Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОССУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОННИКИ

КАФЕДРА ИНФОРМАТИКИ

Отчёт по лабораторной работе №5

По дисциплине «Методы численного анализа»

По теме «Решение нелинейных уравнений»

Выполнил:

студент гр. 653504

Куликов А.Д.

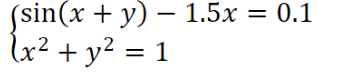
Проверил:

Пашук А. В.

Минск 2018

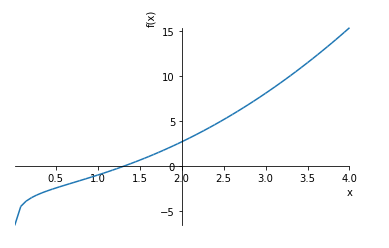
**Задание.**

1. Найти численное решение нелинейного уравнения   
   
2. Найти численное решение системы нелинейных уравнений



**Решение.**

1. **Отделим корни уравнения графически:**



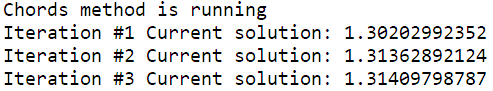
Отсюда можно заключить, что функция непрерывна и монотонно возрастает на всей области определения. Корень находится в промежутке от 1 до 1.5.

1. **Решение уравнения методом хорд.**

Для начала проверим условие сходимости метода хорд на отрезке [1, 1.5].

Как видим, первая и вторая производные непрерывны на этом отрезке и имеют постоянный знак. Следовательно, метод хорд сходится.

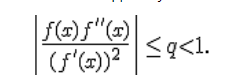
Найдем решение уравнения (запустим метод хорд с параметрами a=1, b=1.5):



Точность 0.001 достигнута через 3 итерации.

1. **Решение уравнения методом Ньютона.**

Для начала проверим сходимость метода Ньютона. По формуле имеем

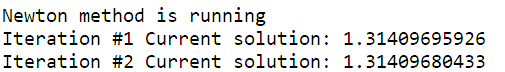


Подставив значения функции и производной, получим

Так как первая производная непрерывна и монотонно возрастает на промежутке [1, 1.5], то максимальное значение она принимает в точке 1.5.

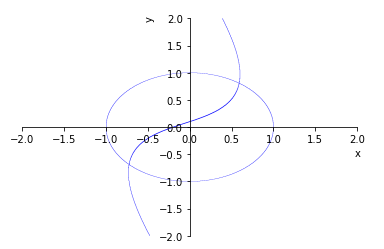
Подставив это значение, получим q = 0.2780761066 < 1, что и требовалось доказать.

Запустим решение уравнения методом Ньютона с начальным приближением, равным 1.25:



Как видим, решение сошлось к тому же ответу за 2 итерации.

1. **Отделим корни системы уравнений графически:**



Первый корень находится в области W1 = [(-1, -0.5), (-1, -0.5)], второй корень находится в области W2 = [(0.5, 1), (0.5, 1)].

1. **Найдем все вспомогательные матрицы и векторы для методов из пунктов 6-8.**

Якобиан системы:

Матрица F (для метода Ньютона):

Матрица Ф (для метода простых итераций):

Якобиан матрицы :

Формулы для вычисления корней уравнения:

А. Метод простых итераций

Б. Метод Ньютона

В. Модифицированный метод Ньютона

Все методы сходятся при .

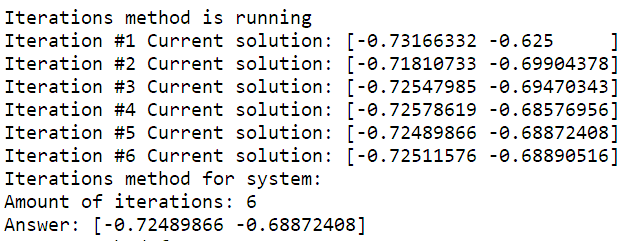
1. **Метод простых итераций для системы нелинейных уравнений.**

Начальное приближение для 1 решения: (-0.75, -0.75), для 2 – (0.5, 0.75).

