Computer Methods in Engineerings
Exercise on the bracket method

Problem 1

Bracket search methods are described in the notes.

a) Implement the bracket search method to locate a minimum of the function $f(x) = x^2 - sin(x)$ on the interval $(a, b) = (0, 1)$. Split the interval (a, b) into four subintervals of length $\Delta x = (b - a)/4.0$ and initially select α and β by $\alpha = a + \Delta x, \beta = b - \Delta x$. Experiment with different values of Δx.

Solution:

Below is a listing of a simple implementation of the bracketing method. The number of iterations required was 10000 before an acceptable solution was found. See also 1.

```
import numpy as np
import matplotlib.pyplot as plt
from math import *

class BracketMinimizer:
  def __init__(self, fn_func, fl_a=0.0, fl_b=1.0, fl_tol=1e-12, ...
      i_max_iter=20000):
    self.fn_func = fn_func
    self.fl_a = fl_a
    self.fl_b = fl_b
    self.fl_tol = fl_tol
    self.i_max_iter = i_max_iter
    self.fl_x0 = None
    self.fl_err = None
    self.i_iterations = None

  def minimize(self):
    fl_dx = (self.fl_b - self.fl_a) / 4.0
    for i_iter in range(self.i_max_iter):
      fl_alpha = self.fl_a + fl_dx
      fl_beta = self.fl_b - fl_dx
      if self.fn_func(fl_alpha) < self.fn_func(fl_beta):
        self.fl_b = fl_beta
      else:
        self.fl_a = fl_alpha
      fl_err = abs(self.fl_b - self.fl_a)
      fl_dx = fl_err / 4.0
      if fl_err < self.fl_tol:
        self.fl_x0 = (self.fl_b + self.fl_a) / 2
        self.fl_err = fl_err
        self.i_iterations = i_iter
        return self.fl_x0
```

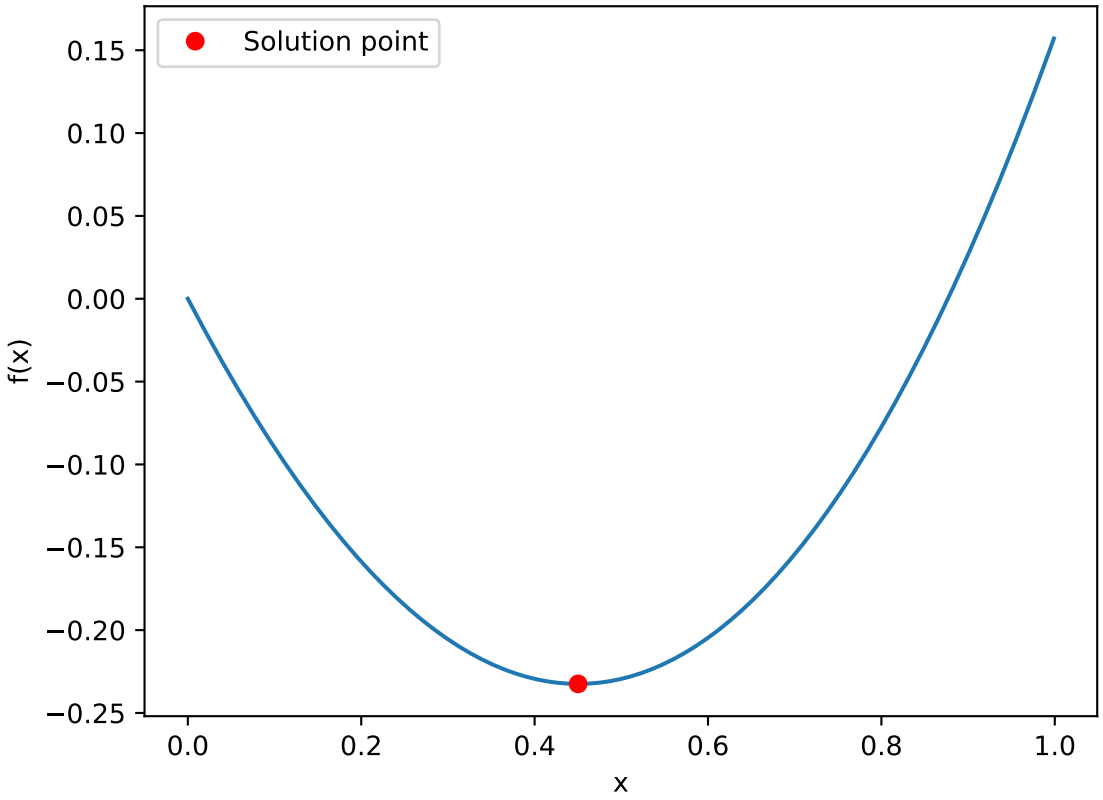

Figure 1: Graph of $f(x) = x^2 - sin(x)$ with minimum point found by simple bracketing.

