**Title page:** ***Algorithms for unconstrained nonlinear optimization. Direct methods***

FEDERAL STATE AUTONOMOUS EDUCATIONAL INSTITUTION
OF HIGHER EDUCATION
ITMO UNIVERSITY

Report
on the practical task No. X
"***Algorithms for unconstrained nonlinear optimization. Direct methods***

***Goal***

"

Performed by
*Abizer Safdari*
*J4134c*
Accepted by
Dr Petr Chunaev

St. Petersburg
2020

**Goal**

The use of direct methods (one-dimensional methods of exhaustive search, dichotomy, golden section search; multidimensional methods of exhaustive search, Gauss, Nelder-Mead) in the tasks of unconstrained nonlinear

**Formulation of the problem**

***I.*** *Use the one-dimensional methods of exhaustive search, dichotomy and golden section search to find an approximate (with precision $\varepsilon = 0.001$) solution $x$: $f(x) \to min$ for the following functions and domains:*

1. $f(x) = x^3, x \in [0, 1]$;
2. $f(x) = |x - 0.2|, x \in [0, 1]$;
3. $f(x) = x\, sin \frac{1}{x}, x \in [0.01, 1]$.

*Calculate the number of $f$-calculations and the number of iterations performed in each method and analyze the results. Explain differences (if any) in the results obtained.*

***II.*** *Generate random numbers $\alpha \in (0,1)$ and $\beta \in (0,1)$. Furthermore, generate the noisy data $\{x_k, y_k\}$, where $k = 0, \ldots, 100$, according to the following rule:*

$$y_k = \alpha x_k + \beta + \delta_k, \quad x_k = \frac{k}{100},$$

*where $\delta_k \sim N(0,1)$ are values of a random variable with standard normal distribution. Approximate the data by the following linear and rational functions:*

1. $F(x, a, b) = ax + b$ *(linear approximant),*
2. $F(x, a, b) = \frac{a}{1+bx}$ *(rational approximant),*

*by means of least squares through the numerical minimization (with precision $\varepsilon = 0.001$) of the following function:*

$$D(a, b) = \sum_{k=0}^{100} (F(x_k, a, b) - y_k)^2.$$

*To solve the minimization problem, use the methods of exhaustive search, Gauss and Nelder-Mead. If necessary, set the initial approximations and other parameters of the methods. Visualize the data and the approximants obtained in a plot* ***separately for each type of approximant****. Analyze the results obtained (in terms of number of iterations, precision, number of function evaluations, etc.).*

## Brief theoretical part

- Optimization is the process of obtaining the best result under given circumstances.
- For discrete problems in which no efficient solution method is known, it might be necessary to test each possibility sequentially in order to determine if it is the solution. Such exhaustive examination of all possibilities is known as exhaustive search, direct search, or the "brute force" method.
- The algorithm applies to any continuous function f(x) on an interval [a,b] where the value of the function f(x) changes sign from a to b. The idea is simple: divide the interval in two, a solution must exist within one subinterval, select the subinterval where the sign of f(x) changes and repeat.
- The **golden-section search** is a technique for finding an extremum (minimum or maximum) of a function inside a specified interval. For a strictly unimodal function with an extremum inside the interval, it will find that extremum, while for an interval containing multiple extrema (possibly including the interval boundaries), it will converge to one of them. If the only extremum on the interval is on a boundary of the interval, it will converge to that boundary point. The method operates by successively narrowing the range of values on the specified interval, which makes it relatively slow, but very robust.
- The Gauss method is an iterative algorithm to solve nonlinear least squares problems. "Iterative" means it uses a series of calculations (based on guesses for x-values) to find the solution. The Gauss is usually used to find the best fit theoretical model although it could also be used to locate a single point.
- The Nelder–Mead method is a commonly applied numerical method used to find the minimum or maximum of an objective function in a multidimensional space. It is a *direct search method* (based on function comparison) and is often applied to nonlinear optimization problems for which derivatives may not be known. However, the Nelder–Mead technique is a heuristic search method that can converge to non-stationary points[1] on problems that can be solved by alternative methods

**Results**

| Function | Method | Min(f(x)) | X:Min(f(x)) | Iteration | F-calculation |
|---|---|---|---|---|---|
| $x^3$ | Exhaustive | 0.00 | 0.00 | 999 | 1000 |
| | Dichotomy | 0.00 | 0.000294 | 11 | 22 |
| | Golden-search | 0.00 | 0.001553 | 12 | 13 |
| $\|x - 0.2\|$ | Exhaustive | 0.00 | 0.20 | 999 | 1000 |
| | Dichotomy | 0.000019 | 0.199981 | 11 | 22 |
| | Golden-search | 0.000073 | 0.200073 | 12 | 13 |
| $\mathbf{X*sin}(\frac{1}{x})$ | Exhaustive | -0.21723 | 0.22285 | 999 | 1000 |
| | Dichotomy | -0.217233 | 0.222482 | 11 | 22 |
| | Golden-search | -0.217230 | 0.222271 | 12 | 30 |

This table show that exhaustive search spend more iterations, calls of f(x) and time than other methods, but this algorithm can be better in some situation, because it can find more than one local minimum unlike dichotomy or golden section search.

