БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ
КАФЕДРА ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ

Лабораторная работа 4

Метод простой итерации решения системы нелинейных уравнений

Вариант 7

Выполнил:
Журик Никита Сергеевич
2 курс, 6 группа
Преподаватель:
Будник Анатолий Михайлович

Минск, 2019

Содержание

1. Постановка задачи

1. Отделить корень и определить шар S_δ.
2. Решить систему нелинейных уравнений методом простой итерации.
3. Вычислить невязку решения.
4. Проанализировать полученные результаты.

2. Решение системы нелинейных уравнений

- Рассмотрим систему нелинейных уравнений:

$$\begin{cases} y - \frac{x^2}{2} + x - 0.5 = 0, \\ 2x + y - \frac{y^3}{6} - 1.6 = 0. \end{cases}$$

Отделим корни путём выражения $y(x)$ из первого уравнения: $y(x) = \frac{x^2}{2} - x + 0.5$. Тогда подставим данное выражение во второе уравнение и отделим корни полученного нелинейного уравнения. Получим для каждого корня отрезки $[a_i, b_i]$, положим

$$x_i^0 = \left(\frac{a_i + b_i}{2}, y\left(\frac{a_i + b_i}{2}\right)\right)^T; \ S_\delta = \left\{\mathbf{x} \middle| ||\mathbf{x} - \mathbf{x}_i^0||_\infty \leq \delta, \ \delta = \max\left(\frac{y(a) + y(b)}{2}, \frac{b-a}{2}\right)\right\} \tag{1}$$

Как видно из результата выполнения программы, для корней получаем:

$$\begin{aligned} x_1^0 &= (0.78282828, 0.02358178)^T \\ r_1 &= 0.05050505 \\ x_2^0 &= (3.81313131, 3.95686389)^T \\ r_2 &= 0.07103867 \end{aligned}$$

- Построим функции $\phi_i(x, y)$ следующим образом:

$$\begin{cases} \phi_1(x, y) = x + c_1 f_1(x, y) + c_2 f_2(x, y), \\ \phi_2(x, y) = y + c_3 f_1(x, y) + c_4 f_2(x, y), \\ \left|\left|\frac{\mathrm{d}\phi}{\mathrm{d}\mathbf{x}}\right|\right|\Big|_{x^0} = 0 \end{cases} \tag{2}$$

Данная система эквивалентна следующей:

$$\begin{cases} 1 + c_1 \frac{\partial f_1}{\partial x}\Big|_{x^0} + c_2 \frac{\partial f_2}{\partial x}\Big|_{x^0} = 0, \\ c_1 \frac{\partial f_1}{\partial y}\Big|_{x^0} + c_2 \frac{\partial f_2}{\partial y}\Big|_{x^0} = 0, \\ c_3 \frac{\partial f_1}{\partial x}\Big|_{x^0} + c_4 \frac{\partial f_2}{\partial x}\Big|_{x^0} = 0, \\ 1 + c_3 \frac{\partial f_1}{\partial y}\Big|_{x^0} + c_4 \frac{\partial f_2}{\partial y}\Big|_{x^0} = 0. \end{cases} \tag{3}$$

Решив данную систему относительно коэффициентов c_i, получим искомые функции $\phi_1(x, y), \phi_2(x, y)$.