.

```
    print("No convergence!")
    return None

  def plot(self, s_filename='bracket.pdf'):
    arr_xx = np.arange(0, 1.0, 0.001)
    arr_ff = self.fn_func(arr_xx)
    plt.plot(arr_xx, arr_ff)
    if self.fl_x0 is not None:
      plt.plot(self.fl_x0, self.fn_func(self.fl_x0), 'ro', ...
          label='Solution point')
    plt.legend()
    plt.xlabel('x')
    plt.ylabel('f(x)')
    plt.savefig(s_filename)
    plt.show()

# Create an instance of the class and use its methods
tBracketMinimizer=BracketMinimizer(lambda x: x**2-np.sin(x))
tBracketMinimizer.minimize()
tBracketMinimizer.plot('bracket.pdf')
```

b) Implement the golden ration method to locate a minimum of the function $f(x) = x^2 - sin(x)$ on the interval $(a, b) = (0, 1)$. Compare the number of iterations used

to find a solution with the number of iterations used in the previous point.

Solution:
Below is a listing of an implementation of the Golden ratio method. The number of iterations required to get an acceptable solution was 58. Compared to the bracketing method used in the previous point, the Golden ratio method is faster by a factor of 96/58. See figure 2 for a plot of the solution.

```
import numpy as np
import matplotlib.pylab as pl
from math import *

def f(x) :

  rval = x*x-np.sin(x)
  return rval

r = (sqrt(5)-1)/2  # inverse golden ratio
n = 100;           # max number of iterations
eps = 1e-12;       # tolerance
a = np.zeros([n])  # start of interval
b = np.zeros([n])  # end of interval
alpha = np.zeros([n])
beta = np.zeros([n])
a[0] = 0;          # lower limit of the interval
b[0] = 1;          # upper limit of the interval
nx=100
x = np.linspace(a[0],b[0],nx);   # intervall x to plot f(x)

for i in range(0,n):
   alpha[i] = a[i]+(1-r)*(b[i]-a[i]);
   beta[i]  = b[i]-(1-r)*(b[i]-a[i]);
   if f(alpha[i]) ≤ f(beta[i]) :
       a[i+1] = a[i];
       b[i+1] = beta[i];
   else :
       a[i+1] = alpha[i];
       b[i+1] = b[i];

   if abs(b[i]-a[i]) < eps :
       x0 = (a[i]+b[i])/2;
       print('Convergence reached after', i, ' iterations.')
       print('x0 =    ', x0)
       print('f(x0) = ', f(x0))
       break

#Plot f(x) and solution point
dx=0.01
xx = np.arange(0,1.0,dx)
ff = f(xx)
pl.plot(xx,ff)
pl.plot(x0,f(x0),'ro',label='Solution point')
pl.legend()
pl.xlabel('x')
pl.ylabel('f(x)')
pl.savefig('golden.pdf')
pl.show()
```

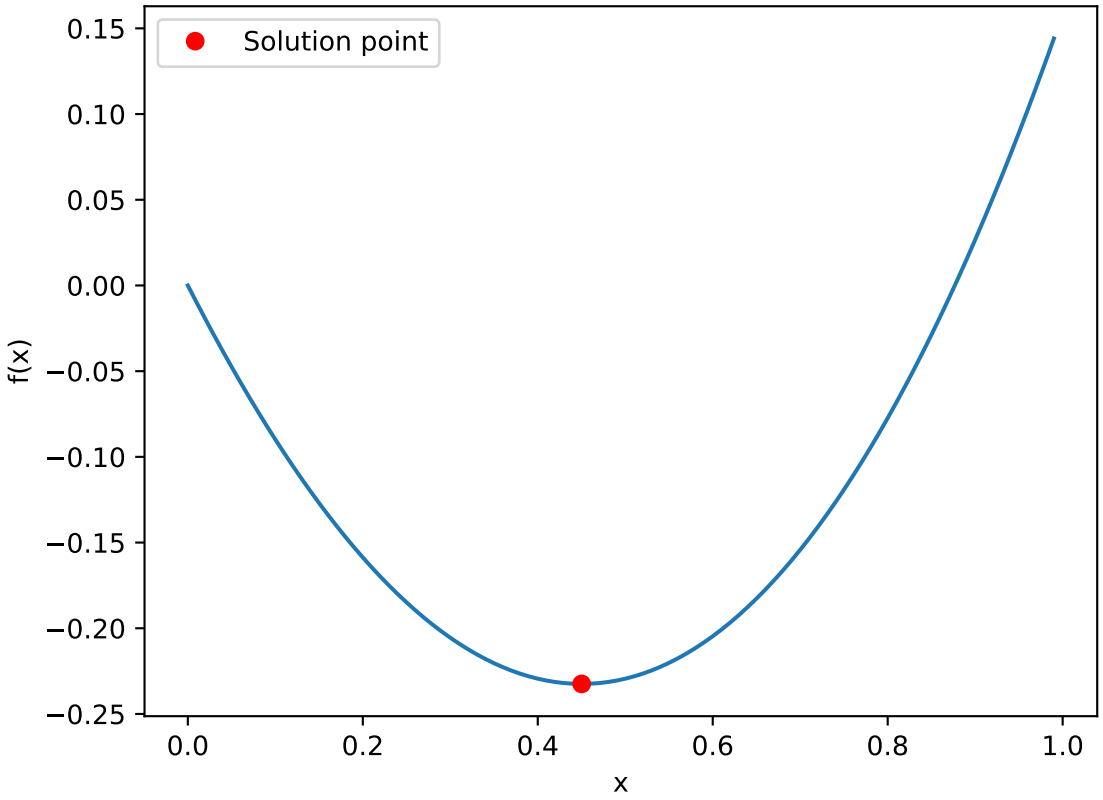

Figure 2: Graph of $f(x) = x^2 - sin(x)$ with minimum point found by the golden ratio method.

