БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ
КАФЕДРА ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ

Лабораторная работа 5

Метод Гаусса-Зейделя 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**

2. Решение системы нелинейных уравнений **1**

3. Листинг программы **1**

4. Вывод программы **3**

5. Выводы **5**

1. Постановка задачи

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

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

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

Как видно из результата выполнения программы, для корней получаем:

$$x_1^0 = (0.78282828, 0.02358178)^T$$
$$r_1 = 0.05050505$$
$$x_2^0 = (3.81313131, 3.95686389)^T$$
$$r_2 = 0.07103867$$

- Построим итерационный процесс метода Гаусса-Зейделя с использованием метода Ньютона для нахождения корней $\mathbf{x}_i$ уравнения $\mathbf{F}(\mathbf{x}) = \mathbf{0}$:

$$x^{k+1,s+1} = x^{k+1,s} - \frac{f(x^{k+1,s}, y^k)}{\frac{\partial f_1}{\partial x}\Big|_{(x^{k+1,s}, y^k)}} \tag{2}$$

$$y^{k+1,s+1} = y^{k+1,s} - \frac{f(x^{k+1}, y^{k+1,s})}{\frac{\partial f_2}{\partial y}\Big|_{(x^{k+1}, y^{k+1,s})}} \tag{3}$$

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))

#SystemGaussZeidel.py

from SystemCommon import *

def gaussZeidel(f, point, outerEps, innerEps):
    def NewtonNonLinear(f, f_prime, x0, epsilon):
        old_x = x0
        new_x = old_x - f(old_x) / f_prime(old_x)
        innerIter = 1
        while (abs(old_x - new_x) >= epsilon):
            old_x = new_x
            new_x = old_x - f(old_x) / f_prime(old_x)
            innerIter += 1
            if innerIter % 50 == 0:
                print("Inner iteration #{0}: {1}".format(innerIter, new_x))
            if innerIter > 200:
                print("Inner iterations diverge, stopping")
                break
        return new_x

    prevPoint = np.copy(point)

    def f_sub1(x):
        return f1(x, prevPoint[1])
```

```
    def f_sub1_prime(x):
        return f1_xprime(x, prevPoint[1])

    def f_sub2(y):
        return f2(prevPoint[0], y)

    def f_sub2_prime(y):
        return f2_yprime(prevPoint[0], y)

    point[0] = NewtonNonLinear(f_sub1, f_sub1_prime, prevPoint[0], innerEps)
    point[1] = NewtonNonLinear(f_sub2, f_sub2_prime, prevPoint[1], innerEps)
    outerIter = 1
    while (infNorm(point - prevPoint) >= outerEps):
        prevPoint = np.copy(point)
        point[0] = NewtonNonLinear(f_sub1, f_sub1_prime, prevPoint[0], innerEps)
        point[1] = NewtonNonLinear(f_sub2, f_sub2_prime, prevPoint[1], innerEps)
        outerIter += 1
        print("Outer iteration #{0}: {1}".format(outerIter, prevPoint))
    return (point, outerIter)


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

    for point in points:
        print("Starting point: " + str(point))
        print("Deficiency before: " + str(f(point[0], point[1])))
        (root, iterNum) = gaussZeidel(f, point, 10 ** -5, 10 ** -4)
        print("Gauss-Zeidel 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]
Outer iteration #2: [0.78282828 0.03435019]
Outer iteration #3: [0.73789243 0.03435019]
Outer iteration #4: [0.73789243 0.12453707]
Outer iteration #5: [0.50092672 0.12453706]
Outer iteration #6: [0.50092674 0.64231256]
Outer iteration #7: [-0.13341304  0.64231251]
Inner iteration #50: 0.02477319950302803
Inner iteration #100: 9.471333614376778
Inner iteration #150: 0.81081306540516
Inner iteration #200: 4.102940911976072
Inner iterations diverge, stopping
Outer iteration #8: [-0.13341299 -3.10043604]
Inner iteration #50: 0.17457316168971415
```