Условие сходимости:

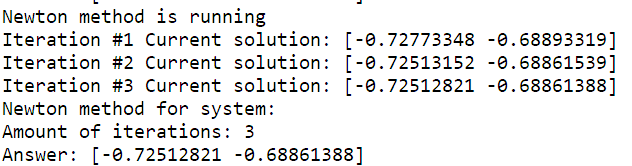
Подставив начальные значения, убедимся в сходимости метода.

Вычисление корней системы:

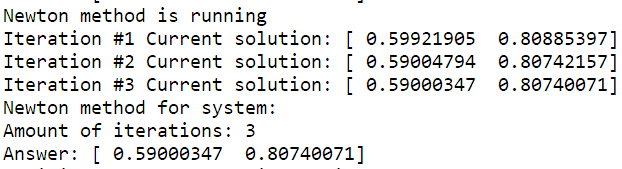


1. **Метод Ньютона для систем нелинейных уравнений.**

Сходимость метода аналогична сходимости метода простых итераций.

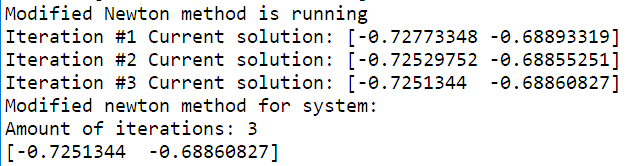
Выполним метод с начальными условиями : 

Выполним метод с начальными условиями :

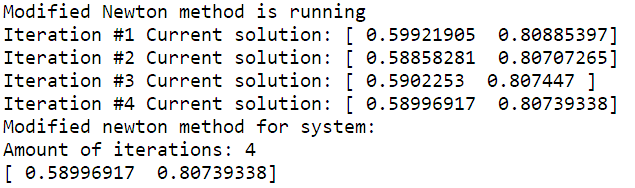


1. **Модифицированный метод Ньютона для систем нелинейных уравнений.**

Сходимость метода аналогична сходимости метода простых итераций.

Выполним метод с начальными условиями : 

Выполним метод с начальными условиями :



Как видим, для нахождения решения, модифицированному методу Ньютона в первом случае понадобилось столько же, а во втором на 1 итерацию больше.

**Приложение**

Программная реализация на языке python:

from matplotlib import pyplot as plt

import numpy as np

from sympy import \*

from sympy.plotting import plot\_parametric

from sympy.utilities.lambdify import lambdastr

from pprint import pprint

def f(x):

return x \* x + np.log(x) - 2

def graph():

x = Symbol('x')

y = Symbol('y')

p1 = plot\_implicit(Eq(sin(x+y)-1.5\*x, 0.1), (x, -2, 2), (y, -2, 2))

p2 = plot\_implicit(Eq(x\*\*2+y\*\*2, 1), (x, -2, 2), (y, -2, 2))

p1.extend(p2)

p1.show()

def newton\_eq(f, fstr, x0, e=0.00001, max\_iterations=10000):

print('Newton method is running')

x = Symbol('x')

df = lambdify(x, fstr.diff(x), 'numpy')

xk = x0 - f(x0) / df(x0)

for k in range(max\_iterations):

xp = xk

xk -= f(xk) / df(xk)

print('Iteration #' + str(k + 1) + ' Current solution: ' + str(xk))

if abs(xp - xk) < e:

break

return xk, k

def chords(f, a, b, e=0.001, max\_iterations=10000):

print('Chords method is running')

xn = b

xp = a

i = 0

while abs(f(xn)) > e:

tmp = xn

xn = xp - f(xp) / (f(xn) - f(xp)) \* (xn - xp)

xp = tmp

i += 1

print('Iteration #' + str(i) + ' Current solution: ' + str(xn))

if i > max\_iterations:

break

return xn, i

def iterations(x0, F, e=0.001, max\_iterations=10000):

print('Iterations method is running')

xp = np.copy(x0)

for i in range(max\_iterations):

xk = np.copy(xp)

for j in range(len(xk)):

xk[j] = F[j](\*xp)

print('Iteration #' + str(i + 1) + ' Current solution: ' + str(xk))

if np.max(np.abs(xk - xp)) < e:

break

xp = xk

return xp, i + 1

def build\_F(exprs):

x, y = symbols('x y')

