Programmieren für Physikerinnen und Physiker, Übung 1

PD Dr. Hendrik Weimer, WS 2017/18, 25.10.2019

1. Python in virtualisiertem GNU/Linux

(a) Installieren Sie eine Virtualisierungsumgebung (z.B. VirtualBox platform package, `https://www.virtualbox.org/wiki/Downloads`) auf Ihrem Computer.

(b) Laden Sie eine für die Virtualisierung vorgefertigte GNU/Linux-Distribution wie Ubuntu 19.04 (z.B. von `https://www.osboxes.org/ubuntu/`, Anleitung unter `https://www.osboxes.org/guide/`) herunter und laden Sie die Distribution in die Virtualisierungsumgebung.

(c) Starten Sie über ein Terminal den Python3-Interpreter und lassen Sie den Text "Hallo, Welt!" ausgeben.

2. Dokumentation mit Docstrings

Neben Kommentaren unterstützt Python auch die Möglichkeit, Code mit Docstrings zu dokumentieren. Docstrings werden mit drei doppelten Anführungszeichen (`"""`) eingeleitet und geschlossen. Die gängigen Konventionen für Docstrings sind im Dokument PEP 257 (`https://www.python.org/dev/peps/pep-0257/`) spezifiziert.

Schreiben Sie einen Docstring für die Funktion `polynomial()` aus der Vorlesung, der konform zu PEP 257 ist.

3. Numerisches Ableiten

(a) Schreiben Sie ein Programm, das die Ableitung der Funktion $f(x) = 1/(x^2+1)$ berechnet. Definieren Sie dazu eine entsprechende Python-Funktion und berechnen Sie die Ableitung an der Stelle $x_0 = 1$. Vergleichen Sie das Ergebnis für verschiedene Schrittweiten mit dem exakten Resultat $f'(x_0) = -1/2$. Wählen Sie die Schrittweiten dabei so, dass sich die Annäherung an das exakte Ergebnis nachvollziehen lässt.

(b) Eine genauerere Diskretisierung der Ableitung liefert die Formel

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h}. \tag{1}$$

Wiederholen Sie die Analyse aus (a), indem Sie für beide Berechnungsarten eigene Funktionen verwenden.

(c) Die zweite Ableitung lässt sich näherungsweise gemäß

$$f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} \tag{2}$$

berechnen. Berechnen Sie die zweite Ableitung analog ($f''(x_0) = 1/2$).

4. Binäre Operationen

Neben der Exklusives-Oder-Operation `^` existieren in Python eine Reihe weiterer Operationen, die auf binärer Ebene arbeiten. Dies sind: binäres Verschieben nach links (`<<`) bzw. rechts (`>>`), sowie binäres Und (`&`) sowie binäres Oder (`|`).

(a) Berechnen Sie für jede dieser Operationen $\circ$ den Wert $9 \circ 3$. Erklären Sie die Ergebnisse, indem Sie für beide Eingabewerte und das Ergebnis die binäre Darstellung mittels der Funktion `bin()` betrachten.

(b*) Das binäre Komplement (`~`) einer Ganzzahl ersetzt jede 0 in der Binärdarstellung durch eine 1 und umgekehrt. Es ist gegeben durch $\tilde{x} = -x-1$, das heisst negative Zahlen werden in Binärdarstellung durch führende Einsen anstatt führender Nullen gekennzeichnet. Zeigen Sie, dass für jede Zweierpotenz k die Anweisung $x\& - k$ die Zahl x auf ein Vielfaches dieser Zweierpotenz abrundet. Zeigen Sie zudem, dass Aufrunden auf das nächste Vielfache einer Zweierpotenz durch die Anweisung $(x + k - 1)\& - k$ erfolgen kann.

*: Zusatzaufgabe

Programmieren für Physikerinnen und Physiker, Übung 2

PD Dr. Hendrik Weimer, WS 2017/18, 01.11.2019

1. Darstellung von Fließkommazahlen

In Python werden Fließkommazahlen des `float`-Datentyps als Binärzahl mit 64 Stellen (d.h. 64 Bit) gespeichert. Die Zahlen werden in der Darstellung $(-1)^s \times m \times 2^{e-1023}$ gespeichert, wobei s das Vorzeichen, m die Mantisse und e der Exponent der Zahl ist. Für die Mantisse stehen 52 Bit, für den Exponenten 11 Bit zur Verfügung, was zusammen mit einem Bit für das Vorzeichen insgesamt 64 Bit Speicherbedarf ergibt. Die Mantisse erfüllt dabei $1 \leq m \leq 2 - 2^{-52}$. Diese Darstellung ist in den meisten Programmiersprachen gebräuchlich, da sie dem technischen Standard IEEE 754 entspricht.

(a) Der Exponent e kann den Maximalwert $e = 2^{11} - 2 = 2046$ annehmen, da der Wert für die größte 11-Bit-Zahl $e = 2^{11} - 1 = 2047$ in Python unendliche Zahlen repräsentiert. Wie lautet die größtmögliche darstellbare Zahl? Überprüfen Sie das Ergebnis mit einem Python-Skript.

(b) Ist der Exponent $e = 0$, dann wird die Zahl als subnormale Zahl interpretiert, welche die Darstellung $(-1)^s \times m \times 2^{-1022}$ hat, wobei hierbei die Mantisse zwischen 0 und $1 - 2^{-52}$ liegt. Wie lautet also die kleinste positive Zahl, die in Python dargestellt werden kann? Überprüfen Sie wiederum Ihr Ergebnis mit einem Skript.

2. Lambda-Funktionen

In Python können Funktionen nicht nur mit der `def`-Anweisung definiert werden sondern mithilfe der Lambda-Anweisung einer Variable zugewiesen werden. Beispielsweise erzeugt

```
>>> f = lambda x: x+1
```

eine Funktion, die einen Parameter `x` erwartet und als Rückgabewert `x+1` liefert. Eine solche Lambda-Funktion kann wie gewohnt durch `f(x)` aufgerufen werden.

