FEDERAL STATE AUTONOMOUS EDUCATIONAL INSTITUTION

OF HIGHER EDUCATION

ITMO UNIVERSITY

Report

on the practical task No. 2

“Algorithms for unconstrained nonlinear optimization. Direct methods”

Performed by

*Mariia Koroleva*

*J42322c*

Accepted by

Dr Petr Chunaev

St. Petersburg

2021

**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 ) solution for the following functions and domains:*

1. *, ;*
2. *, ;*
3. *, .*

*Calculate the number of -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 and . Furthermore, generate the noisy data , where , according to the following rule:*

*,*

*where are values of a random variable with standard normal distribution. Approximate the data by the following linear and rational functions:*

1. *(linear approximant),*
2. *(rational approximant),*

*by means of least squares through the numerical minimization (with precision ) of the following function:*

*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 methods are numerical methods for finding the optimal values of objective functions, for example, within the framework of mathematical models of certain processes. Optimization techniques are widely used in data analysis and machine learning.

Let an objective function f = f (x) be given, where x is, generally speaking, a multidimensional vector from some subset Q of the Euclidean space. The subset of Q can be either bounded or, in particular, coincide with the entire space. In what follows, for brevity, we will consider only the problem of minimizing a function f on the set Q (from minimizing one can go to maximizing by considering

F (x) = −f (x) instead of f (x)).

Methods for minimization can be direct (or zero-), first- and second-order depending on which derivatives are used.

One-dimensional direct methods:

