Exercises for
Cosmology & Galaxies (Summer Term 2018)

Lecturer: Christoph Pfrommer & Lutz Wisotzki; Exercises: Kristian Ehlert

Introduction to numerical algebra and plotting with python

This document gives a brief introduction to the usage of numerical integration, root-finding schemes as well as an example script for a simple plot with python. Many tutorials can be found on the web dealing with python. Guides on how to install python can easily found there for complete beginners. The examples are hopefully sufficiently close to the actual calculations needed in the exercises such that even beginners to general programming should be able to apply necessary changes to solve the exercise. If questions arise, try googling as python is quite popular and the documentation usually well written. Otherwise, drop an email to Kristian (kehlert@aip.de). Information related to found bugs is also most welcome.

TO NOTE: Correct indention is crucial in python. Unfortunately, many pdf viewers neither preserve spaces nor the more relevant tabs correctly. You will have to make them yourself or use the python script provided on the website that includes all shown examples. The following code was tested on a recent version of python 2.7.x The most recent version is 3.5.x. However, the only big obvious change compared to 2.7.x deals with the print function, for which the code can easily be adopted.

1. Numerics

(a) Integrating

For this example, we want to solve the following integral with the free parameters $a, b \in \Re$:

$$\int_{\text{lower}}^{\text{upper}} \mathrm{d}x f(x) = \int_{\text{lower}}^{\text{upper}} \mathrm{d}x \frac{ax^3 - x^2}{x^7 + bx^{-1}}. \tag{1}$$

We import necessary packages:

```
import numpy as np
import scipy.integrate as integrate
```

Defining a function in python that defines $f(x)$, which we want to integrate (note, add a point to make sure python uses floats instead of integers in these calculations, i.e. write 2. instead of 2):

```
def f(x, a, b):
    return (a*x**3.-x**2.)/(x**7.+b*x**(-1.))
```

Defining a generic integration function:

```
def IntGeneric(function, lower, upper, parameters):
    result, error =  integrate.quad(function, lower, upper, args=
   parameters)
    return result
```

Now, we can choose our values for the lower/upper bounds lower, upper and the parameters a, b and integrate (Note, we define the parameters as a tuple using the round brackets $(,)$. Lists defined with square brackets $[,]$ are unsuitable here.). We can also just pass our function f to the integration function.

```
a = 2.
b = 4.5
lower = 0.
upper = 100.
parameters = (a, b)
result = IntGeneric(f, lower, upper, parameters)
```

Finally, you may display the result:

```
print 'The integral yields: %g' %result
The integral yields: 0.298517
```

(b) **Root finding**

Assume, we want to find the value for x that solved an equation of the form:

$$f(x) = g(x) \tag{2}$$

For this, we define $h(z) \equiv f(x) - g(x)$ and rather solve:

$$h(z) = 0 \tag{3}$$

Now, given a user-defined initial value, the numerical solver looks iteratively for the interception through 0, which is also referred to as *root finding*.

In our example, we use:

$$f(x) = x^2 - 7;\ g(x) = \frac{1}{4}\log(x) \tag{4}$$

Analogously to the previous example, we import necessary packages in python;

```
import scipy.optimize as root
import numpy as np
```

and we define $f(x)$ and $g(x)$:

```
def f(x):
    return x**2-7

def g(x):
    return 1./4.*np.log10(x)
```

The root-finding algorithm necessitates the usage of lambda definitions for functions. To illustrate, an alternative definition of $f(x)$ in the lambda framework is given as:

```
f = lambda x: x**2-7
```

The function that finds our root can then be defined as:

```
def findRoot(function, initialguess):
    return root.newton(lambda t: function(t), initialguess)
```

To solve the equation, we just choose an initial value and call our root-finding function, pass the function $f(x) - g(x)$ we would like to solve:

```
initialguess=1
result = findRoot(lambda x: f(x)-g(x),initialguess)
```

and print the result:

```
print 'The nummerical solver gives %g' %result
The nummerical solver gives 2.66579
```

(c) **Solving an equation including an integral numerically**
Here, we solve a combination of the previous cases, i.e. an equation of the form:

$$k(x) = g(x) \tag{5}$$

$$\int_{\mathrm{x}}^{\mathrm{upper}} \mathrm{d}t f(t) = g(x) \tag{6}$$

$$\int_{\mathrm{x}}^{\mathrm{upper}} \mathrm{d}t \frac{at^3 - t^2}{t^7 + bt^{-1}} = \frac{a}{4} \log 10(x). \tag{7}$$

with free parameters $a, b \in \Re$. For convenience, very similar equations to the previous where used here. Only note, the added free parameter in $g(x)$. We import the necessary packages:

```
import scipy.integrate as integrate
import scipy.optimize as root
import numpy as np
```

For completeness, we define the functions:

```
def f(x, a, b):
    return (a*x**3.-x**2.)/(x**7.+b*x**(-1.))
def g(x, a):
    return a/4.*np.log10(x)
```