Schreiben Sie das Programm `ableitung.py` aus der Vorlesung 2 so um, dass es für `quadratic` und `polynomial` Lambda-Funktionen verwendet. Erstellen Sie hierzu zwei Funktionen `make_quadratic(a=1, b=0, c=0)` und `make_polynomial(n, a=1)`, die als Rückgabewert nicht eine Zahl sondern eine entsprechende *Funktion* mit einem einzelnen Parameter zurückgeben.

3. Numerische Interpolation

Durch Interpolation können Schätzungen für Zwischenwerte einer Funktion ermittelt werden, wenn die Funktion an anderen Punkten bekannt ist.

(a) Die einfachste Art der Interpolation ist eine lineare Interpolation, bei der zwischen zwei bekannte Funktionswerte eine Gerade gelegt wird. Schreiben Sie eine Funktion, welche den Funktionswert dieser Gerade zurückgibt. Ermitteln Sie für eine unbekannte Funktion mit $f(1) = 0.5$ und $f(2) = 0.2$ den Interpolationswert an der Stelle $x = 1.5$.

(b) Sind mehr als zwei Datenpunkte der Funktion bekannt, können auch Interpolationen höherer Ordnung verwendet werden. Bei drei Funktionswerten $f(x_0)$, $f(x_1)$ und $f(x_2)$ lässt sich ein quadratisches Interpolationspolynom $p(x)$ gemäß

$$p(x) = \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} f(x_0) + \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)} f(x_1) + \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)} f(x_2) \tag{1}$$

angeben. Berechnen Sie mit dem zusätzlichen Wert $f(0) = 1$ die quadratische Interpolation wiederum an der Stelle $x = 1.5$.

(c) Nehmen Sie nun an, dass die unbekannte Funktion gegeben ist durch $f(x) = 1/(x^2+1)$ und vergleichen Sie die Abweichungen der beiden Interpolationsmethoden.

(d) Im Gegensatz zur Interpolation versucht die Extrapolation Funktionswerte außerhalb des Intervalls der bekannten Werte zu ermitteln, wobei die Berechnungsvorschrift dieselbe bleibt. Berechnen Sie die Extrapolation für $x = 10$ und betrachten Sie wiederum die Abweichungen vom tatsächlichen Funktionswert. Was fällt Ihnen auf?

4. Boolesche Algebra

In der Booleschen Algebra wird anstelle von Zahlen mit den Wahrheitswerten `True` und `False` und den logischen Operationen `not`, `and` und `or` gerechnet. Dabei gelten bekannte Regeln wie das Kommutativgesetz `a or b = b or a` oder das Assoziativgesetz `(a or b) or c = a or (b or c)` (bzw. für `and` analog).

(a) Beweisen Sie die Gültigkeit der de Morganschen Gesetze `not (a or b) = (not a) and (not b)` und `not (a and b) = (not a) or (not b)`.

(b) Vereinfachen Sie den Ausdruck `(a or b) and (not(a) or b)`. Hinweis: Verwenden Sie dazu das Distributivgesetz `x or (y and z) = (x or y) and (x or z)`.

Programmieren für Physikerinnen und Physiker, Übung 3

PD Dr. Hendrik Weimer, WS 2017/18, 08.11.2019

1. Programmablaufplan

Erstellen Sie einen Programmablaufplan für die Funktion `fib(n)` aus der Vorlesung 3.

2. Numerische Integration

In der Vorlesung 3 wurde gezeigt, dass sich das Integral $\int_a^b f(x)dx$ durch die (linke) Riemann-Summe $\sum_{i=0}^{n-1} f(x+i\delta x)\delta x$ näherungsweise berechnen lässt.

(a) Ersetzen Sie die Berechnungsvorschrift durch die rechte Riemann-Summe, d.h., der Funktionswert wird nicht am Anfang sondern am Ende des jeweiligen Intervalls δx ausgewertet. Berechnen Sie das Integral $\int_0^1 x^2 dx$ für verschiedene Anzahl der Schritte n.

(b) Modifizieren Sie Ihr Programm, sodass die Funktion am Mittelpunkt des Intervalls ausgewertet wird (Mittelpunktsregel). Wie wirkt sich das auf die Genauigkeit des Ergebnisses aus?

(c) Wiederholen Sie die vorangengegangen Berechnungen für das Integral $\int_0^1 1/(1+x^3)dx$.

3. Berechnung von Reihen

Schreiben Sie Funktionen zur Berechnung des n-ten Werts der folgenden Reihen. Berechnen Sie anschließend den asymptotischen Wert für $n \to \infty$, indem Sie eine vorher festgelegte Genauigkeit (z.B. 10^{-6}) erreichen.

(a) Geometrische Reihe: $\sum_{n=0}^{\infty} x^n$ für $x = 0.5$ und $x = 0.99$.

(b) Riemannsche Zeta-Funktion: $\zeta(x) = \sum_{n=1}^{\infty} n^{-x}$ für $x = 3$ und $x = 6$.

(c) Exponentialfunktion: $\exp(x) = \sum_{n=0} x^k/k!$ für $x = -1$ und $x = 1$. Schreiben Sie zur Berechnung der Fakultät eine entsprechende Funktion.

4. Newton-Verfahren

Das Newton-Verfahren ermöglicht die Suche nach Nullstellen beliebiger Gleichungen $f(x) = 0$, insbesondere solcher die sich nicht analytisch lösen lassen. Es handelt sich dabei um ein iteratives Verfahren, d.h., die Lösung wird schrittweise angenähert. Die Iterationsvorschrift für das Newton-Verfahren lautet

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}. \tag{1}$$

Der Startwert x_0 ist dabei grundsätzlich beliebig, allerdings kann es passieren, dass das Verfahren für bestimmte Startwerte nicht konvergiert. In diesem Fall muss man einen anderen Startwert wählen.