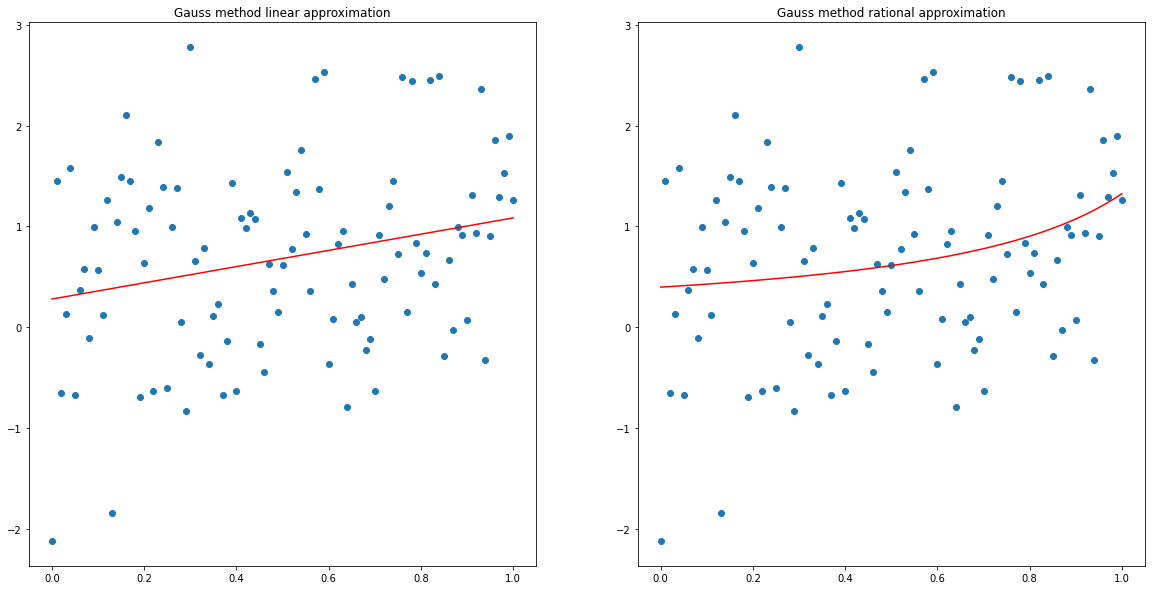
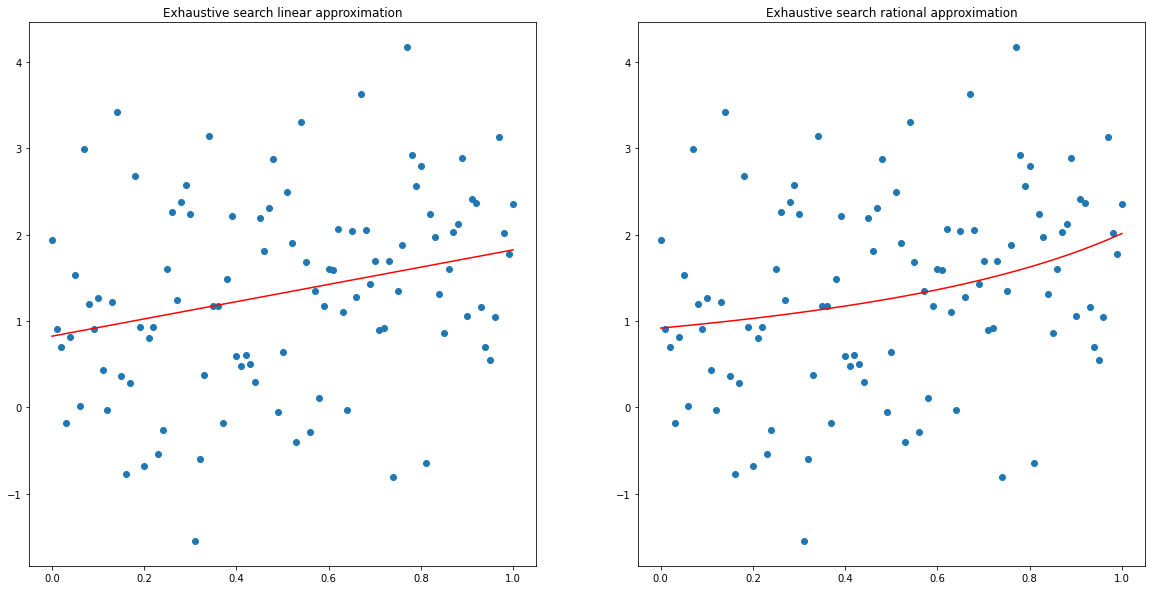
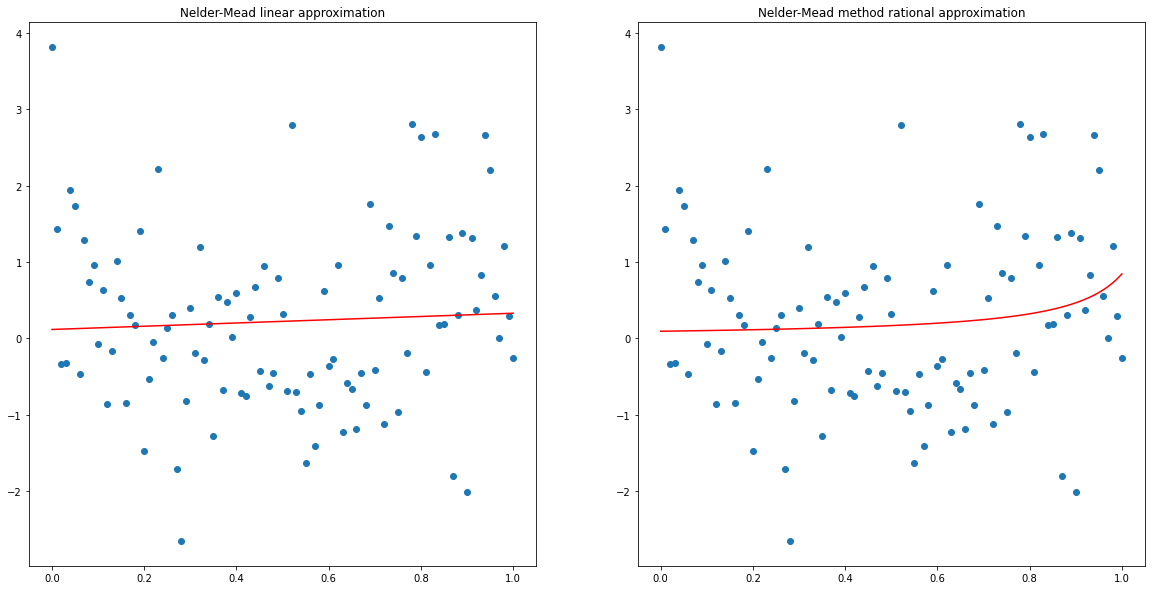
* *Exhaustive search* (brute-force search): calculate f(x) in n points and choosing the point where the function has the least value with a given level of approximation.
* *Dichotomy* method: The search area [a, b] is taken, and then gradually narrows to the area that includes the optimum in delta(<eps) steps.
* *Golden section method:* the same with a special choice of delta

Multidimensional direct methods:

* Exhaustive search
* Gauss method
* Nelder-Mead method

**Results**

All the algorithms were built by use of python3 functions.