If we compare dichotomy and golden section search, we will see that dichotomy use more calls of function, but less iterations. So if we have "light" function, it is better to use dichotomy, else golden section search

| Approximation | Approximation | Minimization method | Iteration | f-calculation |
|---|---|---|---|---|
| Linear | ax+b | Exhaustive | 4000 | 4000 |
| | | Gauss | 7 | 182 |
| | | Nelder mead | 65 | 13 |
| Rational | a/(1+bx) | Exhaustive | 4000 | 4000 |
| | | Gauss | 2 | 52 |
| | | Nelder mead | 60 | 12 |

**Table 2:** Number of iterations and functional calculations performed in each multi dimensional method.

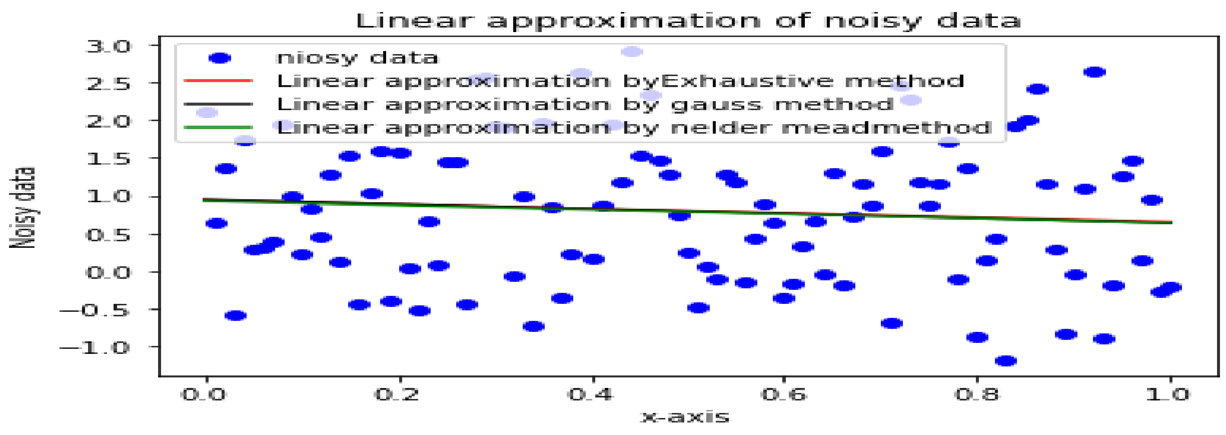

**Figure 1:** The linear approximation of the noisy data using exhaustive, Gauss and Nelder-mead method.

All methods give similar results, which correlate with init data. Method of exhaustive search spends too many time