(a) Schreiben Sie ein Funktion zur Berechnung der Quadratwurzel $\sqrt{y}$, indem Sie die Gleichung $f(x) = x^2 - y = 0$ mittels Newton-Verfahren lösen. Verwenden Sie für die Ableitung das analytische Ergebnis $f'(x) = 2x$. Brechen Sie die Iteration ab, sobald eine vorgegebene Genauigkeit erreicht ist.

(b) Ändern Sie die Funktion ab, dass die Ableitung nicht mehr analytisch sondern numerisch berechnet wird. Wie wirkt sich das auf die Genauigkeit des Verfahrens aus?

5. Zustandsmaschine

Zur Steuerung von Programmen kann oft eine Zustandsmaschine verwendet werden. Dazu wird eine bestimmte Anzahl von möglichen Zuständen festgelegt sowie die Ereignisse, unter denen die Zustände ineinander Übergehen. Der aktuelle Zustand wird dabei in einer Integer-Variable gespeichert.

Als Beispiel hierfür soll ein Getränkeautomat für Kaffee betrachtet werden, wobei ein Kaffee 25 Cent kostet. Der Automat akzeptiert nur Münzen zu 20, 10 und 5 Cent und gibt kein Restgeld. Solange der Automat nicht die erforderliche Summe erhalten hat, soll der aktuelle Betrag angezeigt werden. Sobald genug Geld eingeworden wurde, zeigt der Automat zunächst die Ausgabe "Koche Kaffee ..." an und beginnt darauf wieder bei 0 Cent. Nehmen Sie hierbei an, dass das Kochen des Kaffees instantan erfolgt und mit Ausgabe der Nachricht erledigt ist.

(a) Wie viele Zustände sind erforderlich, um den Automaten zu realisieren? Beachten Sie hierbei, dass der Automat nur erkennen kann, welche Münze zuletzt eingeworfen wurde und außer der Zustandsvariable über keinen internen Speicher verfügt.

(b) Schreiben Sie eine Funktion, die den Kaffeeautomaten implementiert. Die Funktion erhält zwei Parameter: Den aktuellen Zustand und die zuletzt eingeworfene Münze. Der Rückgabewert ist der neue Wert der Zustandvariable.

(c) Schreiben Sie ein Skript, dass den Automaten mit folgenden Münzen füttert: 5 Cent, 20 Cent, 10 Cent, 10 Cent, 10 Cent, 50 Cent, 5 Cent, 5 Cent, 20 Cent.

Programmieren für Physikerinnen und Physiker, Übung 4

PD Dr. Hendrik Weimer, WS 2019/20, 15.11.2019

1. Das Modul `time`

(a) Schreiben Sie ein Skript, welches das Modul `time` importiert. Mit der Funktion `time` können Sie auf die aktuelle Systemzeit zugreifen. Überprüfen Sie die Funktionsweise, indem Sie mit der Funktion `sleep` verschiedene Zeiten (in Sekunden) vergehen lassen.

(b) Messen Sie die Dauer der Integration des Skripts `integration.py` aus Vorlesung 3 für verschiedene Werte der gewünschten Genauigkeit. Vergleichen Sie mit der Funktion `scipy.integrate.quad` aus SciPy.

(c) Die von der Funktion `time` zurückgegebene Zeit kann mit der Funktion `localtime` in ein `struct_time`-Objekt umgewandelt werden, welches wiederum mit der Funktion `asctime` in eine lesbare Form gebracht werden kann. Wo liegt der Nullpunkt der `time`-Funktion?

2. Installation eines externen Moduls

(a) Installieren Sie das Modul `dateutil` in Ihrer Python-Umgebung.

(b) Schreiben Sie eine Funktion, die die Anzahl der vollen Monate zwischen zwei Datumsangaben berechnet, die jeweils als `struct_time`-Objekt übergeben werden sollen. Verwenden Sie hierzu die Funktion `relativedelta` aus `dateutil`.

(c) Berechnen Sie die Anzahl der Monate zwischen dem 31.05.2015 12:00:00 und dem 31.05.2017 11:59:59. Vergleichen Sie mit der Anzahl der Sekunden, die Sie über die Differenz von zwei Aufrufen der Funktion `mktime` aus dem `time`-Modul erhalten können. Ist das Ergebnis korrekt?

3. Entwicklung eigener Module

(a) Schreiben Sie ein Skript, das die ersten 100 Zahlen der Fibonacci-Folge berechnet. Speichern Sie hierzu die Datei `fib.py` aus der Vorlesung 3 im selben Verzeichnis wie Ihr Skript ab und binden Sie das Skript aus der Vorlesung als Modul `fib` ein.

(b) Schreiben Sie ein neues Skript, das die Ableitung der Funktion $f(x) = \sin(x)$ an der Stelle $x = \pi/3$ berechnet. Greifen Sie hierfür auf die Datei `ableitung.py` aus Vorlesung 2 zurück und binden Sie diese als Modul ein, um auf die Funktion `derivative` zurückgreifen zu können.

4. Monte-Carlo-Integration

(a) Schreiben Sie ein Skript, welches die Zahl π über das Integral $4\int_0^1 \sqrt{1-x^2}dx$ numerisch berechnet. Vergleichen Sie die Anzahl der für eine bestimmte Genauigkeit benötigten Funktionsaufrufe mit der in der Vorlesung 4 im Skript `pi.py` vorgestellten Methode.