****

**Conclusions**

The work and the results demonstrate the great difference between minimization algorithms in their efficiency.

Thus, we see that the most ineffective of the considered direct algorithms is exhaustive search. The most time efficient and precise one among multidimensional algorithms is Nelder-Mead method.

**Appendix**

import numpy as np

import itertools

import matplotlib.pyplot as plt

import random

from scipy import optimize

Part I

# Functions

func1 = lambda x: x\*\*3

func2 = lambda x: np.abs(x - 0.2)

func3 = lambda x: x \* np.sin(1/x)

funcs = [func1, func2, func3]

eps = 0.001

# One-dimensional direct methods

def exhaustive\_search(f, a, b, eps=eps):

    n = (b - a) / eps

    x\_k = np.arange(a, b, (b-a)/n)

    x\_m = x\_k[np.argmin(f(x\_k))]

    return x\_m, len(f(x\_k)), len(f(x\_k))

delta = np.random.uniform(0, eps)

def dichotomy(f, a, b, delta = delta, eps=eps):

    iter = 0

    while np.abs(a - b) >= eps:

      x1 = (a + b - delta)/2

      x2 = (a + b + delta)/2

      if f(x1) <= f(x2):

        b = x2

      else:

        a = x1

      iter += 1

    return (a, b), iter\*2, iter

def golden\_section(f, a, b, eps=eps):

    x1 = a + (3 - 5\*\*0.5)\*(b - a)/2

    x2 = b + (5\*\*0.5 - 3)\*(b - a)/2

    i = 0

    f1 = f(x1)

    f2 = f(x2)

    while np.abs(a - b) >= eps:

        if f(x1) <= f(x2):

            b = x2

            x2 = x1

            x1 = a + (3 - np.sqrt(5))/2\*(b - a)

            f2 = f1

            f1 = f(x1)

        else:

            a = x1

            x1 = x2

            x2 = b + (np.sqrt(5) - 3)/2\*(b - a)

            f1 = f2

            f2 = f(x2)

        i += 1

    return (a, b), i + 2, i

# 1

print('Exhaustive search results:')

print(' Function 1')

print('   x\_min = ', exhaustive\_search(func1, 0, 1)[0])

print('   Number of f-calculations = ', exhaustive\_search(func1, 0, 1)[1])

print('   Number of iterations = ', exhaustive\_search(func1, 0, 1)[2])

print(' Function 2')

print('   x\_min = ', exhaustive\_search(func2, 0, 1)[0])

print('   Number of f-calculations = ', exhaustive\_search(func2, 0, 1)[1])

print('   Number of iterations = ', exhaustive\_search(func2, 0, 1)[2])

print(' Function 3')

print('   x\_min = ', exhaustive\_search(func3, 0.01, 1)[0])

print('   Number of f-calculations = ', exhaustive\_search(func3, 0.01, 1)[1])

print('   Number of iterations = ', exhaustive\_search(func3, 0.01, 1)[2])

Exhaustive search results:

Function 1

x\_min = 0.0

Number of f-calculations = 1000

Number of iterations = 1000

Function 2

x\_min = 0.2

Number of f-calculations = 1000

Number of iterations = 1000

Function 3

x\_min = 0.22299999999999984

Number of f-calculations = 990

Number of iterations = 990

# 2

print('Dichotomy method results:')

print(' Function 1')

print('   x\_min section is ', dichotomy(func1, 0, 1)[0])

print('   Number of f-calculations = ', dichotomy(func1, 0, 1)[1])

print('   Number of iterations = ', dichotomy(func1, 0, 1)[2])

print(' Function 2')

print('   x\_min section is  ', dichotomy(func2, 0, 1)[0])

print('   Number of f-calculations = ', dichotomy(func2, 0, 1)[1])

print('   Number of iterations = ', dichotomy(func2, 0, 1)[2])

print(' Function 3')

print('   x\_min section is  ', dichotomy(func3, 0.01, 1)[0])

print('   Number of f-calculations = ', dichotomy(func3, 0.01, 1)[1])

print('   Number of iterations = ', dichotomy(func3, 0.01, 1)[2])

