L3: Nonlinear equations

1 Brief Introduction

Nonlinear equations appear in many applications. In general, non-linear equations cannot be solved analytically, so numerical methods are used instead.

- Here we will study so-called *iterative methods* for solving non-linear equations. The basic idea behind these methods is that you first *guess* a/the root or a range where you think a root lies, and then successively improve the guess until a sufficiently accurate approximate root is found. Each "turn" in such an algorithm, where the guess is improved, is called an *iteration*.
- There are many different methods for solving non-linear equations numerically. Two simple, but common, iterative methods are the *Newton-Raphson method* and the *bisection method*. Here we will take a closer look at how these two methods work.

Start by downloading the Python files for Section 2 of this lab: **Newtondemo.py** and **Bisectiondemo.py**.

Note! If you are using an IDE, you may need to set it up so that interactive figures can be created. For example in Spyder: Preferences → IPython Console → Graphics → Backend → Automatic.

2 The Newton-Raphson method and the bisection method

We will begin by studying the non-linear equation

$$x^2 = 4\sin(x) + 1. \tag{1}$$

The iterative methods we will examine are used to find zeros (or roots) of a function, so the equation must be rewritten in the form $f(x) = 0$. In this case we get

$$f(x) = x^2 - 4\sin(x) - 1. \tag{2}$$

1. Run the program `Newtondemo` and experiment with different choices for the starting guess. Choose the starting guess so that in some cases you find one zero of the function and in some cases the other. Pay attention to the impact of the starting guess on the number of iterations. Once Newton-Raphson has gotten "close enough" to a zero, the error decreases very quickly. Can you identify any rule of how the number of correct digits (i.e., the number of digits that is equal to zero in the error) increases (roughly) in each iteration?
2. The Newton-Raphson method is based on the next guess being given by the intersection of the tangent at the current point with $y = 0$. Try to see it in the figure, i.e. how the Newton-Raphson method works in principle. This can lead to problems when the tangent points far away from the zero. Test starting guesses around $x = 1$. What happens?

3. Run the program `Bisectiondemo` and experiment with different choices for the start interval. Choose the starting range so that in some cases you find one zero and in some cases the other. Note that one can choose large intervals, provided that the interval covers only one zero. Newton-Raphson moved very quickly towards zero once it got "close enough". But what about the bisection method? Which method seems to be the faster?

4. The bisection method is based on successive halving of the interval in such a way that the function values in the interval limits always have different signs. That way there always exists at least one root in the interval, whose length is halved in each iteration. Why doesn't the bisection method work if the starting range includes two zeros? Or no zero? What would happen if a starting interval with three roots was chosen?

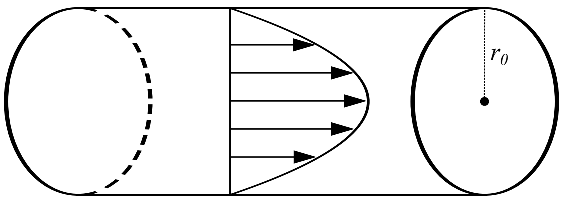
3 SciPy Functions for Solving Nonlinear Equations

When solving non-linear equations in Python, you can use functions from SciPy. This part of the lab is an introduction to one of these functions, `brentq`, intended for nonlinear equations with one unknown (scalar equations). It is imported according to:

```python
from scipy.optimize import brentq
```

The function is based on an algorithm that combines several methods, i.a. the bisection method and a variant of the Newton-Raphson method (which is called the secant method, and which does not require access to the derivative of the function for which the zero point is sought). Like the bisection method, `brentq` requires a starting range where the function changes sign.

We will start here from the problem posed in the laboratory about integration, namely the speed of a flowing liquid in a pipe with radius r_0, see Figure 1. The velocity v



Figur 1: Velocity distribution in cylindrical pipe with radius r_0.

at distance r from the center point is described by the mathematical model

$$v(r) = 2(1 - \frac{r}{r_0})^{1/6}. \tag{3}$$

We want to know at what radius r the velocity is c, i.e. the problem is to find the value of r such that $v(r) = c$. This is a non-linear equation which is conveniently solved using `brentq`.