(b) Erweitern Sie die auf Zufallszahlen basierende Methode, um das Volumen der d-dimensionalen Einheitskugel zu berechnen, welche durch die Ungleichung

$$\sum_{i=1}^{d} x_i^2 < 1 \tag{1}$$

gegeben ist. Bestimmen Sie wiederum das Konvergenzverhalten hin zum exakten Ergebnis $V_d = \pi^{d/2}/\Gamma(d/2+1)$.

(c*) Berechnen Sie nun das Volumen der d-dimensionalen Einheitskugel als mehrdimensionales Integral, welches gegeben ist durch

$$V_d = 2^d \int_0^1 dx_1 \int_0^{\sqrt{1-x_1^2}} dx_2 \int_0^{\sqrt{1-x_1^2-x_2^2}} dx_3 \cdots \int_0^{\sqrt{1-\sum_{i=1}^{d-2} x_i^2}} \sqrt{1-\sum_{i=1}^{d-1} x_i^2}\, dx_{d-1}. \tag{2}$$

Welche der beiden Methoden konvergiert für große Werte von d mit weniger Funktionsaufrufen an das exakte Ergebnis?

*: Zusatzaufgabe

Programmieren für Physikerinnen und Physiker, Übung 5

PD Dr. Hendrik Weimer, WS 2019/20, 22.11.2019

1. Regel 110

Als elementare zellulären Automaten beschreibt man ein dynamische ein-dimensionale Systeme, bei denen der binäre Zustand einer Zelle zur Zeit $t+1$ nur von dem Zustand der Zelle und seinen Nachbarn zur Zeit t abhängt. Ein besonders interessanter zelluärer Automat ist die *Regel 110*, bei der die Zellen jeweils in Dreiergruppen betrachtet werden und mit jedem ganzzahligen Zeitschritt nach folgender Vorschrift aktualisiert werden:

111	110	101	100	011	010	001	000
0	1	1	0	1	1	1	0

Die erste Zeile gibt dabei den Zustand eines Dreierblocks zur Zeit t an, die zweite Zeile den neuen Zustand der mittleren Zelle des Blocks zur Zeit $t+1$. Die Bezeichnung des Automaten ergibt sich aus der binären Darstellung der Vorschrift, d.h. $01101110_2 = 110$.

(a) Schreiben Sie ein Skript zur Realisierung des Automaten für N Zellen. Verwenden Sie einen zufälligen Anfangszustand und geben Sie zu jedem Zeitschritt den aktuellen Status des Automaten in binärer Darstellung aus. Pausieren Sie nach jedem Zeitschritt für eine Sekunde mit der Funktion `time.sleep()`.

(b) Betrachten Sie nun die Fälle $N = 10$ und $N = 11$ und berechnen Sie jeweils die Zeit t_p, bis ein Zustand zu einer beliebigen Zeit t ein zweites Mal zur Zeit $t + t_p$ erscheint. Mitteln Sie t_p über eine hinreichend große Zahl von zufällig gewählten Anfangszuständen. Was fällt Ihnen auf?

2. Quantencomputer

In einem Quantencomputer wird der Zustand des Systems durch einen Vektor ψ beschrieben. Das Betragsquadrat der i-ten Komponenten gibt dabei die Wahrscheinlichkeit an, bei einer Messung den (klassischen) Wert i zu erhalten. Da die Gesamtwahrscheinlichkeit sich auf 1 summieren muss, gilt $\sum_i |\psi_i|^2 = 1$. Operationen in einem Quantencomputer werden durch Matrizen beschrieben, der neue Zustand nach einer solchen Operation ergibt sich durch Matrix-Vektor-Multiplikation.

Im folgenden soll ein Quantencomputer mit zwei Quantenbits (Qubits) betrachten werden, dessen Zustand durch einen vierdimensionalen Vektor beschrieben wird. Der Anfangszustand ist durch den Vektor $\psi_0 = (1/2, 1/2, 1/2, 1/2)^t$ gegeben.

(a) Betrachten Sie die Funktion

$$f(x) = \begin{cases} -1 & \text{für } x = x_0 \\ 1 & \text{sonst} \end{cases} , \tag{1}$$

mit $x \in \{0, 1, 2, 3\}$, welche nur für x_0 den Wert -1 ergibt, und für die anderen drei Werte 1. Wie viele Funktionsaufrufe sind im Mittel nötig, um ein zufällig gewähltes x_0 auf einem klassischen Computer zu bestimmen?

(b) Die Funktion $f(x)$ lässt sich für einen Quantencomputer als Matrix U_f definieren, gemäß

$$\begin{aligned} U_{ii} &= -1 \quad \text{für } i = x_0 \\ U_{ii} &= 1 \quad \text{sonst} \\ U_{ij} &= 0 \quad \text{für } i \neq j. \end{aligned} \tag{2}$$

Schreiben Sie eine Python-Funktion mit x_0 als Parameter, welche die Matrix U_f zurückgibt und berechnen Sie das Produkt $U_f \psi_0$.

(c) Betrachten Sie nun die Matrix U_g, gegeben durch

$$U_g = \frac{1}{2} \begin{pmatrix} 1 & -1 & -1 & -1 \\ -1 & 1 & -1 & -1 \\ -1 & -1 & 1 & -1 \\ -1 & -1 & -1 & 1 \end{pmatrix} \tag{3}$$

Und berechnen Sie das Produkt $\psi_f = U_g U_f \psi_0$. Wie groß ist die Wahrscheinlichkeit, für den Zustand ψ_f als Messergebnis den Wert x_0 zu erhalten? Berechnen Sie hierfür die entsprechende Wahrscheinlichkeiten für jeden möglichen Wert von x_0 und diskutieren Sie das Ergebnis.

Programmieren für Physikerinnen und Physiker, Übung 6

PD Dr. Hendrik Weimer, WS 2019/20, 29.11.2019

1. Binärdateien

(a) Schreiben Sie ein Skript, das einen NumPy-Array mit Werten zwischen 0 und 2 in einem Abstand von 0.1 erstellt und speichern Sie diesen mit `numpy.save` ab.