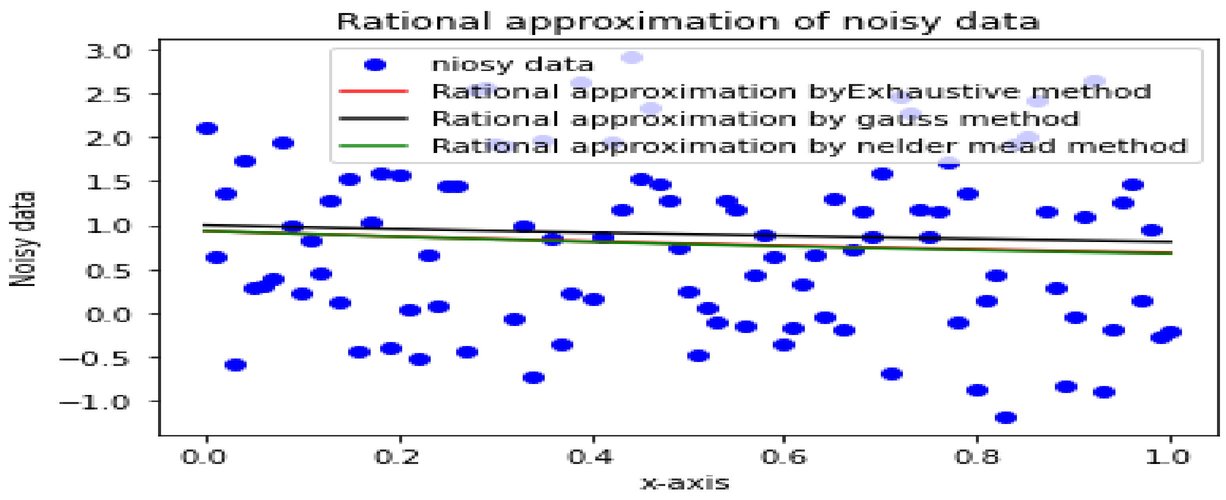

**Figure 2:** The rational approximation of the noisy data using exhaustive, Gauss and Nelder-mead method.

Same situation as linear regression

Nelder-mead is better than Gauss as the it reaches required precision with less function calculations