.

Problem 2

Assume you are going to build a pipeline to transport CO2 from a capture site to an offshore storage site. The pipeline has a length of 1000 km, and the flow rate of CO2 is 0.3 m^3/s. The viscosity of CO2 is approximately 15 Pa s at room temperature. The cost of the pipe is 10 dollar per m^2 of pipe surface, and the cost of operating the pipeline is 1E-11 dollar per pascal per m^3 of gas transported. The income from selling the transport service is 1 dollar per m^3 of gas transported. Here, we disregard the expansion of the gas due to pressure changes, and assume that the flow is laminar. We will thus use the Hagen-Poiseulle equation to calculate the pressure drop in the pipe. A code for calculating the profit from the pipeline as a function of the radius of the pipe is given as follows:

```
import numpy as np
import math as m
import matplotlib.pyplot as plt


class CpipedGas:
    def __init__(self):
        self.pricePerPascalperCubed=1E-11 #dollar per pascal per cube of gas
        self.materialPrice=10 #dollar per m2
        self.flowRate=0.3 #m3 per second
```

```
        self.length=1E6 #1000 km
        self.viscosity= 15 # Pa s, approximately viscosity of CO2 at room ...
            temperature
        self.payment=1 # dollar per cube

    def costPipe(self,radius):
        return m.pi*radius**2*self.length*self.materialPrice

    def costUse(self,radius):
        # Using Hagen-Poiseulle to calculate pressure drop
        deltaP=8*self.viscosity*self.length*self.flowRate/(m.pi*radius**4)
        return deltaP*self.pricePerPascalperCubed

    def profit(self,radius,time):
        return ...
            self.payment*self.flowRate*time-self.costUse(radius)*self.flowRate*time-self.costPi
```

Use this code to find the optimal radius of the pipe using the bracket method, when we assume that we will operate the pipe for 1E8 seconds (a bit more than 3 years).

Solution:

We can find the optimal radius of the pipe by using the bracket method implemented in the class CpipedGas. The code below shows how to create an instance of the class and use its methods to find the optimal radius. The optimal radius is found to be approximately 0.2457 m, with a profit of approximately 2.72E7 dollars, and it took 18 iterations to converge to this solution.

```
    def bracket(self,time):
        from random import random
        #Maximum number of iterations
        self.iN = 100

        #Error tolerance
        self.feps = 1e-3

        #Storing values during run for later plotting
        self.afA=np.zeros(self.iN)
        self.afB=np.zeros(self.iN)
        self.afAlpha=np.zeros(self.iN)
        self.afBeta=np.zeros(self.iN)

        #Starting values for our interval [a,b] and internal points alpha ...
            and beta
        self.afA[0] = 0.1
        self.afB[0] = 0.4

        # This is an iterative algorithm, and we iterate over ii
        ii=0
        while(ii<self.iN and (self.afB[ii]-self.afA[ii])>self.feps):
            #Picking two random numbers between a and b
            self.afAlpha[ii]=(self.afB[ii]-self.afA[ii])*random()+self.afA[ii]
            self.afBeta[ii]=(self.afB[ii]-self.afA[ii])*random()+self.afA[ii]
            #If beta is larger than alpha, then switch the numbers
            if self.afBeta[ii]<self.afAlpha[ii]:
                fTemp=self.afAlpha[ii]
                self.afAlpha[ii]=self.afBeta[ii]
                self.afBeta[ii]=fTemp
```

```
            # Take negative of profit since we are minimizing
            falpha=-self.profit(self.afAlpha[ii],time)
            fbeta=-self.profit(self.afBeta[ii],time)
            # Iterate forward
            ii+=1
            # Reduce the width of the brackets
            if falpha<fbeta:
                self.afA[ii]=self.afA[ii-1]
                self.afB[ii]=self.afBeta[ii-1]
            else:
                self.afA[ii]=self.afAlpha[ii-1]
                self.afB[ii]=self.afB[ii-1]

        #print('Number of object function evaluations: ',2*ii)
        return (self.afA[ii]+self.afB[ii])/2, ...
            self.profit((self.afA[ii]+self.afB[ii])/2,time), ii

# Create an instance of the class and use its methods
tpipedGas=CpipedGas()
#tpipedGas.plot(1E8) #plot profit
#print(1E8/(60*60*24*365)) #convert seconds to years
sol,val,iter=tpipedGas.bracket(1E8)
print('Optimal radius (m), profit ($), iterations: ',sol,val,iter)
```