verbindungslinie zwischen Körper und Massenzentrum.
Daraus ergibt sich als mögliche Richtung für die Geschwindigkeit

$$\frac{\vec{v}_i}{|\vec{v}_i|} = \frac{(\vec{r}_i - \vec{r}_{s,i}) \times \vec{z}}{|(\vec{r}_i - \vec{r}_{s,i}) \times \vec{z}|}, \quad \vec{z} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \tag{9}$$

# 6 Aufgaben, Teil 1 – 19.10.2016

### A-1 Bewegung – Analyse der Aufgabenstellung

Der Dozent, der bezüglich der Gravitations-Simulation auch als Auftraggeber (Kunde) auftritt, hat die Aufgabenstellung im Kapitel 4 formuliert. Für die Studierenden, die die Rolle der Entwickler-Teams einnehmen ist diese Beschreibung Ausgangspunkt für ihre Arbeit.

(a) Auftraggeber: „Die Aufgabenstellung ist verständlich, eindeutig und vollständig. Als professionelles Entwickler-Team sollten Sie mich daher nicht mit unnötigen Rückfragen belästigen. Außerdem bezahle ich Sie ja für die Umsetzung meiner Wünsche."
Diskutieren Sie diese Aussage mit Ihren Kommilitonen und formulieren Sie gemeinsam eine Antwort an den Kunden (ca. vier Sätze), mit der Sie die Erfolgschancen Ihres Projekts deutlich vergrößern können.

(b) Leiten Sie aus der Aufgabenstellung (Abschnitt 4) konkrete Anforderungen für die zu entwickelnde Software ab und formulieren Sie das Ergebnis als Dokument, in dem jede *einzelne* Anforderung einen einprägsamen Titel und eine kurze Beschreibung (wenige Sätze) erhält.

(c) Bezüglich welcher Details weicht die Aufgabenstellung von typischen Kunden-Anfragen in der Realität ab?

(d) Erstellen Sie gemeinsam eine Liste offener Fragen, die mit dem Kunden (Dozenten) im Rahmen der Vorlesung geklärt werden müssen.
**Hinweis:** Es werden *nur* die Fragen beantwortet, die Sie auch stellen.

(e) Welches Risiko ergibt sich, wenn die vorhergehende Teilaufgabe nicht sorgfältig bearbeitet wird? (ca. drei Sätze)

### A-2 Bewegung – Physikalische Beschreibung

Als Ausgangspunkt für eigene Nachforschungen wurden grundlegende Begriffe in Abschnitt 5.1 zusammengefasst.

(a) Erklären Sie mit eigenen Worten die Bedeutung der definierten physikalischen Größen.

(b) Weshalb befinden sich über den Symbolen Vektor-Pfeile?

(c) Erstellen Sie eine Skizze eines Läufers (Strichmännchen), der sich durch ein hügeliges Gelände bewegt und ergänzen Sie die Zeichnung durch eine geeignete Repräsentation der genannten physikalischen Größen.

### A-3 Bewegung – Gravitationskraft

Als spezielles physikalisches System, das simuliert werden soll, wurde eine Menge von Körpern gewählt, die über Gravitation wechselwirken.

(a) Beschaffen Sie sich Daten über unser Sonnensystem, insbesondere die Massen und Radien der beteiligten Körper sowie die Bahngeschwindigkeiten und Abstände der Planeten zur Sonne. (In SI-Einheiten, also kg, m, s und Kombinationen daraus)

(b) Diskutieren Sie Formel für die Gravitationskraft mit Ihren Kommilitonen und klären Sie insbesondere die Bedeutung des Begriffs „Punktmasse“.

(c) Skizzieren Sie die Bahn der Erde um die Sonne und zeichnen Sie analog zur vorhergehenden Aufgabe Position, Geschwindigkeit und Beschleunigung ein.

(d) Begründen Sie, warum die Erde sich um die Sonne bewegt und nicht davonfliegt oder in die Sonne stürzt.

### A-4 Bewegung – Schrittweise Simulation

Für mehr als zwei Körper können im Allgemeinen keine geschlossenen Formeln für die durch Gravitation beeinflusste Bewegung angegeben werden.

(a) Beschreiben Sie anhand Ihrer Skizze aus Aufgabe 3c anschaulich, wie eine schrittweise Simulation der Planetenbewegung möglich ist.

(b) Passen Sie die vorgegebene Formel (1) an das vorliegende Problem an.