**Appendix**

```
"""
Created on Wed Oct  5 12:15:48 2020

@author: abizer
"""
import numpy as np
import matplotlib.pyplot as plt
# defining functions
def y1(x):
    return x**3
def y2(x):
    return abs(x-0.2)
def y3(x):
    return x*np.sin(1/x)

# one dimensional exhaustive method for optimized minization
def exhaustive(y, a,b,n):
    h=(b-a)/n
    x=np.arange(a, b+h, h )
    xm=[]
    i=0
    while i <n-1:
        if y(x[i])>y(x[i+1]) and y(x[i+1])<y(x[i+2]):
            xm.append(x[i+1])
        i=i+1
    if len(xm)==0:
        if y(x[0])>y(x[-1]):
            xm.append(x[-1])
        else:
            xm.append(x[0])
    ym=y(np.array(xm))
    ymin=min(ym)
    xmin=xm[np.where(ym==ymin)[0][0]]
    return ymin, xmin, i, n
y1_min, x1_min , i1, j1 = exhaustive(y1, 0, 1, 1000)
y2_min, x2_min , i2 , j2= exhaustive(y2, 0, 1, 1000)
y3_min, x3_min , i3, j3 = exhaustive(y3, 0.01, 1, 1000)
print('The minimum of function x^3 in the given domain using \
 exhaustive search is %f found at :x= %f' %(y1_min, x1_min) )
print('The minimum of function |x-0.2| in the given domain using \
 exhaustive search is %f found at :x= %f' %(y2_min, x2_min) )
print('The minimum of function x*sin(1/x) in the given domain using\
 exhaustive search is %f found at :x= %f' %(y3_min, x3_min) )
```

```
print('The number of f-calulations and iterations performed in \
 exhaustive method are %d, %d, repectively' %(i1, j1))

# Dichotomy method for optimized minization
def dichotomy(y,a,b,err):
    d=0.0001
    x1=(a+b-d)/2
    x2=(a+b+d)/2
    i=0
    j=0
    while abs(a-b)>err :
        if y(x1)<=y(x2):
            b=x2
        else:
            a=x1
        x1=(a+b-d)/2
        x2=(a+b+d)/2
        i=i+1
        j=j+2
    xmin=(a+b)/2
    return y(xmin),xmin, i, j
y1_min, x1_min , i1, j1 = dichotomy(y1, 0, 1, 0.001)
y2_min, x2_min , i2 , j2= dichotomy(y2, 0, 1, 0.001)
y3_min, x3_min , i3, j3 = dichotomy(y3, 0.01, 1, 0.001)
print('The minimum of function x^3 in the given domain using \
 dichotomy method is %f found at :x= %f' %(y1_min, x1_min) )
print('The minimum of function |x-0.2| in the given domain using \
 dichotomy method is %f found at :x= %f' %(y2_min, x2_min) )
print('The minimum of function x*sin(1/x) in the given domain using \
 dichotomy method is %f found at :x= %f' %(y3_min, x3_min) )
print('The number of f-calulations and iterations performed in\
 dichotomy method are %d, %d repectively' %(i1, j1))

# Golden section method for optimized minization
def golden_section(y,a,b,err):
    gr=(5**(0.5)+3)/2
    x2=b-(b-a)/gr
    x1=a+(b-a)/gr
    i=0
    j=0
    while abs(x2-x1)>err :
        if y(x1)<=y(x2):
            b=x2
            x2=x1
        else:
            a=x1
            x1=x2
        x2=b-(b-a)/gr
        x1=a+(b-a)/gr
        i=i+1
        j=j+1
    xmin=(a+b)/2
    return y(xmin) ,xmin, i, j+1

y1_min, x1_min , i1, j1 = golden_section(y1, 0, 1, 0.001)
y2_min, x2_min , i2 , j2= golden_section(y2, 0, 1, 0.001)
y3_min, x3_min , i3, j3 = golden_section(y3, 0.01, 1, 0.001)
print('The minimum of function x^3 in the given domain using \
 golden section method is %f found at :x= %f' %(y1_min, x1_min) )
print('The minimum of function |x-0.2| in the given domain using \
 golden section method is %f found at :x= %f' %(y2_min, x2_min) )
print('The minimum of function x*sin(1/x) in the given domain using \
 golden section method is %f found at :x= %f' %(y3_min, x3_min) )
print('The number of f-calulations and iterations performed in \
 golden section method are %d, %d repectively' %(i2, j2))

# generating the noisy data
alpha=np.random.rand()
beta=np.random.rand()
delta=[]
for i in range(101):
 delta.append(np.random.normal(0,1))
xk=np.arange(0, 1.01, 0.01)
yk=alpha*xk+beta+delta

# defining linear approximation function
def FL(x,a,b):
    return a*x+b
# defining rational approximation function
def FR(x,a, b):
    return a/(1+b*x)
# sum of least squares for linear approximation
def D1(a,b):
    err=0
    for i in range(101):
        err+=((a*xk[i]+b) -yk[i])**2
    return err
#sum of least squares for rational approximation
def D2(a,b):
    err = 0
    for i in range(101):
        err += ((a/ (xk[i] * b+1) )- yk[i]) ** 2
    return err

# exhaustive method for multi dimensional
def exhaustive_multi(y):
    m=float('inf')
    a=np.arange(-1, 1, 0.01)
    b=np.arange(-1, 1, 0.01)
    k=0
    n=0
    for i in range(len(a)):
        for j in range(len(b)):
            k=k+1
            n=n+1
            if y(a[i], b[j])<m:
                m=y(a[i],b[j])
                xm1=a[i]
                xm2=b[j]
    return m, xm1, xm2, k, n

# defining golden section method for gauss method
def gss2(y,a1,b1,a2, b2, b0,err):
    gr=(5**(0.5)+3)/2
    x2=b1-(b1-a1)/gr
    x1=a1+(b1-a1)/gr
    y2=b2-(b2-a2)/gr
    y1=a2+(b2-a2)/gr

    i=0
    while abs(x2-x1)>err and abs(y2-y1)>err:
        if y(x1, b0)<=y(x2, b0):
            b1=x2
            x2=x1
        else:
            a1=x1
            x1=x2
        x2=b1-(b1-a1)/gr
        x1=a1+(b1-a1)/gr
        xmin=(a1+b1)/2
        if y(xmin, y1 )<=y(xmin, y2):
            b2=y2
            y2=y1