```
Inner iteration #100: -3.800006551545003
Inner iteration #150: 0.5744518296000121
Inner iteration #200: -1.7171939616247336
Inner iterations diverge, stopping
Outer iteration #9: [ 1.55227771 -3.10043601]
Outer iteration #10: [0.78244634 3.00130083]
Outer iteration #11: [-1.45002075  2.4317437 ]
Inner iteration #50: -5.662500625826174
Inner iteration #100: -2.341275086599574
Inner iteration #150: 21.178110974791274
Inner iteration #200: -5.467261043018148
Inner iterations diverge, stopping
Outer iteration #12: [-1.20533158 -3.65801999]
Inner iteration #50: -1.5937065561165737
Inner iteration #100: 3.765079764002724
Inner iteration #150: 2.454574228446361
Inner iteration #200: -2.139315444748457
Inner iterations diverge, stopping
Outer iteration #13: [-1.66800932 -3.56952485]
Inner iteration #50: -6.180554577536984
Inner iteration #100: -0.05656764414575699
Inner iteration #150: -3.127539100530332
Inner iteration #200: 0.9964170578002651
Inner iterations diverge, stopping
Outer iteration #14: [ 0.56738152 -3.73282343]
Inner iteration #50: 4.155432188936307
Inner iteration #100: -1.367224621398623
Inner iteration #150: 4.719866191058152
Inner iteration #200: 56.63500889277552
Inner iterations diverge, stopping
Outer iteration #15: [1042.83009349   -2.65541322]
Outer iteration #16: [28.76977527 23.29680983]
Outer iteration #17: [7.82595192 7.23718158]
Outer iteration #18: [4.80451878 4.83955686]
Outer iteration #19: [4.1111274  4.18214437]
Outer iteration #20: [3.89210801 3.99364391]
Outer iteration #21: [3.82617901 3.92967368]
Outer iteration #22: [3.80345276 3.90994173]
Outer iteration #23: [3.79640545 3.90308659]
Outer iteration #24: [3.79395297 3.9009552 ]
Outer iteration #25: [3.79319001 3.90021285]
Outer iteration #26: [3.79292422 3.89998183]
Outer iteration #27: [3.79284151 3.89990135]
Outer iteration #28: [3.79281269 3.8998763 ]
Gauss-Zeidel root: [3.79280372 3.89986758]
Deficiency after: [-8.72706865e-06 -1.79386595e-05]
Number of iterations: 28


Starting point: [3.81313131 3.95685389]
Deficiency before: [ 0.         -0.34209107]
Outer iteration #2: [3.81313131 3.90600941]
Outer iteration #3: [3.7949989  3.90600941]
Outer iteration #4: [3.7949989  3.90052948]
Outer iteration #5: [3.79303759 3.90052948]
Outer iteration #6: [3.79303759 3.89993568]
Outer iteration #7: [3.79282498 3.89993568]
Outer iteration #8: [3.79282498 3.8998713 ]
Outer iteration #9: [3.79280193 3.8998713 ]
Gauss-Zeidel root: [3.79280193 3.89986432]
Deficiency after: [-6.98076505e-06 -1.16342003e-09]
```

```
Number of iterations: 9


[array([3.79280372, 3.89986758]), array([3.79280193, 3.89986432])]
```

5. Выводы

- Как видно из результата работы программы, метод Гаусса-Зейделя в окрестности меньшего корня не сошёлся по причине расхождения внутренних итераций по формуле метода Ньютона. В окрестности большего корня метод Гаусса-Зейделя сошёлся за $k_{Gauss-Zeidel} = 9$ итераций с невязкой $||r_{Gauss-Zeidel}||_\infty = 6.98076505e-06$.
- Сравним точность и скорость сходимости метода Гаусса-Зейделя с методом простой итерации в окрестности большего корня:

$$\begin{aligned} k_{Gauss-Zeidel} &= 9; \\ k_{SimpleIter} &= 4; \\ ||r_{Gauss-Zeidel}||_\infty &= 6.98076505e-06; \\ ||r_{SimpleIter}||_\infty &= 3.22813787e-07. \end{aligned}$$

  Нетрудно заметить, что метод Гаусса-Зейделя уступает методу простой итерации, что связано со специальным видом канонического представления системы в методе простой итерации. Однако метод Гаусса-Зейделя не требует решения СЛАУ и несколько проще в реализации, что делает его потенциально более применимым к решению систем с количеством уравнений больше двух.