Praktische Übung: Computergestützte Datenauswertung

Institut für Experimentelle Teilchenphysik

Prof. Dr. G. Quast
Dr. Andreas Poenicke
`http://comp.physik.kit.edu`

SS16 – Blatt 02
zu bearbeiten am 9.5. (Gr. a) bzw. 23.5. (Gr. b)

Übung 2.1: Kennenlernen von Python (2)
Setzen Sie das `Python 2`-Tutorial (http://www.python-kurs.eu/kurs.php) aus der letzten Übung fort. 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 `math.exp()`-Funktion nutzen zu können, verwenden Sie z.B. `import math` am Programmbeginn. In welchem Wertebereich funktioniert das Programm?
Hinweis zur Vorgehensweise bei 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.

Übung 2.2: 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 `array`s.
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(), numpy.random.randn()`), die Berechnung von Minimum, Maximum, Mittelwert und anderen statistischen Größen eines `array`s oder auch Skalar- und Vektorprodukt von zwei `array`s 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.

Übung 2.3: 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), das 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 2.2 zurück. Dort hatten Sie die Häufigkeiten des Auftretens der einzelnen Zahlen beim Würfelspiel bestimmt. Wenn diese Häufigkeiten in Form eines `array`s 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 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 $0.1 * x^2$) 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. 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!
Übrigens: Ähnliche Problemstellungen, also der Vergleich von Daten mit einer Modellfunktion, treten in der Datenauswertung sehr häufig auf, und es lohnt sich daher, an Ihrem Code sehr sorgfältig zu arbeiten, damit Sie ihn oder Teile davon später weiter verwenden können. Trennen Sie daher die Erzeugung der darzustellenden Daten von der eigentlichen Darstellung; denken Sie darüber nach, eine eigene Funktion zu definieren, in der Sie die grafische Darstellung kapseln. Sehen Sie Optionen vor, um die grafische Darstellung zu beeinflussen (Farben, Linienbreiten, Form und Größe der Marker, Größe der Achsenbeschriftung, Art der Achsen usw.). Sie können die Funktionalität später noch verbessern und erweitern, an neue Anforderungen anpassen und die Anwendung flexibilisieren (z. B. das Setzen der Grafik-Optionen wiederum in eine eigene Funktion auslagern). Und: vergessen Sie nicht, Ihren Code zu dokumentieren, also sorgfältig mit Kommentaren zu versehen, die die grundsätzliche Funktion, Ein- und Ausgabeparameter und evtl. “Programmiertricks“ beschreiben. Sie erhalten dann Ihr eigenes, flexibles Grafik-Werkzeug für den Vergleich von Daten mit Modellen, auf das Sie immer wieder zurückgreifen 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.