We will need a function to integrate $f(t)$:

```
def IntGeneric(function, lower, upper, parameters):
    result, error = integrate.quad(function, lower, upper, args=
   parameters)
    return result
```

Some small additions are needed compared to the previous example to adapt our root-finding function to the current problem:

```
def findRootComplx(function, upper,initialguess):
    return root.newton(lambda z: function(z), initialguess)
```

This allows us to solve the initially stated equation for a choice of the upper integration boundary *upper*, the parameters a, b and the initial guess:

```
parameters = (3.,2.)
upper = 4.
initialguess = 1.
result = findRootComplx(lambda z: IntGeneric(f, z, upper,
    parameters)-g(z, parameters[0]), upper, initialguess)
```

and print the result:

```
print 'The nummerical solver gives %g' %result
The nummerical solver gives 1.66864
```

2. Plotting

(a) Simple example

This short example will teach the basics of plotting a function. We will use the function from the first tutorial with a fixed lower boundary and $a = b = 1$, i.e.

$$f(x) = \int_{-4}^{x} \mathrm{d}y \frac{y^3 - y^2}{y^7 + y^{-1}}. \tag{8}$$

and plot it for varying upper boundary $0 < x < 100$ (see Figure 1).

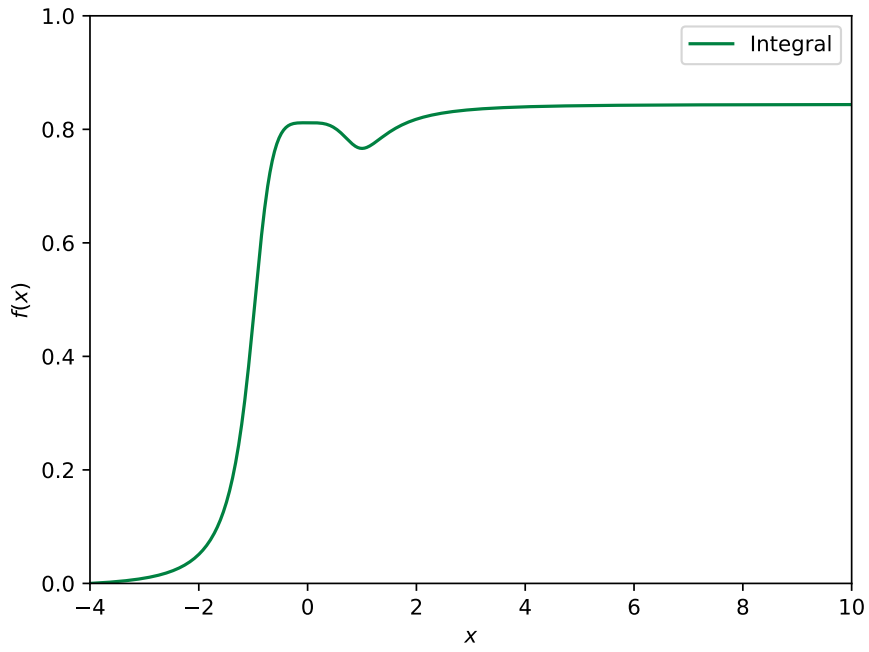

Figure 1: Our simple plot of an integral.

First, we import our packages and define our function $f(x)$ as well as the integration function as before:

```
import scipy.integrate as integrate
import numpy as np
```

```
def f(x, a, b):
    return (a*x**3.-x**2.)/(x**7.+b*x**(-1.))

def IntGeneric(function, lower, upper, parameters):
    result, error = integrate.quad(function, lower, upper, args=
   parameters)
    return result
```

Then, we have to initialize our x values as an array X.

```
X = np.linspace(-4.,10.,1e3)
```

We calculate the corresponding y-values one by one, so we store our results in a list Y. We have to initialize the list Y and then add our results with the build-in function *append*.

```
Y = []
parameters = (1.,1.)
for x in X:
    y = IntGeneric(f,-4.,x,parameters)
    Y.append(y)
```

To start with the actual plot, we import the popular matplotlib library.

```
import matplotlib.pylab as plt
```

We initialize our figure (i.e. the 'page') and an axes (i.e. the box on the page, the actual graph is plotted on).

```
fig = plt.figure()
ax = plt.axes()
```

The default colors provided by python are limited in number and aesthetics. Thus, we define our color through the 6 digit code that many 'color-picker' websites provide:

```
color = '#008141'
```

Now, we are ready to plot the function on our axes *ax* and add a label to the graph, that shows up in our legend.

```
ax.plot(X,Y,c=color,label='Integral')
```

We can now label our axis, plot the legend and limit the range of values to make the plot look more polished. Note, python supports LaTeX commands by default.

```
ax.set_xlim([-4,10])
ax.set_ylim([0,1])
ax.set_xlabel(r'$x$')
ax.set_ylabel(r'$f(x)$')
ax.legend()
```

Finally, we save the plot as a (high-resolution) pdf but many other formats like png are also supported.

```
fig.savefig('simpleExample.pdf',dpi=450)
```

(b) **Advanced example**

This example will complicate things a bit by including a parameter in the previous integral that has a functional dependence on the integration variable $a(y)$ ($b = 1$):

$$f(x) = \int_{-4}^{x} \mathrm{d}y \frac{y^3 a(y) - y^2}{y^7 + y^{-1}}. \tag{9}$$

and plot it for varying values of the functional parameter: $a(y) = y^2 + 1$, $a(y) = 4$, $a(y) = y^2 - y^{-1}$ and $a(y) = (y + 1)^2$ (see Figure 2).