5. The function `brentq` is used to find roots, so the first step is to rewrite the equation as a root-finding problem. That is find a function $f(r)$ such that the solution to the equation $f(r) = 0$ is the same as the solution to $v(r) = c$.

6. When solving scalar nonlinear equations, it is often useful to examine the problem graphically, in order to get an idea for how many zeros there seem to be and what might be a good starting guess or starting range. Start by defining $f(r)$ as a Python function, using the structure

```
def f(r):
  return...
```

   Implement the function so that it can be called with `r` as a NumPy vector and return $f(r)$ for all values in the vector. Use $r_0 = 3$ and $c = 1.8$. Then plot the function with, for example, `matplotlib`. Keep in mind that the radius r cannot be less than 0 or greater than r_0, so we are interested in $f(r)$ on the interval $r \in [0, r_0]$. How many zeros seem to be there?

7. Solve the non-linear equation using `brentq` through the call

```
root = brentq(f,xl,xr)
```

   where `f` is the Python function and the last two arguments define the starting range. The root is stored in the variable `root`. Think about what constitutes a suitable starting interval, based on the figure. Verify that the function has solved the equation correctly by studying the figure or calling `f(root)`.

8. In this case, it would be good if we could easily vary the parameters r_0 and c when using `brentq`. These parameters must then be passed into `brentq` and in turn further into `f`. Modify the function `f` so that r_0 and c are passed in as arguments; the function structure becomes

```
def f(r,r0,c):
  return...
```

   Try plotting the function as before with some different choices for r_0 and c so that you know it works. To use `brentq` when the function `f` has more than one argument, the keyword `args` is used and the call becomes

```
root = brentq(f,xl,xr,args=(r0,c))
```

   It is now easy to change r_0 and c to other values, then plot and find the zero again. Do that for some different choices of r_0 and c.

9. Finally, set $r_0 = 3$ and $c = 4$ and try to solve the equation using `brentq`. What is happening and why? Study the figure.

There are several functions in SciPy for solving non-linear equations, see the online documentation at `https://docs.scipy.org/doc/scipy/reference/optimize.html`. If you are inspired to do more, you can substitute `brentq` in your code with another SciPy function, for example `toms748`. Read the documentation to understand how to call the various functions.

4 Calculation of proportion of water vapor

Now you have finished your studies and arrive at your first workplace. There you are placed in a group that works with a chemical engineering process where water vapor is heated to such high temperatures that water molecules begin to dissociate (separate) into oxygen (O_2) and hydrogen (H_2) according to the process

$$H_2O \rightleftharpoons H_2 + \frac{1}{2}O_2. \tag{4}$$

The reaction takes place under the pressure $p = 3$ atmospheres and with the equilibrium constant $K = 0.05$. Your colleagues and you discuss what percentage of the water molecules will disintegrate. One of the colleagues browses the literature and finds a mathematical model for how the proportion of molecules that dissociate, x, depends on the pressure and the equilibrium constant of the reaction. According to the model, the relationship is

$$K = \frac{x}{1-x}\sqrt{\frac{2p}{2+x}}. \tag{5}$$

The problem is that here it is not possible to solve for x.[1]

10. Write a Python program that answers the above problem, ie. solves the non-linear equation (5) with $p = 3$ and $K = 0.05$. Use the previous part of the lab as a template. Rewrite it as a root-finding problem, plot the function and call `brentq`. Think about what x stands for, and what interval would therefore be physically reasonable, when plotting and choosing starting intervals.

[1]The problem formulation is based on problem 6.14 in Steven C. Chapra, *Applied Numerical Methods with MATLAB for Engineers and Scientists, 4th Edition*, McGraw-Hill, 2018, p 191.