МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«Национальный исследовательский ядерный университет «МИФИ»

(НИЯУ МИФИ)

Институт Интеллектуальных Кибернетических Систем

Кафедра Кибернетики

Лабораторная работа №2:

По курсу «Численные методы»

Вариант 16

Работу выполнил: студент группы Б17-511: Чудновец И.В.

Проверил: Саманчук В.Н.

Москва 2019

Постановка задачи

Решить систему нелинейных уравнений методом Ньютона:

$$\begin{cases} 9xz^2 + yz + 2y^2x^3 = -175 \\ 0{,}5y^3z^2 - 0{,}8x^2y - 5{,}5 = -4{,}2 \\ 0{,}7z^3 - 6{,}3y^3 = 12{,}6 \end{cases}$$

Начальное приближение (0,5; 0,5; 0,5)

$\varepsilon = 5 * 10^{-3}$

Методика решения

Для решения задачи была написана программа на языке Python, в которой реализован

Алгоритм решения системы нелинейных уравнений методом Ньютона.

Теоретическая справка

Дана система n нелинейных уравнений с n неизвестными:

$$\begin{cases} f_1(x_1, x_2, \cdots, x_n) \\ f_2(x_1, x_2, \cdots, x_n) \\ \vdots \\ f_n(x_1, x_2, \cdots, x_n) \end{cases} \quad (1)$$, где $f_i(x_1, x_2, \cdots, x_n): R^n \to R, i = 1, \cdots, n$ нелинейные функции,

определенные и непрерывные в некоторой области $G \subset R^n$, или в векторном виде (где $x = (x_1, \cdots, x_n)^T,\ F(x) = [f_1(x), \cdots, f_n(x)]^T$)

$F(x) = 0$ (2)

Требуется найти такой вектор $x_* = (x_{*1}, \cdots, x_{*n})^T$, который при подстановке в систему (1) превращает каждое уравнение в верное числовое равенство.

Метод Ньютона для решения нелинейных систем уравнений:

Метод используется для решения систем вида (1) или (2). Формула для нахождения решения: $x^{(k+1)} = x^{(k)} - W^{-1}\left(x^{(k)}\right) * F\left(x^{(k)}\right),\ k = 0, 1, 2, \ldots$ (3), где $W(x) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}$ - матрица Якоби.

Так как процесс вычисления обратной матрицы является трудоемким, преобразуем (3) следующим образом:

$$\Delta x^{(k)} = -W^{-1}\left(x^{(k)}\right) * F\left(x^{(k)}\right), \qquad k = 0, 1, 2, \dots$$

где $\Delta x^{(k)} = x^{(k+1)} - x^{(k)}$ – поправка к приближению $x^{(k)}$.

Умножим последнее выражение слева на матрицу Якоби $W\left(x^{(k)}\right)$:

$$W\left(x^{(k)}\right)\Delta x^{(k)} = -W\left(x^{(k)}\right)W^{-1}\left(x^{(k)}\right)F\left(x^{(k)}\right) = -F\left(x^{(k)}\right), \qquad k = 0, 1, 2, \dots$$

В результате получена система линейных алгебраических уравнений относительно поправки $\Delta x^{(k)}$. После ее определения вычисляется следующее приближение

$$x^{(k+1)} = x^{(k)} + \Delta x^{(k)}.$$

Алгоритм метода Ньютона для решения нелинейных систем:

1) Задать начальное приближение $x^{(0)}$ и малое положительное число ε (точность). Положить $k = 0$.
2) Решить систему линейных алгебраических уравнений относительно поправки $W\left(x^{(k)}\right)\Delta x^{(k)} = -F\left(x^{(k)}\right)$
3) Вычислить следующее приближение: $x^{(k+1)} = x^{(k)} + \Delta x^{(k)}$.
4) Если $\Delta^{(k+1)} = \max\limits_i \left|x_i{}^{(k+1)} - x_i{}^{(k)}\right| \leq \varepsilon$, процесс закончить и положить $x_* \cong x^{(k+1)}$. Если $\Delta^{(k+1)} > \varepsilon$, то положить $k = k + 1$ и перейти к пункту 2.

