Rechnernutzung in der Physik: Zusatz Python-Einführung

Institut für Experimentelle Teilchenphysik
Institut für Theoretische Teilchenphysik

Prof. Dr. G. Quast, Prof. Dr. M. Steinhauser
Dr. Th. Chwalek, Dr. A. Mildenberger
`http://comp.physik.kit.edu`

WS2018/19 – Zusatzblatt 01
Bearbeitungszeitraum: bis Di, 06.11.2018

Die Python-Einführung (5 Aufgabenblätter) ist von denjenigen Teilnehmern zu bearbeiten, die die Rechnernutzung im Umfang von 6 Leistungspunkten absolvieren möchten (GymPO, BSc Physik SPO 2010) und nicht bereits erfolgreich an der Veranstaltung *Computergestützte Datenauswertung* teilgenommen haben. In diesem Teil sind 80% der Pflichtaufgaben erfolgreich zu bearbeiten.

Aufgabe 101: Aufsetzen Ihrer Arbeitsumgebung

Sie sollten für diesen Kurs eine eigene Arbeitsumgebung auf Ihrem eigenen System aufsetzen. Dazu können Sie entweder die virtuelle Maschine nutzen, siehe `http://www.ekp.kit.edu/~quast/VM-DaA`, oder Sie installieren die notwendige Software unter Windows, auf macOS oder auf Ihrem eigenen Linux-System. Hinweise dazu finden Sie auf der Homepage der Vorlesung unter Vorlesungsfolien.

Falls Sie keinen eigenen Rechner besitzen, können Sie die Computer im Poolraum Physik nutzen; dort ist das Betriebssystem Linux mit der grafischen Benutzeroberfläche KDE installiert.

Treffen Sie eine Entscheidung, welche der Möglichkeiten Sie nutzen möchten, und führen Sie die notwendigen Schritte zur Installation auf Ihrem System aus. Sie können Ihre Entscheidung später jederzeit revidieren.

Neben dem Arbeiten mit der Maus lassen sich viele Dinge auch mit Textbefehlen erledigen, und das häufig sogar schneller. Um Befehle eingeben zu können, benötigen Sie ein sogenanntes Terminal (Textfenster) - die `Konsole` unter Linux bzw. die `Eingabeaufforderung` unter Windows - wenn Sie dazu die Anwendung `WinPython Command Prompt` verwenden, steht Ihnen die volle Python-Umgebung zur Verfügung.

Bevor Sie zu eigenen Unternehmungen aufbrechen, testen Sie Ihre neu aufgesetzte Umgebung mit den Beispielen aus der Vorlesung. Laden Sie alle Dateien von der Vorlesungsseite `http://comp.physik.kit.edu/Lehre/Rechnernutzung/` herunter und wechseln Sie aus der Konsole mit dem Befehl `cd` in das entsprechende Verzeichnis. Geben Sie dann der Reihe nach die folgenden Befehle ein:

- `python simplePlot.py`
- `python simpleData.py`
- `python animate_wave.py`
- `python plot3dfunction.py`
- `python animated_Gauss.py`
- `python animate_pi.py`

Zunächst werden wir ausschließlich die Programmiersprache `Python` mit der Entwicklungsumgebung `idle` nutzen. Starten Sie die Anwendung `idle` oder einen Texteditor und schauen Sie ich damit jeweils den Quellcode der Dateien an, um einen ersten Eindruck von der Sprache *Python* und ihren Möglichkeiten zu bekommen. Die elementaren Grundlagen zum Verständnis und für Ihre eigene Anwendungen werden wir uns im Laufe dieses Kurses erarbeiten. Diese Beispiele oder Teile davon können Sie später für eigene Entwicklungen nutzen.

Aufgabe 102: Kennenlernen von Python

Um einige Grundlagen der Programmiersprache `Python` kennen zu lernen, sollten Sie die ersten Kapitel eines Python-Kurses für (Programmier-)Anfänger durcharbeiten:

Besuchen Sie dazu `http://www.python-kurs.eu/kurs.php` und führen Sie die ersten Beispiele des *Python 2 Tutorial* in Ihrer eigenen Arbeitsumgebung aus. Sie sollten ihn mindestens bis zum Kapitel *Operatoren* durcharbeiten, es schadet aber nicht, wenn Sie bis zum Kapitel *Formatierte Ausgabe* gelangen.

Als kleinen **Test des Gelernten** schreiben Sie nun ein eigenes Programm, das die **Exponentialfunktion** eines Eingabe-Wertes berechnet. Orientieren Sie sich am Beispiel-Programm `kehrwert.py` von der Kurswebseite. Um die `numpy.exp()`-Funktion nutzen zu können, verwenden Sie z.B. `import numpy` am Programmbeginn. In welchem Wertebereich liefert das Programm sinnvolle Werte?

