Python exercises for Chapter 2

1. Write a script containing a function `bisection` with

   - Input: function f, the extremes of the interval $[a,b]$, the tolerance, tol, for the length of the subintervals, and the maximum number of iterations, $maxiter$.
   - Output: the approximated zero, and the number of performed iterations.

   Use it to approximate the three roots of $f(x) = x^3 - 2x^2 + 1$. Follow these steps:

   (a) Plot f in the interval $(-1,2)$ to locate subintervals where f changes of sign (do not save the figure to a file).

   (b) Use your function `bisection.py` in these subintervals ($tol = 1.e-9$, $maxiter = 200$).

   (c) Print the roots **in the same order** than below using the following format:
   `print('{:.10e}'.format(sol), file=results)`

Solution:
-6.1803398933e-01
1.0000000000e+00
1.6180339884e+00

Note: Your result may be slightly different due to the election of the initial intervals.

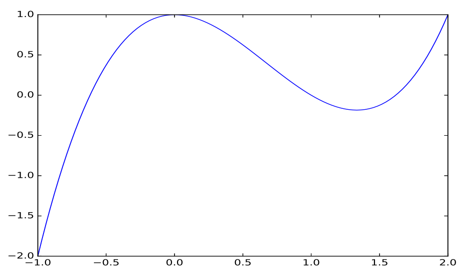

Figure : Exercises 1, 2, and 3

2. Write a script containing a function `newton` with

   - Input: function f, its differential df, the initial value x_0, the tolerance for the absolute difference between two iterations, tol, and the maximum number of iterations, $maxiter$.
   - Output: the approximated zero, and the number of performed iterations.

   Use it to approximate the three roots of $f(x) = x^3 - 2x^2 + 1$. Follow these steps:

   (a) Plot f in the interval $(-1,2)$ to locate the initial values for each root (do not save the figure to a file).

   (b) Use your function `newton.py` in the initial values ($tol = 1.e-9$, $maxiter = 200$).

   (c) Print the roots **in the same order** than below using the following format:
   `print('{:.10e}'.format(sol), file=results)`

Solution:
-6.1803398875e-01
1.0000000000e+00
1.6180339887e+00

Note: Your result may be slightly different due to the election of the initial guesses. The plot is the same than in Exercise 1.

3. Write a script containing a function `secant` with

   - Input: function f, the initial values x_0, x_1, the tolerance for the absolute difference between two iterations, tol, and the maximum number of iterations, $maxiter$.
   - Output: the approximated zero, and the number of performed iterations.

   Use it to approximate the three roots of $f(x) = x^3 - 2x^2 + 1$. Follow these steps:

   (a) Plot f in the interval $(-1, 2)$ to locate the initial values (do not save the figure to a file).

   (b) Use your function `secant.py` in these initial values ($tol = 1.e-9$, $maxiter = 200$).

   (c) Print the roots **in the same order** than below using the following format:
   `print('{:.10e}'.format(sol), file=results)`

Solution:
-6.1803398875e-01
1.0000000000e+00
1.6180339887e+00

Note: Your result may be slightly different due to the election of the initial guesses. The plot is the same than in Exercise 1.

4. Write a script containing a function `fixedpoint` with

   - Input: function f, the initial value x_0, the tolerance for the absolute difference between two iterations, tol, and the maximum number of iterations, $maxiter$.
   - Output: the approximated zero, and the number of performed iterations.

   Use it to approximate the **zero** of $f(x) = e^{-x} - x$. Follow these steps:

   (a) Plot f in the interval $(-1, 1)$ to locate the initial value (do not save the figure to a file).

   (b) Use your function `fixedpoint.py` in this initial value ($tol = 1.e-9$, $maxiter = 200$).

   (c) Print the root using the following format:
   `print('{:.10e}'.format(sol), file=results)`

Solution:
5.6714329012e-01

Note: Your result may be slightly different due to the election of the initial guess.

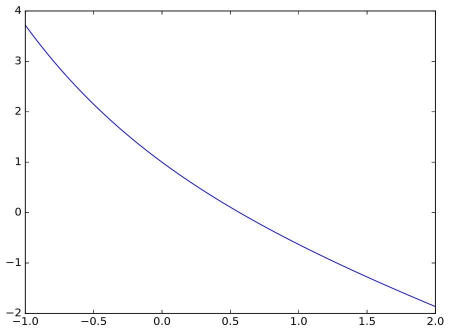

Figure : Exercise 4

5. Find all the non-negative solutions of $\sin x - 0.1x = 0$ using Newton's method. Follow these steps:

   (a) Plot the function in the interval $(-1, 20)$ to locate the initial values (do not save the figure to a file).

   (b) Use the function `newton` of the module `scipy.optimize` in the initial values ($tol = 1.e-10$, $maxiter = 100$).

   (c) Print the root, using the following format:
   `print('{:.10e}'.format(sol), file=results)`

Solution:
0.0000000000e+00
2.8523418945e+00
7.0681743581e+00
8.4232039324e+00

Note: Your result may be slightly different due to the election of the initial guesses.