(b) Schreiben Sie ein zweites Skript, welches die eben erstellte Datei mittels `read()` einliest. Die einzelnen Elemente des so erhaltenen Bytestrings sind Ganzzahlen zwischen 0 und 255. Geben Sie jedes Element (Byte) in Hexadezimal-Darstellung aus (hierfür können Sie `x` als Formatierungsanweisung verwenden). Zur besseren Lesbarkeit trennen Sie die einzelnen Bytes mit Leerzeichen ab und geben 16 Bytes pro Zeile aus.

(c) Wenn der Zahlenwert eines Bytes zwischen 32 und 126 liegt, entspricht dies einem druckbaren Wert im ASCII-Standard, welchen Sie mit der Funktion `chr()` erhalten können. Geben Sie parallel zur hexadezimalen Darstellung die druckbaren Bytes der Datei aus. Verwenden Sie als Platzhalter für nicht-druckbare Zeichen einen Punkt.

2. Box-Muller-Methode

Mit der Box-Muller-Methode können Sie aus zwei gleichverteilten Zufallszahlen x_1 und x_2 zwischen 0 und 1 mit der Vorschrift

$$y_1 = \sqrt{-\log x_1}\cos(2\pi x_2) \quad (1)$$

$$y_2 = \sqrt{-\log x_1}\sin(2\pi x_2) \quad (2)$$

zwei normalverteilte Zufallszahlen mit Mittelwert 0 und Standardabweichung 1 erhalten.

(a) Schreiben Sie eine Funktion, welche N normalverteilte Zufallszahlen mit Mittelwert μ und Standardabweichung σ zurückgibt.

(b) Prüfen Sie die Normalverteilung mit einem Shapiro-Wilk-Test durch die Funktion `scipy.stats.shapiro`. Der zweite Rückgabewert der Funktion gibt dabei eine untere Schranke für die Wahrscheinlichkeit an, dass es sich um eine Normalverteilung handelt. Vergleichen Sie mit der Student-t-Verteilung aus `numpy.random.standard_t` für verschiedene Werte des Parameters $n > 0$ der Verteilung. Diskutieren Sie das Ergebnis.

3. Klimawandel

(a) Laden Sie von `https://data.giss.nasa.gov/gistemp/graphs_v3/Fig.A2.txt` Messwerte zur Veränderung der globalen Temperatur und lesen Sie die Datei mit einem Skript ein. Zeigen Sie mit einem geeigneten statistischen Test, dass das Referenz-Intervall den Zeitraum von 1951–1980 beschreibt.

(b) Warum ist es sinnvoll, die Referenz auf den Zeitraum von 1951–1980 festzulegen? Belegen Sie Ihre Argumentation mit einer statistischen Analyse des Fünfjahresmittels über ein Zeitfenster von 30 Jahren.

(c) Schreiben Sie eine Funktion, die das Mittel der Temperaturabweichung für ein Zeitfenster von n Jahren um das Jahr x herum berechnet. Berechnen Sie den Verlauf dieser gemittelten Temperatur für alle Jahre für $n = 5$, $n = 10$ und $n = 25$.

(d) Visualisieren Sie Ihre Ergebnisse mit `matplotlib`.

Programmieren für Physikerinnen und Physiker, Übung 7

Dr. Hendrik Weimer, WS 2019/20, 06.12.2019

1. Quantenstatistik

In der Quantenstatistik werden Fermionen durch die Fermi-Dirac-Verteilung beschrieben, deren Wahrscheinlichkeitsverteilung gegeben ist durch

$$p(x) = \frac{1}{\log(2)} \frac{1}{e^x + 1}.$$

Schreiben Sie einen Zufallszahlen-Generator für diese Verteilung und prüfen Sie, dass die Gesamtwahrscheinlichkeit auf 1 normiert ist. Plotten Sie anschließend ein Histogramm für Zufallszahlen aus der Verteilung und zeigen Sie, dass das Histogramm für genügend große Anzahl von Zufallszahlen gegen die Wahrscheinlichkeitsverteilung konvergiert.

2. Fiese Funktionen

(a) Plotten Sie die Funktion $f(x) = \sin(x^{-1/5})$ im Intervall $10^{-9} \leq x \leq 1$ mit logarithmischer x-Achse.

(b) Plotten Sie die Funktion $g(x) = \exp(-1/x^2)$ im Intervall $-0.01 \leq x \leq 0.01$ mit logarithmischer y-Achse.

(c) Plotten Sie die Funktion

$$h(x) = \begin{cases} x^2 & \text{für } x \in \mathbb{Q} \\ 0 & \text{für } x \notin \mathbb{Q} \end{cases}$$

im Intervall $0 \leq x \leq 1$.

3. Bethe-Gitter (*)

(a) Im Bethe-Gitter hat jeder Gitterpunkt (außer den Endpunkten) z nächste Nachbarn. Jedem Gitterpunkt wird dabei eine ganzzahlige Distanz k zum Ursprung zugeordnet. Der Ursprung hat dabei $k = 0$, die z rotationssymmetrisch mit dem Urspung verbundenen Punkte haben $k = 1$. Zeigen Sie, dass für die Anzahl der Gitterpunkte zu einem bestimmten $k > 0$ gilt

$$N_k = z(z-1)^{k-1}. \tag{1}$$

(b) Schreiben Sie eine Funktion, die ein Geradenstück in Polarkoordinaten plottet (d.h. von r_1, φ_1 nach r_2, φ_2). Benutzen Sie diese Funktion um ein Bethe-Gitter mit $z = 3$ und bis zu $k = 6$ zu plotten.