F = []

for i in range(len(exprs)):

F.append(lambdify((x, y), exprs[i], 'numpy'))

return F

def build\_jacobian(syms, funcs):

J = []

for i in range(len(funcs)):

J.append([])

for sym in syms:

J[i].append(lambdify(syms, funcs[i].diff(sym), 'numpy'))

return J

def eval\_jacobian(J, vals):

rows, cols = J.shape

M = np.zeros(J.shape)

for i in range(rows):

for j in range(cols):

M[i, j] = J[i, j](\*vals)

return M

def eval\_F(F, vals):

F1 = np.zeros(F.shape)

for i in range(len(F)):

F1[i] = F[i](\*vals)

return F1

def newton\_syst(J, F, x0, e=0.001, max\_iterations=10000):

print('Newton method is running')

xp = np.copy(x0)

xk = np.copy(x0)

for i in range(max\_iterations):

xp = np.copy(xk)

xk = xk - np.dot(

np.linalg.inv(eval\_jacobian(J, xk)),

eval\_F(F, xk))

print('Iteration #' + str(i + 1) + ' Current solution: ' + str(xk))

if np.max(np.abs(xp - xk)) < e:

break

return xk, i + 1

def newton\_syst\_mod(J, F, x0, e=0.001, max\_iterations=10000):

print('Modified Newton method is running')

xp = np.copy(x0)

xk = np.copy(x0)

J0 = np.linalg.inv(eval\_jacobian(J, xk))

for i in range(max\_iterations):

xp = np.copy(xk)

xk = xk - np.dot(J0, eval\_F(F, xk))

print('Iteration #' + str(i + 1) + ' Current solution: ' + str(xk))

if np.max(np.abs(xp - xk)) < e:

break

return xk, i + 1

def main():

a = 1

b = 1.5

ans, iters\_chords = chords(f, a, b)

x, y = symbols('x y')

fstr = x\*\*2 + log(x) - 2

plot(x\*\*2 + log(x) - 2, (x, 0.01, 4))

ans1, iters\_newton\_eq = newton\_eq(f, fstr, 1.25)

print(ans, ans1)

print(iters\_chords, iters\_newton\_eq)

graph()

F = build\_F([2/3\*(sin(x+y) - 0.1), x\*\*2+y\*\*2-1+y])

x0 = np.zeros(2)

x0, iters\_iter = iterations([-0.5, -0.4], F)

print('Iterations method for system: ')

print('Amount of iterations: ' + str(iters\_iter))

print('Answer: ' + str(x0))

J = np.array(build\_jacobian([x, y], [sin(x+y)-0.1-1.5\*x, x\*\*2+y\*\*2-1]))

F = np.array(build\_F([sin(x+y)-0.1-1.5\*x, x\*\*2+y\*\*2-1]))

x0, iters\_newton\_sys = newton\_syst(J, F, [0.5, 0.75])

print('Newton method for system: ')

print('Amount of iterations: ' + str(iters\_newton\_sys))

print('Answer: ' + str(x0))

x0, iters\_newton\_sys\_mod = newton\_syst\_mod(J, F, [0.5, 0.75])

print('Modified newton method for system: ')

print('Amount of iterations: ' + str(iters\_newton\_sys\_mod))

print(x0)

if \_\_name\_\_ == '\_\_main\_\_':

main()