МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Вятский государственный университет»
(ФГБОУ ВО «ВятГУ»)
Факультет автоматики и вычислительной техники
Кафедра электронных вычислительных машин

ЧИСЛЕННЫЕ МЕТОДЫ РЕШЕНИЯ СИСТЕМ УРАВНЕНИЙ

Отчет по лабораторной работе №2 дисциплины
«Вычислительная математика»

Выполнил студент группы ИВТ-21 ______________________________/Рзаев А.Э./
Проверил преподаватель ________________________________/Архангельский В.В./

2016 г.

1 Постановка задачи

1. Решить систему линейных уравнений 4-го порядка методом Гаусса с точностью e=0,001.

Уравнения системы:

1,00*x1+0,55*x2-0,13*x3+0,34*x4=0,13
0,13*x1-0,17*x2+0,33*x3+0,17*x4=0,11
0,11*x1+0,18*x2-0,22*x3-0,11*x4=1,00
0,13*x1-0,12*x2+0,21*x3+0,22*x4=0,18

2. Решить систему линейных уравнений 4-го порядка с точностью e=0,0001:

- методом простой итерации.

Уравнения системы:

x1=0,14*x1+0,23*x2+0,18*x3+0,17*x4-1,42
x2=0,12*x1-0,14*x2+0,08*x3+0,09*x4-0,83
x3=0,16*x1+0,24*x2-0,35*x4+1,21
x4=0,23*x1-0,08*x2+0,55*x3+0,25*x4+0,05

3. Решить систему линейных уравнений 3-го порядка методом обратной матрицы с точностью e=0,001.

Уравнения системы:

2*x1+2*x2+3*x3=1
x1+x2+3*x3=2
2*x1+x2+2*x3=-1

4. Решить систему нелинейных уравнений 2-го порядка методом Ньютона с точностью e=0,001.

Уравнения системы:

x-exp(y-0,4)-3,5=0
y-sin(x)=0

5. Проверить результаты с помощью системы MathCad.

2 Ход выполнения

Метод Гаусса

Состоит из двух этапов:
1) Прямой ход;
2) Обратный ход.

На этапе прямого хода систему уравнений записывают в виде матрицы коэффициентов перед неизвестными, которую затем с помощью элементарных преобразований приводят в треугольному (ступенчатому) виду (либо устанавливают, что система несовместна). А именно, среди элементов первого столбца матрицы выбирают ненулевой, перемещают его на крайнее верхнее положение перестановкой строк и вычитают получившуюся после перестановки первую строку из остальных строк, помножив её на величину, равную отношению первого элемента каждой из этих строк к первому элементу первой строки, обнуляя тем самым столбец под ним. После того, как указанные преобразования были совершены, первую строку и первый столбец мысленно вычёркивают и продолжают пока не останется матрица нулевого размера. Если на какой-то из итераций среди элементов первого столбца не нашёлся ненулевой, то переходят к следующему столбцу и проделывают аналогичную операцию.

На этапе обратного хода осуществляется выражение всех получившихся базисных переменных через небазисные и построение фундаментальной системы решений, либо, если все переменные являются базисными, то выражение в численном виде единственного решения системы линейных уравнений. Эта процедура начинается с последнего уравнения, из которого выражают соответствующую базисную переменную (а она там всего одна) и подставляют в предыдущие уравнения, и так далее, поднимаясь по «ступенькам» наверх.

Решение первой системы уравнений методом Гаусса в системе MathCad и в написанной программе приведено на рисунках 1 и 2.