Programmieren für Physikerinnen und Physiker, Übung 8

PD Dr. Hendrik Weimer, WS 2019/20, 13.12.2019

1. Rabi-Oszillationen

In der Quantenphysik wird das Treiben eines Systems zwischen zwei möglichen Zuständen (0 und 1) durch Rabi-Oszillationen beschrieben. Dabei beschreibt die Funktion

$$p_1(t) = \sin^2(\Omega t)$$

die Wahrscheinlichkeit, dass eine Messung des Systems den Wert 1 liefert (und mit Wahrscheinlichkeit $1 - p_1(t)$ den Wert 0). Bei den Messwerten handelt es sich also um einer Bernoulli-Verteilung folgenden Zufallszahlen, wobei der Parameter der Bernoulli-Verteilung zeitabhängig ist.

(a) Schreiben Sie ein Skript, das für 20 gleichverteilte Zeitpunkte zwischen $t = 0$ und $t = 2\pi/\Omega$ M Zufallszahlen (0 oder 1) liefert, wobei die Wahrscheinlichkeit für den Wert 1 gerade $p_1(t)$ entsprechen soll.

(b) Berechnen Sie den Mittelwert und den Standardfehler (Standardabweichung geteilt durch $\sqrt{M}$) für jeden Zeitpunkt und visualisieren Sie das Ergebnis als Funktion der Zeit für verschiedene Werte von M. Stellen Sie dabei den Standardfehler als Fehlerbalken dar.

2. Langtons Ameise

Langtons Ameise beschreibt die Bewegung einer einzelnen Figur auf einem zweidimensionalen Quadratgitter. Jedem Gitterplatz ist dabei eine Binärzahl zugeordnet, am Anfang soll das gesamte Gitter auf den Wert 0 initialisiert sein. Mit jedem Zeitschritt ändert die Ameise seine Bewegungsrichtung, invertiert den Binärwert des aktuellen Gitterplatzes und bewegt sich um einen Gitterplatz vorwärts. Zunächst sei die Ameise am Ursprung positioniert, mit der Bewegungsrichtung nach oben. Die Bewegung der Ameise folgt dann folgenden Regeln:

1. Auf einem Gitterplatz mit dem Wert 0 dreht sich die Ameise um 90° nach rechts, setzt den Wert des Gitterplatzes auf 1 und bewegt sich einen Gitterplatz vorwärts.
2. Auf einem Gitterplatz mit dem Wert 1 dreht sich die Ameise um 90° nach links, setzt den Wert des Gitterplatzes auf 0 und bewegt sich einen Gitterplatz vorwärts.

(a) Schreiben Sie ein Python-Skript, das Langtons Ameise realisiert. Visualisieren Sie ersten 200 Schritte der Ameise in einer Animation, welche die Veränderung der Gitterkonfiguration farblich darstellt. Verwenden Sie zur Darstellung der aktuellen Position der Ameise eine weitere Farbe.

(b) Berechnen Sie die ersten 15.000 Schritte der Ameise und stellen Sie diese in einer Zeitraffer-Animation dar.

3. Zoom in die Mandelbrot-Menge

Verwenden Sie das Skript `mandelbrot.py` aus der Vorlesung, um die Mandelbrot-Menge zwischen $(x, y) = (-2.5 - 1.5)$ und $(1, 1.5)$ zu visualisieren. Erweitern Sie das Skript, sodass die Regionen zwischen $(-0.8, 0.1)$ und $(-0.7, 0.2)$ sowie zwischen $(-0.79, 0.13)$ und $(-0.78, 0.14)$ als Insets dargestellt werden. Markieren Sie jeweils auf der nächsthöheren Ebene den Ausschnitt, in den hereingezoomt wird.

Programmieren für Physikerinnen und Physiker, Übung 9

PD Dr. Hendrik Weimer, WS 2019/20, 20.12.2019

1. Oszillierender Integrand

Berechnen Sie numerisch das Integral der Funktion $f(x) = \sin(mx)$ in den Grenzen zwischen 0 und 1 für verschiedene Werte von m und plotten Sie das Ergebnis in Abhängigkeit von m. Was passiert, wenn m sehr groß wird?

2. Berechnung mehrerer Lösungen

Die Gleichung $z^3 = 1$ besitzt im Komplexen drei verschiedene Lösungen.

(a) Berechnen Sie die drei Lösungen, indem Sie verschiedene Startwerte ausprobieren. Stellen Sie die komplexe Zahl dafür als zweidimensionalen Vektor zweier reeller Zahlen dar, d.h. $z = x_0 + ix_1$.

(b) Stellen Sie in Abhängigkeit des Startwerts grafisch dar, in welche Lösung das Verfahren konvergiert. Verwenden Sie drei unterschiedliche Farben, um die jeweilige Lösung zu kennzeichen.

3. Finite Size Scaling

In vielen Situationen ist es notwendig von numerischen Berechnungen für eine endliche Teilchen N auf den thermodynamischen Limes mit $N \to \infty$ zu schließen. Hierzu soll eine Funktion $x(N)$ betrachtet werden, welche die Form

$$x(N) = x_\infty + \frac{a}{N^b}$$

hat. Gegeben seien dabei folgende Daten:

N	$x(N)$
100	0.23552325
200	0.19156163
500	0.16732625
1000	0.14938689

(a) Bestimmen Sie die Parameter x_∞, a und b indem Sie die Funktion $x(N)$ an die Daten fitten (Hinweis: Sie können durch einen dritten optionalen Parameter der Funktion `curve_fit` eine Liste von Startwerten für den Fit vorgeben.) Begründen Sie durch einen doppellogarithmischen Plot der Daten (x in Abhängigkeit von $1/N$) und der gefitteten Funktion, warum die obige Form der Funktion sinnvoll ist.

(b) Der zweite Rückgabewert von `curve_fit` ist die Kovarianzmatrix, welche die Ungenauigkeit des Fits angiebt. Die Unsicherheiten der Fit-Parameter sind dabei die Quadratwurzeln der Diagonalelemente der Kovarianzmatrix. Ist der Wert für x im thermodynamischen Limes von Null verschieden?