Решение задачи

newtonsmethod.py

```
from sympy import *

ROUNDING = 10

def _stop(delta, E):
    m = max(map(lambda x: abs(x), delta))
    return m <= E


def _create_w(F, list_var):
    m = []
    for eq in F:
        empty = []
        for var_ in list_var:
            empty.append(eq.diff(var_))
        m.append(empty)
    return Matrix(m)


def _solve_linear_system(system, list_var):
    solution = solve_linear_system(system, *list_var)
    return Matrix([N(solution[var_], ROUNDING) for var_ in list_var])


class NewtonsMethod:
    @staticmethod
    def solve(x_0, F, *var_list, E=1e-3):
```

```
        x_k = Matrix(x_0)
        delta = []
        for i in range(1, len(var_list) + 1):
            var(f'dx{i}')
            delta.append(eval(f'dx{i}'))
        print('Матрица системы нелинейных уравнений F:')
        display(F)
        w = _create_w(F, var_list)
        print('Матрица Якоби W:')
        display(w)
        i = 0
        print(f'x{i} = {x_0}')
        w_k = w
        f_k = F
        for var_ in zip(var_list, x_k):
            w_k = w_k.subs(var_[0], var_[1])
            f_k = f_k.subs(var_[0], var_[1])
        delta_k = _solve_linear_system(w_k.row_join(-f_k), delta)
        x_k_1 = x_k + delta_k
        i += 1
        print(f'x{i} = {list(x_k_1)}')
        while not _stop(delta_k, E):
            x_k = x_k_1
            w_k = w
            f_k = F
            for var_ in zip(var_list, x_k):
                w_k = w_k.subs(var_[0], var_[1])
                f_k = f_k.subs(var_[0], var_[1])
            delta_k = _solve_linear_system(w_k.row_join(-f_k), delta)
            x_k_1 = x_k + delta_k
            i += 1
            print(f'x{i} = {list(x_k_1)}')
        return list(x_k_1)

    @staticmethod
    def residual(F, *var_list, solution=None):
        if not solution:
            print('ERROR: Give Solution!')
        residual = F
        for var_ in zip(var_list, solution):
            residual = residual.subs(var_[0], var_[1])
        return residual
```

test_Newton.ipynb

```
from sympy import *
init_printing()
---
# Вариант 16
from newtonsmethod import NewtonsMethod
var('x y z')
var_s = [x, y, z]
x_0 = [0.5, 0.5, 0.5]
F = Matrix([9 * x * z ** 2 + y * z + 2 * y ** 2 * x ** 3 + 175,
           0.5 * y ** 3 * z ** 2 -0.8 * x ** 2 * y - 5.5 + 4.2,
          0.7 * z ** 3 - 6.3 * y ** 3 - 12.6])
solution = NewtonsMethod.solve(x_0, F, *var_s, E=5e-3)
print('\nНевязка решения:')
display(NewtonsMethod.residual(F, *var_s, solution=solution))
```

Условные обозначения:

--- - означает, что код в Jupyter Notebook разделён в разные ячейки

Результат работы

```
Матрица системы нелинейных уравнений F:
```

$$\begin{bmatrix} 2x^3y^2 + 9xz^2 + yz + 175 \\ -0.8x^2y + 0.5y^3z^2 - 1.3 \\ -6.3y^3 + 0.7z^3 - 12.6 \end{bmatrix}$$

```
Матрица Якоби W:
```

$$\begin{bmatrix} 6x^2y^2 + 9z^2 & 4x^3y + z & 18xz + y \\ -1.6xy & -0.8x^2 + 1.5y^2z^2 & 1.0y^3z \\ 0 & -18.9y^2 & 2.1z^2 \end{bmatrix}$$

```
x0 = [0.5, 0.5, 0.5]
x1 = [-6.12586982944049, -5.74505482765380, -30.3721601155121]
x2 = [-4.24889265025558, -5.05862808888196, -20.6372304558754]
x3 = [-2.89035047938523, -4.40828050046548, -14.3042190858396]
x4 = [-1.91539470727002, -3.77783904989337, -10.2239671821008]
x5 = [-1.27519044428482, -3.17406293382373, -7.56407812196994]
x6 = [-0.983248347434710, -2.62524083315657, -5.74480606865836]
x7 = [-1.03893714735477, -2.15747196130178, -4.41358810763631]
x8 = [-1.41270433400950, -1.78605785673426, -3.38221869227709]
x9 = [-2.03719692944014, -1.53507348533094, -2.59458723453281]
x10 = [-2.54168533318898, -1.44510663301571, -2.16703969836817]
x11 = [-2.61670099623916, -1.44496232525595, -2.09436103372173]
x12 = [-2.61644177861225, -1.44533274803424, -2.09336754937583]
Невязка решения:
```

$$\begin{bmatrix} -2.9037593975545 \cdot 10^{-5} \\ 1.71001984661245 \cdot 10^{-6} \\ -5.92765680984542 \cdot 10^{-7} \end{bmatrix}$$

Заключение

В работе требовалось решить систему нелинейных уравнений методом Ньютона. Для решения данной задачи была написана программа. Вычисления происходили при начальном приближении (0,5; 0,5; 0,5). $\varepsilon = 5 * 10^{-3}$. Ответ: (-2.61644177861225, -1.44533274803424, -2.09336754937583)