(c) Wie kann man mit Hilfe dieser Formel den Funktionswert $\vec{r}(t_2)$ aus $\vec{r}(t)$ bestimmen, wenn $t_2$ deutlich größer als $t$ ist?[1]

(d) Visualisieren Sie den Ablauf einer Simulation der Bewegung von $N$ Körpern, die über Gravitation wechselwirken, durch ein Ablauf-Diagramm, aus dem die einzelnen Schritte und die dazugehörige Formeln so hervorgehen, dass das Verfahren konkret implementiert werden kann.
**Hinweis:** Diese Aufgabe bezieht sich nur auf die eigentliche numerische Berechnung, nicht jedoch auf die Interaktion mit dem Benutzer, die Verteilung auf mehrere Rechner etc.

(e) Warum ergibt diese Rechnung nur Näherungen an die exakten Positionen und Geschwindigkeiten?

(f) Begründen Sie, warum man $\Delta t$ nicht zu groß wählen darf.

[1] Stichwort: Iteratives Verfahren

# 7 Aufgaben, Teil 2 – 09.11.2016

### A-5 Grob-Entwurf

Eine der Komplexität der Aufgabe angemessene Architektur stellt die Grundlage für die nächsten Entwicklungsschritte dar. Am Anfang eines größeren Projekts ist es nicht unüblich, dass noch nicht alle Technologien (z.B. Parallelisierung, Verteilung) bekannt sind. Dies erfordert abstrakt formulierte Entwürfe.

**Hinweis:** Für die später geforderte Optimierung, Parallelisierung und Verteilung ist es hilfreich, wenn die Positionen **aller** Himmelskörper in **einer** Liste gespeichert sind. Gleiches gilt für die Geschwindigkeiten.

(a) Erarbeiten Sie mit Ihrem Team mehrere alternative Ansätze (mindestens vier) zur Umsetzung der Aufgabe und fertigen Sie Skizzen an, die die Architektur visualisieren. (Papier/Bleistift, jew. eine A4-Seite)

(b) Diskutieren Sie jeden Vorschlag und erstellen Sie jeweils eine Liste mit Vor- und Nachteilen und eine Aufstellung von Voraussetzungen, die für eine erfolgreiche Umsetzung jedes Vorschlags erfüllt sein müssen (Auf Papier oder elektronisch).

(c) Entscheiden Sie sich für einen Ansatz und verfeinern Sie ihn so lange, bis Sie von der Machbarkeit überzeugt sind.

(d) Warum kann es sinnvoll sein, mehrere Ansätze parallel zu verfolgen?

(e) Sammeln Sie offene Fragen, die mit dem Kunden oder mit Unterstützung externer Berater beantwortet werden müssen.

### A-6 Testfälle

Auf systematisches Testen kann in keinem Software-Projekt verzichtet werden.

(a) Sammeln Sie Gründe, die dafür bzw. dagegen sprechen, sich zu einem möglichst frühen Zeitpunkt im Projekt-Ablauf mit der Test-Problematik auseinander zu setzen. (Tabelle)

(b) Erstellen Sie ein Liste mit Testfällen, die sich nur auf das physikalische Modell, nicht jedoch auf die konkrete Implementierung beziehen.
Beispiel: Die Masse eines Körpers muss positiv sein.

(c) Welche Beziehung sollte zwischen den in Aufgabe A-1b erarbeiteten Anforderungen und den Testfällen bestehen?

(d) Formulieren Sie weitere Testfälle, die sich auf das zu implementierende Gesamtsystem beziehen.

### A-7 Implementierung und Test der Berechnungen

Im ersten Schritt soll die Berechnung der Bahnkurven von Punktmassen im Gravitationsfeld implementiert und überprüft werden.

(a) Setzen Sie Ihren Entwurf aus A-4d in Python-Software um, die ausgehend von einem initialisierten System die Positionen der Körper in Abhängigkeit von der Zeit berechnet und in geeigneter Form speichert.

(b) Überlegen Sie sich realistische Testfälle und stellen Sie die Bahnkurven grafisch dar.

(c) Begründen Sie schriftlich, warum die von Ihnen durchgeführten Tests Aussagen über die Qualität Ihrer Implementierung zulassen.

### A-8 System Sonne-Erde mit GUI

Ein in der Vorlesung diskutiertes Beispiel demonstriert, wie die Ausgabe von `OpenGL` mit `Qt` und Python kombiniert werden kann.

(a) Identifizieren Sie die Programmteile, die Sie für eine interaktive Visualisierung Bahnkurve aus der letzten Teilaufgabe benötigen.

(b) Modifizieren Sie das Programm zunächst so, dass eine Kreisbahn (ohne Gravitations-Berechnung) ausgegeben wird.

(c) Nutzen Sie die in A-7 entstandene Software, um die Bahn der Erde um die Sonne grafisch auszugeben.