Hinweis zur Vorgehensweise beim Programmieren: Starten Sie einen Editor bzw. die Entwicklungsumgebung `idle`. Während der Programmentwicklung empfiehlt es sich, neu eingegebenen Code regelmäßig zu testen. Speichern Sie dazu den aktuellen Stand ihres Programmcodes ab und führen ihn aus (entweder innerhalb von `idle`, oder durch Aufruf von `python` auf der Kommandozeile). Es ist auch üblich und sinnvoll, mit jeweils geeignet platzierten `print`-Befehlen zu überprüfen, ob wirklich genau das geschieht, was Sie sich vorgestellt hatten. Es ist normal, dass Python Sie gelegentlich mit Fehlermeldungen konfrontiert, die auf den ersten Blick nicht immer einsichtig sind. Korrigieren und testen Sie ihren Code und bauen Sie ihre Programme so schrittweise aus gut getesteten Einzelkomponenten auf, bis Sie am Ende ein zufriedenstellendes Gesamtergebnis erhalten.

Aufgabe 103: Arbeiten mit `numpy`

Für das Arbeiten mit Daten sind effiziente Datenstrukturen notwendig, die den einfachen Umgang mit Vektoren oder allgemein Tensoren erlauben. Dazu dient das Python-Paket `numpy`, in das Sie sich nun ein wenig einarbeiten sollen. Das einfache `numpy`-Tutorial unter dem Link http://python-kurs.eu/numerisches_programmieren_in_Python.php. gibt eine gute Einführung. Machen Sie sich mit der grundlegenden Funktionalität vertraut, insbesondere, wie man die allem zu Grunde liegenden Datenstrukturen, „numpy-arrays“, mit Daten füllt und mit ihnen arbeitet. Schauen Sie sich diesen Kurs bis zum Kapitel *Numerische Operationen auf Numpy-Arrays* an. Eine sehr angenehme Eigenschaft von `numpy` ist die vektorisierte Verarbeitung von Daten, d. h. arithmetische Operatoren und Funktionen wirken elementweise auf ganze `arrays`.

Für die Problemstellungen, die wir in diesem Kurs behandeln, sind die Funktionen wichtig, die das Erzeugen und Initialisieren von `arrays`, das Füllen mit Datensequenzen und Zufallszahlen (siehe und recherchiere `numpy.zeros()`, `numpy.linspace()`, `numpy.random.rand()` und `numpy.random.randn()`), die Berechnung von Minimum, Maximum, Mittelwert und anderen statistischen Größen eines `arrays` oder auch Skalar- und Vektorprodukt von zwei `arrays` ermöglichen. Das `numpy`-Paket liefert auch eine große Zahl an mathematischen Funktionen (sin(), cos(), exp() usw.), die ebenfalls elementweise operieren und daher als Eingabe sowohl einfache Zahlen als auch `arrays` akzeptieren.

Sie können nun als Anwendung des Gelernten folgende kleine **Aufgabe** programmieren:
Schreiben Sie ein Programm, das eine Zufallszahl ausgibt, die dem Wurf mit einem Würfel entspricht. Nutzen Sie die Funktion `numpy.random.rand()`, die eine im Intervall $]0, 1]$ gleichverteilte Zufallszahl zurück gibt und überlegen Sie, durch welche Operationen Sie daraus eine Ausgabe der Zahlen $\{1, 2, \ldots, 6\}$ mit der Wahrscheinlichkeit von $1/6$ erzeugen können. Erzeugen Sie $N = 120$ solcher Zufallszahlen $i \in \{1, ..., 6\}$ und geben Sie die Häufigkeit aus, mit der jede der Zahlen vorgekommen ist.

Aufgabe 104: Arbeiten mit `matplotlib`

Der erste Schritt einer jeden Arbeit mit Daten besteht in deren Visualisierung. Dazu stellt das Python-Paket `matplotlib` eine Vielzahl einfach zu verwendender Methoden bereit. Natürlich gibt es auch zu `matplotlib` ein Tutorial (http://matplotlib.org/users/pyplot_tutorial.html oder auch http://python-kurs.eu/matplotlib.php), die Sie sich kurz anschauen sollten. Da `matplotlib` ein sehr mächtiges Paket ist, sollten Sie sich daran gewöhnen, Methoden und Parameter bei Bedarf zu recherchieren. Es ist auch übliche und legitime Praxis, die `matplotlib`-Beispiele zu verwenden und an das eigene Problem anzupassen bzw. Code-Fragmente daraus in eigene Programme zu übernehmen.