3. Листинг программы

Для реализации алгоритма был использован Python и библиотеки numpy и matplotlib.

```
#SystemCommon.py
import math
import numpy as np
import matplotlib.pyplot as plt

def f1(x, y):
```

```
    return y - 0.5 * x ** 2 + x - 0.5

def f1_xprime(x, y):
    return -x + 1

def f1_yprime(x, y):
    return 1

def f2(x, y):
    return 2 * x + y - (y ** 3) / 6 - 1.6

def f2_xprime(x, y):
    return 2

def f2_yprime(x, y):
    return 1 - (y ** 2) / 2

def f(x, y):
    return np.array([f1(x, y), f2(x, y)], dtype=np.double)

def df(x, y):
    return np.array([[f1_xprime(x, y), f1_yprime(x, y)], [f2_xprime(x, y), f2_yprime(x, y)]],
        dtype=np.double)

def ysub(x):
    return 0.5 * x ** 2 - x + 0.5

def ysub_prime(x):
    return x - 1

def f2_ysub(x):
    return f2(x, ysub(x))

def f2_ysub_prime(x):
    return 2 + ysub_prime(x) - (ysub(x) ** 2) * ysub_prime(x) / 2

def infNorm(x):
    return np.max(np.abs(x))

import Newton

intervals = Newton.get_intervals_table(0.0, 5.0, f2_ysub, 100)

roots = np.array([(interval[0] + interval[1]) / 2 for interval in intervals])
radii = np.array([max((ysub(interval[1]) - ysub(interval[0])) / 2, interval[1] - interval[0]) for
    interval in intervals])

print("x values for starting points: " + str(roots))
print("Sphere radia: " + str(radii))

points = np.array([(root, ysub(root)) for root in roots], dtype=np.double)

print("Starting points: " + str(points))

#SystemSimpleIteration.py

from SystemCommon import *

c = np.zeros((1, 1))

def getC(point):
    global c
    A = np.array([[f1_xprime(point[0], point[1]), f2_xprime(point[0], point[1]), 0, 0],
                  [f1_yprime(point[0], point[1]), f2_yprime(point[0], point[1]), 0, 0],
                  [0, 0, f1_xprime(point[0], point[1]), f2_xprime(point[0], point[1])],
                  [0, 0, f1_yprime(point[0], point[1]), f2_yprime(point[0], point[1])]],
```

```
                    dtype=np.double)
    b = np.array([-1, 0, 0, -1])
    c = np.linalg.solve(A, b)

def phi1(x, y):
    global c
    return x + c[0] * f1(x, y) + c[1] * f2(x, y)

def phi2(x, y):
    global c
    return y + c[2] * f1(x, y) + c[3] * f2(x, y)

def phi(x, y):
    return np.array([phi1(x, y), phi2(x, y)], dtype=np.double)

def phi1_xprime(x, y):
    global c
    return 1 + c[0] * f1_xprime(x, y) + c[1] * f2_xprime(x, y)

def phi1_yprime(x, y):
    global c
    return c[0] * f1_yprime(x, y) + c[1] * f2_yprime(x, y)

def phi2_xprime(x, y):
    global c
    return c[2] * f1_xprime(x, y) + c[3] * f2_xprime(x, y)

def phi2_yprime(x, y):
    global c
    return 1 + c[2] * f1_yprime(x, y) + c[3] * f2_yprime(x, y)

def simpleIteration(phi, point, outerEps):
    prevPoint = np.copy(point)
    point = phi(prevPoint[0], prevPoint[1])
    iterNum = 1
    while (infNorm(point - prevPoint) >= outerEps):
        prevPoint = np.copy(point)
        point = phi(prevPoint[0], prevPoint[1])
        iterNum += 1
        print("Iteration #{0}: {1}".format(iterNum, prevPoint))
    return (point, iterNum)

if __name__ == '__main__':
    roots = []

    for point in points:
        print("Starting point: " + str(point))
        print("Deficiency before: " + str(f(point[0], point[1])))
        getC(point)
        print("C vector: " + str(c))
        (root, iterNum) = simpleIteration(phi, point, 10 ** -5)
        print("Simple iteration root: " + str(root))
        if len(roots) == 0:
            roots = [root]
        else:
            roots.append(root)
        print("Deficiency after: " + str(f(root[0], root[1])))
        print("Number of iterations: " + str(iterNum))
        print("\n")

    print(roots)
```

4. Вывод программы

```
x values for starting points: [0.78282828 3.81313131]
Sphere radia: [0.05050505 0.07103867]
Starting points: [[0.78282828 0.02358178]
 [3.81313131 3.95685389]]

Starting point: [0.78282828 0.02358178]
Deficiency before: [ 0.         -0.01076384]
C vector: [ 0.56073156 -0.56088752 -1.12177504  0.12180891]
Iteration #2: [0.78886559 0.02227065]
Iteration #3: [0.78885538 0.02229109]
Simple iteration root: [0.78885541 0.02229102]
Deficiency after: [-2.06040962e-10 -2.04281037e-12]
Number of iterations: 3


Starting point: [3.81313131 3.95685389]
Deficiency before: [ 0.         -0.34209107]
C vector: [0.39678845 0.05810901 0.11621802 0.16346828]
Iteration #2: [3.79325274 3.90093285]
Iteration #3: [3.79281652 3.8999033 ]
Iteration #4: [3.79279978 3.89986497]
Simple iteration root: [3.79279915 3.89986352]
Deficiency after: [-1.28808213e-08 -3.22813787e-07]
Number of iterations: 4


[array([0.78885541, 0.02229102]), array([3.79279915, 3.89986352])]
```

5. Выводы

- Метод простой итерации сошёлся для обоих начальных приближений: для меньшего корня за 3 итерации с точностью $2.06040962e{-}10$ при требуемой точности $\epsilon = 1e{-}5$, а для большего корня - за 4 итерации с точностью $3.22813787e-07$. Таким образом, примем $k_{SimpleIter} = 4$; $||r_{SimpleIter}||_\infty = 3.22813787e - 07$.

- Высокая скорость сходимости и точность обусловлены выбором представления системы в каноническом виде, а именно условием равенства нулю нормы матрицы Якоби.