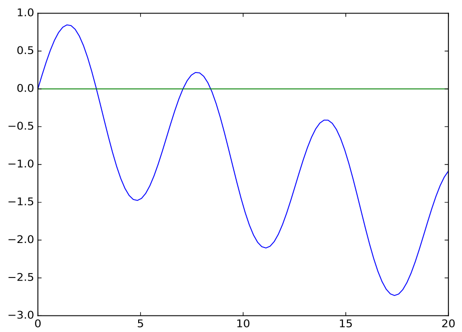

Figure : Exercise 5

6. Find the smallest positive zero of $f(x) = \cosh(x)\cos(x) - 1$ using the secant method provided by the function `newton` of the module `scipy.optimize`. Give the value of the approximated zero using the following format: `print('{:.10e}'.format(sol), file=results)`

Solution:
4.7300407449e+00

Note: Your result may be slightly different due to the election of the initial guess.

7. Approximate the value of $\sqrt[3]{75}$ using the bisection method provided by the function `bisect` of the module `scipy.optimize`. Use the initial interval $[3,5]$, $xtol = 1e-16$, and $maxiter = 100$. Give the full information output of `bisect`.

Solution:
converged: True
flag: 'converged'
function_calls: 51
iterations: 49
root: 4.217163326508743

8. For solving the equation $x + \ln x = 0$ by the fixed point method, we consider the following functions:

$$(i)\ f_1(x) = e^{-x} \qquad (ii)\ f_2(x) = \frac{x+e^{-x}}{2} \qquad (iii)\ f_3(x) = -\ln x$$

   (a) Check graphically that all these functions have the same fixed point (do not save the figure to a file).
   (b) Use your function `fixedpoint` with $x_0 = 0.5$, $tol = 1.e-16$, and $maxiter = 200$ to investigate how many iterations are needed to approximate the fixed point with each function f_1, f_2 and f_3.
   (c) For f_3, the method is not convergent. Modify your function `fixedpoint` to get the output `-1` for the number of iterations (in this way, we signal an error).

Print the number of iterations performed for each function using the following format:
`print('{:d}'.format(numiter), file=results)`

Solution:
63
24
-1

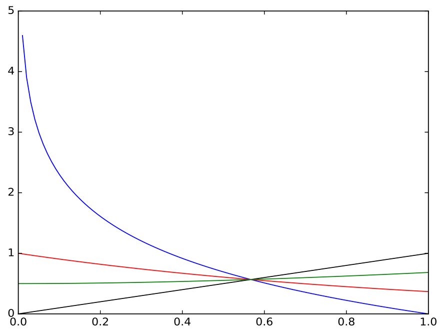

Figure : Exercise 8

9. Determine the coordinates of the two points where the circles $(x-2)^2+y^2=4$ and $x^2+(y-3)^2=4$ intersect. Start by estimating the locations of the points from a plot (do not save it to a file), and then use the function `newton` of the module `scipy.optimize` ($tol = 1.e-10$, $maxiter = 100$) to estimate the coordinates. Use the following format for each root (x,y):
`print('{:.10e}'.format(x), '{:.10e}'.format(y), file=results)`

Solution:
2.7942330788e-01 1.0196155386e+00
1.7205766921e+00 1.9803844614e+00

Note: Your result may be slightly different due to the election of the initial guesses.

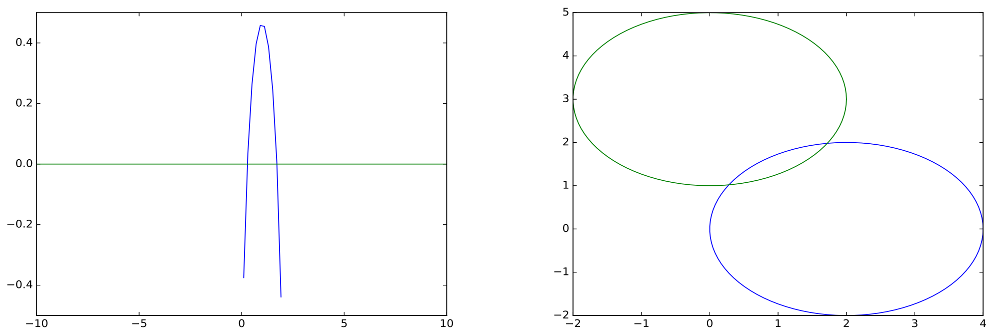

Figure : Exercise 9

10. The equations

$$\sin(x)+3\cos(x)-2=0,$$
$$\cos(x)-3\sin(y)=0,$$

have a solution in the vicinity of the point $(1,1)$.

(a) Use the function `newton` of the module `scipy.optimize` ($tol = 1.e-10$, $maxiter = 100$) to estimate the solution x of the first equation, and then compute the solution y of the second.

(b) Use the the function `fsolve` of the module `scipy.optimize` ($tol = 1.e-10$) to directly estimate the solution of the whole system.

Print both solutions using the format
`print('{:.10e}'.format(x), '{:.10e}'.format(y), file=results)`

Solution:
1.2078276782e+00 1.1862838300e-01
1.2078276782e+00 1.1862838300e-01