Programmieren für Physikerinnen und Physiker, Übung 10

PD Dr. Hendrik Weimer, WS 2019/20, 10.01.2020

1. Einschanzentournee

Eine punktförmige Skispringerin springt von einem infinistesimalen Schanzentisch einen um 45° geneigten Hang hinunter. Beim waagerechten Absprung hat sie eine Geschwindigkeit in x-Richtung von $v_x = 90\,\mathrm{km/h}$.

(a) Berechnen Sie numerisch die Trajektorie der Skispringerin $(x(t), y(t))$ und plotten Sie den Weg, den Sie in der Luft zurücklegt. Welche Weite hat Sie bei der Landung erzielt?

(b) Durch einen kräftigen Absprung kann die Springerin einen zusätzlichen Geschwindigkeitsschub von $\Delta v = 5\,\mathrm{m/s}$ erreichen. Die Richtung dieses Schubs kann dabei durch ihre Absprungtechnik beliebig gewählt werden. Berechnen Sie numerisch die für eine große Weite optimale Absprungsrichtung und plotten Sie für diesen Fall wiederum ihren Sprung.

2. Let it snow

Betrachten Sie die Dynamik von Schneeflocken auf einemn eindimensionalen Gitter unendlicher Größe. An jedem Gitterplatz kann maximal eine Schneeflocke sitzen. Zu jedem Zeitschritt kann jede Schneeflocke mit der Wahrscheinlichkeit $p_- = 0.1$ schmelzen, wodurch sie verschwindet. Zudem können zu jedem Zeitschritt neue Schneeflocken vom Himmel schneien. Dabei soll angenommen werden dass neue Schneeflocken nur mit einer Wahrscheinlichkeit p_+ an den Gitterplätzen entstehen können, an deren Nachbarplätzen bereits eine Schneeflocke sitzt.

(a) Schreiben Sie ein Python-Skript, welches das Langzeit-Verhalten für verschiedene Werte von p_+ berechnet, beginnend mit einer einzelnen Schneeflocke im Ursprung des Gitters. (Hinweis: Speichern Sie zu jeder Schneeflocke die aktuelle Position als Integer-Variable, um ein beliebig großes Gitter simulieren zu können.)

(b) Wie sind die Aussichten für weiße Weihnachten (kein Verschwinden aller Schneeflocken) in Abhängigkeit von p_+?

Als kleines Weihnachtsgeschenk werden alle Aufgaben wie Zusatzaufgaben gewertet. Alles Gute für 2020!

Programmieren für Physikerinnen und Physiker, Übung 11

PD Dr. Hendrik Weimer, WS 2019/20, 17.01.2020

1. **Mathematisches Pendel** Ein mathematisches Pendel der Länge $l = 0.5\ m$ werde beschrieben durch den Auslenkungswinkel φ. Dessen Bewegung folgt bekanntlich der Differentialgleichung

$$\ddot{\varphi} + \omega^2 \sin(\varphi) = 0 \tag{1}$$

wobei $\omega = \sqrt{\frac{g}{l}}$ die Kreisfrequenz des Pendels ist mit g der Schwerebeschleunigung.

   (a) Schreiben Sie ein Skript, dass numerisch die Bewegung des Pendels berechnet. Schreiben Sie dafür die Bewegungsgleichung (1) in eine Differentialgleichung erster Ordnung um. Berechnen Sie die Bewegung des Pendels für eine Auslenkung von $\varphi_0 = 15\,\mathrm{deg}$ und verschwindender Anfangsgeschwindigkeit. Stellen Sie das Ergebnis in einer Animation dar.

   (b) Berechnen Sie die Trajektorie des Pendels für $\varphi_0 = 0$ und verschiedene Anfangsgeschwindigkeiten und stellen Sie die Ergebnisse im Phasenraum dar. Ab welcher Geschwindigkeit schwingt das Pendel über? Stellen Sie hierfür eine geeignete Gleichung auf und lösen Sie diese numerisch.

2. **Sonnenflecken**

   Fluktuationen des solaren Magnetfeldes führen zu dunklen Bereichen auf der Sonnenoberfläche, den sogenannten Sonnenflecken. In der Datei `sunspots.txt` ist die Anzahl der Sonnenflecken seit Beginn der Aufzeichnungen tabelliert. Die erste Spalte nummeriert die Monate n seit Januar 1749, die zweite Spalte zeigt die Anzahl der Sonnenflecken s_n.

   (a) Schreiben Sie ein Skript, welches die Daten einliest und in einem Array speichert. Berechnen Sie den laufenden Mittelwert $S_{r,n} = \sum_{k=n-r}^{n+r} s_n/(2n+1)$, und plotten Sie die Rohdaten sowie die laufenden Mittelwerte für $r = 5$, $r = 20$ und $r = 30$. Stellen Sie auf der x-Achse das Datum dar (z.B. mittels `time.strptime` und `time.mktime` sowie `matplotlib.dates.epoch2num` und `matplotlib.pyplot.plot_date`). Reduzieren Sie den Plot auf die letzten 1000 Datenpunkte.

   (b) Berechnen Sie die Koeffizienten a_k der diskreten Fouriertransformation der Sonnenfleckendaten mit der Funktion `numpy.fft.fft`. Der erste Koeffizient ist der (hier uninteressante) Mittelwert; setzen Sie diesen manuell auf Null. Plotten Sie die Koeffizienten als Punkte in der komplexen Ebene. Plotten Sie außerdem die Betragsquadrate $|a_k|^2$ als Funktion der Fourierfrequenzen f_k (erhältlich mit `numpy.fft.fftfreq`) sowie als Funktion der Perioden $1/f_k$ in Jahren. Warum ist das Spektrum symmetrisch? Reduzieren Sie den Plot auf Perioden zwischen 0 und 30 Jahren, so dass die Periodizität des Sonnenzyklus von $\sim$11 Jahren gut sichtbar wird.