Dichotomy method results:

Function 1

x\_min section is (0, 0.0009349087389926897)

Number of f-calculations = 24

Number of iterations = 12

Function 2

x\_min section is (0.19956904631294362, 0.2005039550519363)

Number of f-calculations = 24

Number of iterations = 12

Function 3

x\_min section is (0.2220638079809162, 0.2229962753136589)

Number of f-calculations = 24

Number of iterations = 12

# 3

print('Golden section results:')

print(' Function 1')

print('   x\_min section is ', golden\_section(func1, 0, 1)[0])

print('   Number of f-calculations = ', golden\_section(func1, 0, 1)[1])

print('   Number of iterations = ', golden\_section(func1, 0, 1)[2])

print(' Function 2')

print('   x\_min section is  ', golden\_section(func2, 0, 1)[0])

print('   Number of f-calculations = ', golden\_section(func2, 0, 1)[1])

print('   Number of iterations = ', golden\_section(func2, 0, 1)[2])

print(' Function 3')

print('   x\_min section is  ', golden\_section(func3, 0.01, 1)[0])

print('   Number of f-calculations = ', golden\_section(func3, 0.01, 1)[1])

print('   Number of iterations = ', golden\_section(func3, 0.01, 1)[2])

Golden section results:

Function 1

x\_min section is (0, 0.000733137435857404)

Number of f-calculations = 17

Number of iterations = 15

Function 2

x\_min section is (0.199706745025657, 0.2004398824615144)

Number of f-calculations = 17

Number of iterations = 15

Function 3

x\_min section is (0.22235669058885382, 0.22308249665035265)

Number of f-calculations = 17

Number of iterations = 15

Part II

alpha = np.random.uniform(0, 1)

beta = np.random.uniform(0, 1)

# Generating the noisy data

X = np.arange(0, 1.01, 0.01)

def y(x):

  return alpha\*x + beta + np.random.normal(0, 1)

Y = np.array([y(x) for x in X])

# Functions for approximation

def F\_linear(x, a, b):

    return a\*x + b

def F\_rational(x, a, b):

    return a/(1 + b\*x)

def D(func, X, Y, a, b):

    return sum([(func(x\_i, a, b) - y\_i)\*\*2 for x\_i, y\_i in zip(X, Y)])

eps = 0.001

# Exhaustive search

def exhaustive\_multidim(f, X, Y, a, b):

    f\_min = D(f, X, Y, a, b)

    a\_opt = 0

    b\_opt = 0

    i = 0

    options = np.arange(-1, 1 + eps, eps)

    for a, b in list(itertools.product(options, repeat=2)):

        new\_f = D(f, X, Y, a, b)

        if new\_f < f\_min:

            f\_min = new\_f

            a\_opt = a

            b\_opt = b

        i += 1

    return a\_opt, b\_opt, i+1, i

# Linear function approximation

%%time

F\_linear\_opt\_1 = exhaustive\_multidim(F\_linear, X, Y, alpha, beta)

F\_linear\_opt\_1

CPU times: user 7min 24s, sys: 1.02 s, total: 7min 25s Wall time: 7min 28s

print('Optimal linear function 1: {}x + {}'.format(round(F\_linear\_opt\_1[0], 2), round(F\_linear\_opt\_1[1], 2)))

print('Number of iterations: {}'.format(F\_linear\_opt\_1[2]))

print('Number of function evaluations: {}'.format(F\_linear\_opt\_1[3]))

Optimal linear function 1: 1.0x + 0.82

Number of iterations: 4004002

Number of function evaluations: 4004001

# Rational function approximation

%%time

F\_rational\_opt\_1 = exhaustive\_multidim(F\_rational, X, Y, alpha, beta)