        else:
            a2=y1
            y1=y2
          y2=b2-(b2-a2)/gr
          y1=a2+(b2-a2)/gr
          ymin=(a2+b2)/2
          i+=1
    return xmin, ymin, i

# deging gauss method for multi dimensional functions
def gauss(y, err):
    b0=0.5
    a1, b1=1, 1
    i=0
    j=0
    while abs(y(a1, b1)-y(a1, b0))> err:

        a1= gss2(y, -1,1,-1,1,b0 ,0.001)[0]
        b0=b1
        b1= gss2(y, -1,1,-1,1,b0, 0.001)[1]
        j+=gss2(y, -1,1,-1,1,b0, 0.001)[2]
        i+=1
        p=abs(b1-b0)
    return y(a1, b1), a1, b1, i, 2*j ,p

# defining nelder mead method for multi dimensional functions
def nelder_mead(y, alpha, beta, gamma, err):
    x1=np.array([0.5,0.5])
    x2=np.array([1,1])
    x3=np.array([1,0])
    xi=np.array([x1,x2,x3])
    i=0
    while abs((x3[0]-x2[0])**2 +(x3[1]-x2[1])**2)> err and abs((x3[0]-
x1[0])**2+(x3[1]-x1[1])**2)> err:
        xi=np.array([x1,x2,x3])
        yi=np.array([y(x1[0],x1[1]), y(x2[0],x2[1]), y(x3[0], x3[1])])
        ys=sorted(yi)
        yl=ys[0]
        xl=xi[np.where(yi==yl)[0][0]]
        yg=ys[1]
        xg=xi[np.where(yi==yg)[0][0]]
        yh=ys[2]
        xh=xi[np.where(yi==yh)[0][0]]
        xm=(xl+xg)/2
        xr=(1+alpha)*xm-alpha*xh
        yr=y(xr[0], xr[1])
        if yr<yg:
                xh=xr
            else:
                if yr < yh:
                    xh=xr
                xc=(xh+xm)/2
                yc=y(xc[0], xc[1])
                if yc< yh:
                    xh=xc
            if yr< yl :
                xe= (1-gamma)*xm+gamma*xr
                ye=y(xe[0], xe[1])
                if ye <yr:
                    xh=xe
                else:
                    xh=xr
            if yr> yg:
                xc=beta*xh+(1-beta)*xm
                if yc < yh:
                    xh=xc
            i+=1
            x1=xh
            x2=xg
            x3=xl
        p=abs((x3[0]-x2[0])**2 +(x3[1]-x2[1])**2)
        return y(xl[0], xl[1]), xl[0], xl[1], i, 5*i , p

# number of f-caluclations and iterations peroformed for least square error inlinear
approximation
print('The number of f-caluclation and iteration performed in \
 exhaustive multi dimensional method is %d, %d repectively for \
 least square error in linear approximation'%(exhaustive_multi(D1)[3],
exhaustive_multi(D1)[4]) )
print('The number of f-caluclation and iteration performed in \
 gauss method is %d, %d repectively and the precision %f \
 for least square error in linear approximation '%(gauss(D1, 0.001)[4],
gauss(D1, 0.001)[3] , gauss(D1, 0.001)[5]) )
print('The number of f-caluclation and iteration performed in \
 nelder mead method is %d, %d repectively and the precision %f \
 for least square error in linear approximation.'%(nelder_mead(D1, 1,
0.5, 2, 0.001)[3], nelder_mead(D1, 1, 0.5, 2, 0.001)[4], nelder_mead(D1, 1,
0.5, 2, 0.001)[5]) )
# ploting linear approximation of noisy data
yl_exhaustive_multi=FL(xk, exhaustive_multi(D1)[1], exhaustive_multi(D1)[2] )
yl_gauss=FL(xk, gauss(D1, 0.001)[1], gauss(D1, 0.001)[2] )
yl_nelder_mead=FL(xk, nelder_mead(D1,1, 0.5, 2, 0.001)[1], nelder_mead(D1,1,
0.5, 2, 0.001)[2] )
plt.plot(xk,yk,'bo', label='niosy data')
plt.plot(xk, yl_exhaustive_multi, '-r', label='Linear approximation byExhaustive
method')
plt.plot(xk, yl_gauss, '-k', label='Linear approximation by gauss method')
plt.plot(xk, yl_nelder_mead, '-g', label='Linear approximation by nelder mead
method')
plt.xlabel('x-axis')
plt.ylabel('Noisy data')
plt.title('Linear approximation of noisy data')
plt.legend()
plt.show()
# number of f-caluclations and iterations peroformed for least square error
inRational approximation
print('The number of f-caluclation and iteration performed in \
 exhaustive multi dimensional method is %d, %d repectively for \
 least square error in Rational approximation'%(exhaustive_multi(D2)[3],
exhaustive_multi(D2)[4]) )
print('The number of f-caluclation and iteration performed in \
 gauss method is %d, %d repectively and the precision %f \
 for least square error in Rational approximation '%(gauss(D2, 0.001)[4],
gauss(D2, 0.001)[3] , gauss(D2, 0.001)[5]) )
print('The number of f-caluclation and iteration performed in \
 nelder mead method is %d, %d repectively and the precision %f \
 for least square error in Rational approximation.'%(nelder_mead(D2, 1,
0.5, 2, 0.001)[3], nelder_mead(D2, 1, 0.5, 2, 0.001)[4], nelder_mead(D2, 1,
0.5, 2, 0.001)[5]) )
# ploting rational approximation of noisy data
yr_exhaustive_multi=FR(xk, exhaustive_multi(D2)[1], exhaustive_multi(D2)[2] )
yr_gauss=FR(xk, gauss(D2, 0.001)[1], gauss(D2, 0.001)[2] )
yr_nelder_mead=FR(xk, nelder_mead(D2,1, 0.5, 2, 0.001)[1], nelder_mead(D2,1,
0.5, 2, 0.001)[2] )
plt.plot(xk,yk,'bo', label='niosy data')
plt.plot(xk, yr_exhaustive_multi, '-r', label='Rational approximation by Exhaustive
method')
plt.plot(xk, yr_gauss, '-k', label='Rational approximation by gauss method')
plt.plot(xk, yr_nelder_mead, '-g', label='Rational approximation by nelder mead
method')
plt.xlabel('x-axis')
plt.ylabel('Noisy data')
plt.title('Rational approximation of noisy data')
plt.legend()
plt.show()
```