3. Diffusionsbegrenztes Wachstum

Auf einem quadratischen $L \times L$ Gitter soll die Brownsche Bewegung von Teilchen modelliert werden.

(a) In jedem Zeitschritt wird eine zufällige Bewegungsrichtung ausgewählt (rechts, links, oben oder unten) und das Teilchen um einen Gitterplatz in die entsprechende Richtung bewegt. Wenn das Teilchen durch den Schritt das Gitter verlassen würde, wird der Schritt verworfen und erneut eine zufällige Richtung ausgewählt. Setzen Sie ein Teilchen auf den mittleren Platz eines Gitters mit $L = 101$, und plotten Sie den Pfad des Teilchens für 10000 Schritte.

(b) Nun wollen wir „klebrige“ Teilchen betrachten. Zu Beginn befindet sich ein einziges Teilchen in der Mitte des Gitters. Das Teilchen folgt einer Brownschen Bewegung, bis es an den Rand des Gitters stößt und dort kleben bleibt. Dann erscheint ein zweites Teilchen in der Mitte, und bewegt sich wiederum, bis es entweder am Rand oder am ersten Teilchen kleben bleibt, und so weiter. Jedes neue Teilchen beginnt in der Mitte und bewegt sich so lange, bis es am Rand oder einem anderen Teilchen kleben bleibt (die Teilchen kleben auch an diagonal benachbarten Plätzen zusammen). Es erscheinen neue Teilchen, bis das zuletzt erschienene Teilchen schon auf seinem Startplatz an einem benachbarten Teilchen kleben bleibt. Simulieren Sie diesen Prozess auf einem Gitter mit $L = 101$. Erstellen Sie ein Bild des Ergebnisses.

(c) Erstellen Sie eine Animation, welche zeigt, wie sich die einzelnen Teilchen nach und nach am Rand und aneinander anlagern. (Die Brownschen Bewegungen müssen nicht mit animiert werden, nur die finalen Positionen.)

Programmieren für Physikerinnen und Physiker, Übung 12

PD Dr. Hendrik Weimer, WS 2019/20, 24.01.2020

1. Sympy-Ausdrücke manipulieren

(a) Formen Sie den Ausdruck $p = (x^4 + x^3 - 4x^2 - 4x)/(x^4 + x^3 - x^2 - x)$ um zu $(x-2)(x+1)(x+2)/(x^3 + x^2 - x - 1)$.

(b) Formen Sie den Ausdruck $q = (x + z^2 + 1)(y - z^2 - 1)$ um zu $xy - x(z^2+1) + y(z^2+1) - (z^2+1)^2$.

Benutzen Sie hierzu die sympy-Funktionen `expand`, `factor`, `collect`, `use` und `epath` bzw. `EPath.apply`.

2. Analytische Integration

Gegeben sei das Integral

$$\int_0^\infty dx\, \frac{\sin(x)}{\sqrt{x}}. \tag{1}$$

(a) Versuchen Sie zunächst, das Integral numerisch mit `scipy.integrate.quad` zu berechnen.

(b) Berechnen Sie das bestimmte Integral direkt mit SymPy. Geben Sie den numerischen Wert des Integrals aus und vergleichen Sie mit dem von SciPy berechneten Ergebnis.

(c) Berechnen Sie die Stammfunktion des zugehörigen unbestimmten Integrals. Wandeln Sie das Ergebnis mit der `lambdify`-Funktion in eine Python-Funktion um. Zeigen Sie analytisch durch Auswerten der Stammfunktion an den Grenzen, dass Sie dasselbe Ergebnis wie zuvor erhalten.

(d) Zeigen Sie mit SymPy, dass der Integrand an der Stelle $x = 0$ verschwindet.

3. Getriebener harmonischer Oszillator

Gegeben sei die Differentialgleichung

$$\ddot{x}(t) = -x(t) - \dot{x}(t) + \cos(\omega t).$$

Berechnen Sie die Lösung dieser Differentialgleichung mit SymPy und plotten Sie das Ergebnis mit Matplotlib für verschiedene Werte von ω um den Wert $\omega = 1$ herum.

Programmieren für Physikerinnen und Physiker, Übung 13

PD Dr. Hendrik Weimer, WS 2019/20, 31.01.2020

1. Modulare Arithmetik

Erweitern Sie das Skript `mint.py` um die Funktionen Multiplikation und Potenzieren. Decken Sie zudem den Fall ab, wenn einer der Operanden eine Fließkommazahl ist. Das Ergebnis soll hierbei auf die nächste Ganzzahl gerundet werden.

2. Betragsweiser Vergleich komplexer Zahlen

Schreiben Sie eine Subklasse zur Klasse `complex`, welche die Vergleichsoperationen $<$, $>$, $\leq$ und $\geq$ unterstützt, indem diese Operationen betragsweise durchgeführt werden.

3. Vererbung

Schreiben Sie eine Klasse, die einen Zylinder definiert und eine Methode zur Berechnung des Volumens zur Verfügung stellt.

(a) Starten Sie hierzu von einer Superklasse, die allgemeine geometrische Objekte definiert. Schreiben Sie nun zwei Subklassen für einen Kreis und eine Strecke mit entsprechenden Methoden zur Berechnung der Kreisfläche und Länge der Strecke. Schreiben Sie nun die Klasse für Zylinder als Subklasse von Kreis und Strecke.

(b) Die Addition zweier Zylinder mit gleichem Radius bedeutet natürlicherweise, dass diese an ihren Grundflächen zusammengesetzt werden. Überladen Sie den `+`-Operator mit einer entsprechenden Methode.