Als erste Anwendung kommen wir noch einmal auf Aufgabe 103 zurück. Dort hatten Sie die Häufigkeiten des Auftretens der einzelnen Zahlen beim Würfelspiel bestimmt. Wenn diese Häufigkeiten in Form eines `arrays` mit Namen `h` vorliegen, können Sie das Ergebnis mit der Funktion `matplotlib.pyplot.bar()` als Balkengrafik darstellen. Importieren Sie dazu zunächst `matplotlib.pyplot` unter dem Alias-Namen `plt` in

Ihr Programm. Mit den Befehlen `plt.bar([1,2,3,4,5,6], h)` und `plt.show()` erzeugen Sie die Grafik.

Bearbeiten Sie nun folgende (Standard-) **Aufgabe**:
Stellen Sie eine Parabel $f(x) = x^2$ im Wertebereich $x \in [0., 5.]$ grafisch dar. Erzeugen Sie simulierte, fehlerbehaftete „Datenpunkte“ für $x \in \{1., 2., 3., 4.\}$, die jeweils dem Wert $f(x)$ mit einer Gauß-förmigen Unsicherheit von 10 % des wahren Werts entsprechen. Tragen Sie die Datenpunkte mit Fehlerbalken in die eigene Grafik ein.

Hilfe: Verwenden Sie `np.linspace()`, um 100 x-Werte zwischen 0. und 5. zu erzeugen. Berechnen Sie die zugehörigen y-Werte und verwenden Sie `plt.plot()` um die Parabel zu zeichnen. Erzeugen Sie einen zweiten `numpy-array` mit den x-Werten {1.,2.,3.,4.} und berechnen Sie wieder die zugehörigen y-Werte. Erzeugen Sie nun mit Hilfe von `np.random.randn()` ein `numpy-array` mit vier Zufallszahlen aus einer Standard-Normalverteilung. Wenn Sie diese Werte mit der gewünschten Unsicherheit (also $0.1 \cdot x^2$) multiplizieren, erhalten Sie die Zufallskomponente eines jeden Datenpunktes, die Sie zu den eben berechneten y-Werten addieren. Tragen Sie die so erzeugten Datenpunkte mit `plt.errorbar()` in Ihr Diagramm ein. Einige matplotlib-Funktionen haben die Eigenart standardmäßig alle Punkte mit Linien zu verbinden, ein Verhalten das bei Messdaten idR. eher nicht gewünscht wird. Durch z.B. die Option `fmt="o"` lässt sich dieses Verhalten umschalten.

Bringen Sie nun noch Achsenbeschriftungen an (siehe `plt.xlabel(), plt.ylabel()`. Vergessen Sie am Ende nicht die Zeile `plt.show()`, damit Ihre Grafik auch auf dem Bildschirm erscheint!

Aufgabe 105: Darstellen von Funktionen (*) **Testat**

In dieser Aufgabe soll eine Schar von Funktionen mit Hilfe von `numpy / scipy / matplotlib` dargestellt werden. Als Beispiel zur Verwendung sehen Sie sich bitte die Vorlesungsbeispiele und insbesondere das Programm `PlotBeispiel.py` von der Webseite an, das Sie als Vorlage für diese Aufgabe modifizieren können.

Aufgabe: Zeichnen Sie eine Schar von Resonanzkurven eines mit einer harmonischen Kraft $F_0 \sin(\omega t)$ angetriebenen harmonischen Oszillators mit Resonanzfrequenz ω_0, statischer Amplitude $A_s = F_0/m$ und Dämpfung $D = \gamma/\omega_0$, die Sie aus der klassischen Mechanik kennen,

$$A(\eta = \omega/\omega_0, A_s, D) = \frac{A_s}{\sqrt{(1-\eta^2)^2 + (2\eta D)^2}}$$

Stellen Sie für $A_s = 1.0$ im Bereich $\eta \in [0., 3.]$ eine Schar von Resonanzkurven mit verschiedenen Parametern $D \in [0.05, 3.]$ grafisch dar.

Hilfe: In der Vorlage wird eine `python`-Funktion `MyFunction()` zur Berechnung der Y-Werte aus einem `numpy-array` von X-Werten definiert. Diese Funktion sollten Sie modifizieren, so dass die Werte der Resonanzkurven für die entsprechenden Parameter berechnet werden.
Hinweis: Da Sie recht viele Kurven einzeichnen wollen, empfiehlt es sich, die Werte der gewünschten Parameter aus einer Liste zu entnehmen, z. B. für $D = 0.1$, $D = 1.0$ und $D = 3.0$:

```
for D in (0.1, 1., 3.0):
   Y = ...
   plt.plot(...)
```

Anmerkung: Das Script zur Lösung dieser Aufgabe ergibt eine recht allgemeine Vorlage, die Sie auch später zur Darstellung von Kurven immer wieder verwenden (und ggf. verfeinern) können.

Hinweis: Mit dem Rechnernamen `fphctssh.physik.uni-karlsruhe.de` können Sie von überall aus mittels ssh/scp Programm per Netzwerk auf einen Poolrechner zugreifen.