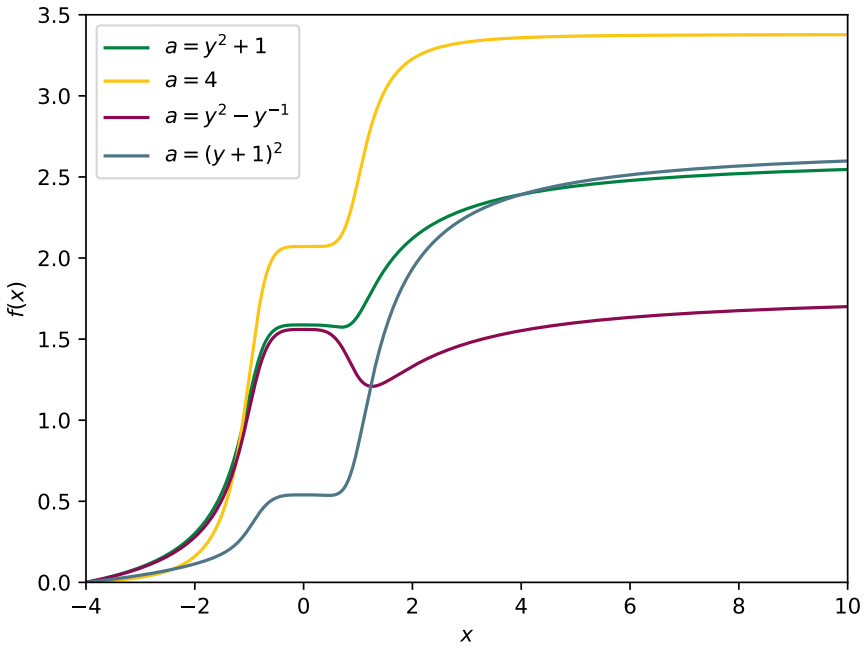

Figure 2: Our plot of a parametrized integral.

We start by importing the required modules and defining our functions. The definition of $f(x)$ only needs the slight modification $a \to a(x)$ to accommodate for a being a function now.

```
import scipy.integrate as integrate
import numpy as np
import matplotlib.pylab as plt

def f(x, a, b):
    return (a(x)*x**3.-x**2.)/(x**7.+b*x**(-1.))

def IntGeneric(function, lower, upper, parameters):
    result, error = integrate.quad(function, lower, upper, args=
    parameters)
    return result
```

Now, we initiate the lists dealing with the varying definitions of $a(x)$, our labels for the final plot, the different colors corresponding to different models and an array with our y-values.

```
A = [
    lambda x: x**2.+1.,
```

```
    lambda x: 4.,
    lambda x: x**2.-x**(-1.),
    lambda x: (x+1.)**(2.),
    ]

Labels = [
        r'$a=y^2+1$',
        r'$a=4$',
        r'$a=y^2-y^{-1}$',
        r'$a=(y+1)^{2}$',
        ]

Colors = ['#008141','#fdc513','#8b0a50','#4f7687']

X = np.linspace(-4.,10.,1e3)
```

To compute our y-values for varying models for $a(y)$, we need two loops this time (x-values, models of a) and therefore initialize two individual lists Y and YY, which we combine to form the single list YY, i.e.

```
YY = []
for a in A:
    Y = []
    for x in X:
        parameters = (a,1.)
        y = IntGeneric(f,-4.,x,parameters)
        Y.append(y)
    YY.append(Y)
```

We initiate our plot by defining our figure and axes:

```
fig = plt.figure()
ax = plt.axes()
```

Now we use a single loop to unpack our y-values and label as well as color our individual line plots:

```
YY = []
for a in A:
    Y = []
    for x in X:
        parameters = (a,1.)
        y = IntGeneric(f,-4.,x,parameters)
        Y.append(y)
    YY.append(Y)
```

Analogously to the previous example, we set our labels & limits and include the legend. Finally, we can save the plot.

```
ax.set_xlim([-4,10])
ax.set_ylim([0,3.5])
ax.set_xlabel(r'$x$')
ax.set_ylabel(r'$f(x)$')
ax.legend()
fig.savefig('advancedExample.pdf',dpi=450)
```