F\_rational\_opt\_1

CPU times: user 9min 19s, sys: 1.12 s, total: 9min 21s

Wall time: 9min 21s

print('Optimal rational function 1: {}/(1{}x)'.format(round(F\_rational\_opt\_1[0], 2), round(F\_rational\_opt\_1[1], 2)))

print('Number of iterations: {}'.format(F\_rational\_opt\_1[2]))

print('Number of function evaluations: {}'.format(F\_rational\_opt\_1[3]))

Optimal rational function 1: 0.92/(1-0.54x)

Number of iterations: 4004002

Number of function evaluations: 4004001

# Rational function approximation

%%time

F\_rational\_opt\_1 = exhaustive\_multidim(F\_rational, X, Y, alpha, beta)

F\_rational\_opt\_1

CPU times: user 9min 19s, sys: 1.12 s,

total: 9min 21s Wall time: 9min 21s

print('Optimal rational function 1: {}/(1{}x)'.format(round(F\_rational\_opt\_1[0], 2), round(F\_rational\_opt\_1[1], 2)))

print('Number of iterations: {}'.format(F\_rational\_opt\_1[2]))

print('Number of function evaluations: {}'.format(F\_rational\_opt\_1[3]))

Optimal rational function 1: 0.92/(1-0.54x)

Number of iterations: 4004002

Number of function evaluations: 4004001

# Gauss

eps = 0.001

def dichotomy\_for\_D(f, X, Y, a, b, a\_opt, b\_opt, par = 'a'):

    i = 0

    delta = np.random.uniform(0, eps)

    while np.abs(a - b) > eps:

        x1 = (a + b - delta) / 2

        x2 = (a + b + delta) / 2

        f\_1 = D(f, X, Y, x1, b\_opt) if par == 'a' else D(f, X, Y, a\_opt, x1)

        f\_2 = D(f, X, Y, x2, b\_opt) if par == 'a' else D(f, X, Y, a\_opt, x2)

        if f\_1 <= f\_2:

            b = x2

            fmin = f\_1

        else:

            a = x1

            fmin = f\_2

        i += 1

    return x1, i\*2, i

def gauss(f, X, Y, z1 = 0, z2 = 0):

    f\_min\_prev = D(f, X, Y, z1, z2)

    z1\_prev = z1

    z1 =  dichotomy\_for\_D(f, X, Y, -1, 1 + eps, z1, z2)[0]

    f\_min = D(f, X, Y, z1, z2)

    i = 1 + dichotomy\_for\_D(f, X, Y, -1, 1 + eps, z1, z2)[2]

    j = 2 + dichotomy\_for\_D(f, X, Y, -1, 1 + eps, z1, z2)[1]

    l = 1

    while (np.abs(f\_min - f\_min\_prev) >= eps) and (np.abs(z1 - z1\_prev) >= eps):

      if l % 2 == 0:

          z1\_prev = z1

          z1 =  dichotomy\_for\_D(f, X, Y, -1, 1 + eps, z1, z2)[0]

          f\_min\_prev = f\_min

          f\_min = D(f, X, Y, z1, z2)

          i += 1 + dichotomy\_for\_D(f, X, Y, -1, 1 + eps, z1, z2)[2]

          j += 1 + dichotomy\_for\_D(f, X, Y, -1, 1 + eps, z1, z2)[1]

      else:

          z2\_prev = z2

          z2 =  dichotomy\_for\_D(f, X, Y, -1, 1 + eps, z1, z2, par = 'b')[0]

          f\_min\_prev = f\_min

          f\_min = D(f, X, Y, z1, z2)

          i += 1 + dichotomy\_for\_D(f, X, Y, -1, 1 + eps, z1, z2, par = 'b')[2]

          j += 1 + dichotomy\_for\_D(f, X, Y, -1, 1 + eps, z1, z2, par = 'b')[1]

          if np.abs(z2 - z2\_prev) < eps:

              break

      l += 1

    return z1, z2, j, i

%%time

F\_linear\_opt\_2 = gauss(F\_linear, X, Y, alpha, beta)

CPU times: user 214 ms, sys: 0 ns, total: 214 ms

Wall time: 218 ms

print('Optimal linear function 2: {}x + {}'.format(round(F\_linear\_opt\_2[0], 2), round(F\_linear\_opt\_2[1], 2)))

print('Number of iterations: {}'.format(F\_linear\_opt\_2[3]))

print('Number of function evaluations: {}'.format(F\_linear\_opt\_2[2]))