(d) Ergänzen Sie die GUI um Bedienelemente, die es Ihnen erlauben, ausgewählte Anfangsbedingungen der Simulation zu modifizieren und das Ergebnis der Simulation anzuzeigen.

# 8 Aufgaben, Teil 3 – 07.12.2016

### A-9 Initialisierung eines Systems aus Punktmassen

Für die Simulation unseres Sonnensystems konnten Sie die Geschwindigkeiten der Körper, die zu den bekannten Bahnen führen, aus Tabellen ablesen. Um die Geschwindigkeiten zufällig um ein schweres Zentrum (z.B. ein schwarzes Loch) verteilter Massen zu finden, die zu einer recht stabilen Konfiguration führt, müssen Näherungen durchgeführt werden.

(a) Erweitern Sie Ihre Simulation so, dass $N$ Körper zufällig, aber recht gleichmäßig in einem Raumbereich (Formel 7 auf Seite 9) verteilt werden. Die Massen der Körper werden ebenfalls automatisch aus einem Intervall $[m_{\min}, m_{\max}]$ gewählt. Es soll außerdem möglich sein, eine (grosse) Masse im Zentrum des Raumbereichs zu positionieren.

(b) Implementieren Sie die Initialisierung der Geschwindigkeiten (Formeln 8 und 9 auf Seite 9) für ein System aus $N$ Körpern.

(c) Finden Sie heraus, ob diese Formeln auch für kleine Systeme ($N = 2, 3$) gelten und welche Bahnkurven sich für unterschiedliche Massen-Verhältnisse ergeben.

### A-10 Grafische Benutzeroberfläche

Die Parameter der Simulation sollen über eine grafische Benutzerschnittstelle eingegeben werden können. Außerdem soll eine animierte grafische Ausgabe erfolgen, so dass die Bewegungen der Körper direkt am Bildschirm beobachtet werden können.

(a) Entwerfen Sie eine grafische Benutzersschnittstelle, so dass die Anforderungen (Seite 7) erfüllt sind.

(b) Implementieren und testen Sie die entstandene Schnittstelle mit einem System aus mindestens 10 Körpern, wobei sich in der Mitte ein typisches schwarzes Loch befinden soll.

### A-11 Profiling und Optimierung

Ein entscheidender Schritt bei der Optimierung einer Anwendung bezüglich der Rechenzeit ist das „Profiling“.

(a) Führen Sie eine Analyse Ihrer Simulations-Anwendung durch und erstellen Sie eine Liste mit Methoden oder Funktionen, von deren Optimierung Sie sich einen merklichen Effekt bezüglich der Laufzeit versprechen.

(b) Optimieren Sie Ihr Programm an einer entscheidenden Stelle und vergleichen Sie die Auswirkungen mit Ihren Erwartungen.

**Hinweis:** Die geschickte Verwendung von `numpy` ist eine bewährte Methode, um Berechnungen zu beschleunigen.

# 9 Aufgaben, Teil 4 – 21.12.2017

### A-12 Optimierung mit `cython`

In vielen Fällen sind die Möglichkeiten begrenzt, die Ausführung aufwändiger Programmteile innerhalb einer interpretierten Sprache wie Python zu beschleunigen. Das Paket `cython`[2] bietet durch die weitgehend automatische Erzeugung von C-Code aus Python eine elegante Möglichkeit zur Optimierung rechenintensiver Funktionen.

(a) Optimieren Sie Ihre Simulation durch den Einsatz `cython`.

(b) Messen und dokumentieren Sie die erreichte Beschleunigung.

### A-13 Simulation – Parallelisierung und Verteilung

Eine Verteilung der Rechenarbeit auf mehrere Prozessoren und Rechner kann den Ablauf von Simulationen deutlich beschleunigen.

(a) Begründen Sie, warum sich das vorliegende Problem für die Parallelisierung und Verteilung eignet. (ca. vier Sätze)

(b) Wie können Sie sicherstellen, dass sich die Ergebnisse der Simulation durch die Parallelisierung nicht ändern.

### A-14 Simulation – Detaillierte Planung und Implementierung

Planen Sie die verteilte Simulation im Detail und führen Sie eine Implementierung inklusive Test und Dokumentation im Team durch. Der Fortschritt und auftretende Probleme sollen dabei schriftlich festgehalten werden.
Ziel ist dokumentierte Software, die Sie an den „Kunden“ ausliefern können.

### A-15 Fazit (freiwillig)

Fassen Sie die Kenntnisse und Fähigkeiten, die Sie sich im Rahmen dieser Veranstaltung erarbeitet haben, in einer Tabelle zusammen und stellen Sie Beziehungen zu anderen Lehrveranstaltungen her.

[2] www.cython.org