Optimal linear function 2: 0.8x + 0.28

Number of iterations: 323

Number of function evaluations: 579

%%time

F\_rational\_opt\_2 = gauss(F\_rational, X, Y, alpha, beta)

CPU times: user 328 ms, sys: 0 ns, total: 328 ms

Wall time: 334 ms

print('Optimal rational function 2: {}/(1{}x)'.format(round(F\_rational\_opt\_2[0], 2), round(F\_rational\_opt\_2[1], 2)))

print('Number of iterations: {}'.format(F\_rational\_opt\_2[3]))

print('Number of function evaluations: {}'.format(F\_rational\_opt\_2[2]))

Optimal rational function 2: 0.4/(1-0.7x)

Number of iterations: 414

Number of function evaluations: 723

# Nelder-Mead

def F\_linear\_point(x, point):

    return point[0]\*x + point[1]

def F\_rational\_point(x, point):

    return point[0]/(1 + point[1]\*x)

def D\_point(point, f, X, Y):

    return sum([(f(x\_i, point) - y\_i)\*\*2 for x\_i, y\_i in zip(X, Y)])

def nelder\_mead(f, X, Y):

    result = optimize.minimize(D\_point, np.random.rand(2), args=(f, X, Y), method ='nelder-mead')

    return result.x

%%time

F\_linear\_opt\_3 = nelder\_mead(F\_linear\_point, X, Y)

CPU times: user 27.1 ms, sys: 724 µs, total: 27.8 ms

Wall time: 29.1 ms

print('Optimal linear function 3: {}x + {}'.format(round(F\_linear\_opt\_3[0], 2), round(F\_linear\_opt\_3[1], 2)))

Optimal linear function 3: 0.21x + 0.12

%%time

F\_rational\_opt\_3 = nelder\_mead(F\_rational\_point, X, Y)

CPU times: user 38.7 ms, sys: 0 ns, total: 38.7 ms

Wall time: 73.9 ms

print('Optimal rational function 3: {}/(1{}x)'.format(round(F\_rational\_opt\_3[0], 2), round(F\_rational\_opt\_3[1], 2)))

Optimal rational function 3: 0.09/(1-0.89x)

Plots

# Exhaustive search

fig, axes = plt.subplots(1, 2, figsize=(20, 10))

axes.flat[0].plot(X, F\_linear(X, F\_linear\_opt\_1[0], F\_linear\_opt\_1[1]), color = 'red')

axes.flat[0].scatter(X, Y)

axes.flat[0].set\_title('Exhaustive search linear approximation')

axes.flat[1].plot(X, F\_rational(X, F\_rational\_opt\_1[0], F\_rational\_opt\_1[1]), color = 'red')

axes.flat[1].scatter(X, Y)

axes.flat[1].set\_title('Exhaustive search rational approximation')

# Gauss method

fig, axes = plt.subplots(1, 2, figsize=(20, 10))

axes.flat[0].plot(X, F\_linear(X, F\_linear\_opt\_2[0], F\_linear\_opt\_2[1]), color = 'red')

axes.flat[0].scatter(X, Y)

axes.flat[0].set\_title('Gauss method linear approximation')

axes.flat[1].plot(X, F\_rational(X, F\_rational\_opt\_2[0], F\_rational\_opt\_2[1]), color = 'red')

axes.flat[1].scatter(X, Y)

axes.flat[1].set\_title('Gauss method rational approximation')

# Nelder-Mead

fig, axes = plt.subplots(1, 2, figsize=(20, 10))

axes.flat[0].plot(X, F\_linear(X, F\_linear\_opt\_3[0], F\_linear\_opt\_3[1]), color = 'red')

axes.flat[0].scatter(X, Y)

axes.flat[0].set\_title('Nelder-Mead linear approximation')

axes.flat[1].plot(X, F\_rational(X, F\_rational\_opt\_3[0], F\_rational\_opt\_3[1]), color = 'red')

axes.flat[1].scatter(X, Y)

axes.flat[1].set\_title('Nelder-Mead method rational approximation')