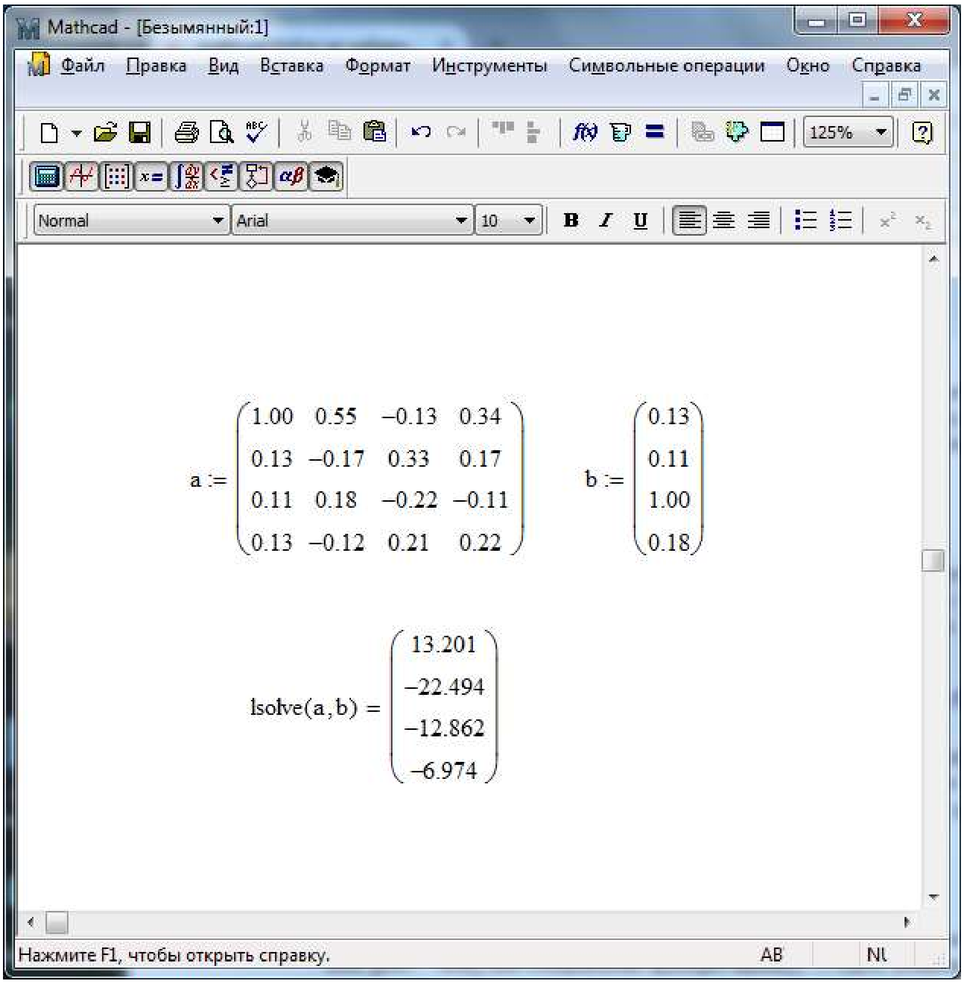

Рисунок 1 – Решение первой системы в MathCad.

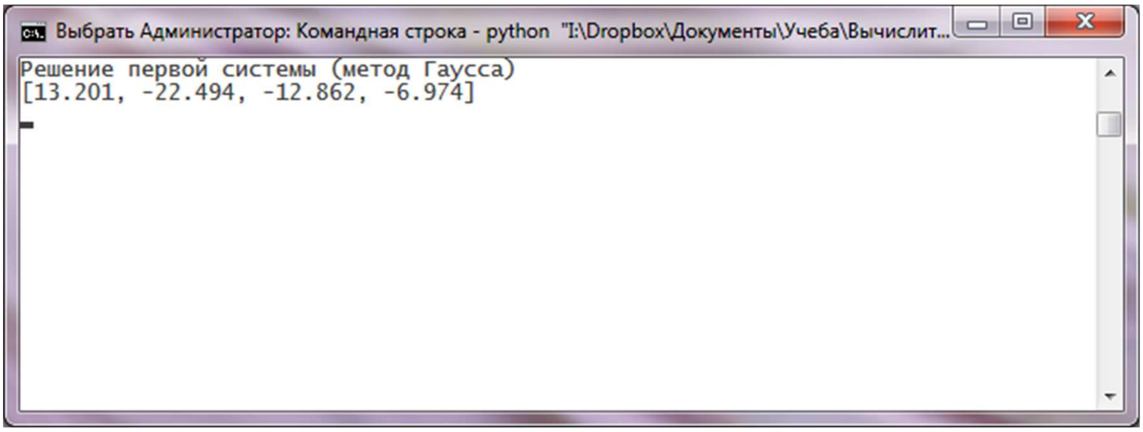

Рисунок 2 – Решение первой системы в программе.

Метод простых итераций

Для того, чтобы построить итеративную процедуру метода Якоби, необходимо провести предварительное преобразование системы уравнений $Ax = b$ к итерационному виду $x = Bx + g$. Оно может быть осуществлено по одному из следующих правил:

- $B = E - D^{-1}A, g = D^{-1}b$

- $B = -D^{-1}(L + D), g = D^{-1}b$

где в принятых обозначениях D означает матрицу, у которой на главной диагонали стоят соответствующие элементы матрицы A, а все остальные нули; тогда как матрицы U и L содержат верхнюю и нижнюю треугольные части A, на главной диагонали которых нули, E—единичная матрица.

Формула для расчёта приближения выглядит следующим образом:

$$x_i^{(k+1)} = \sum_{j=1}^{i-1} b_{ij} x_j^{(k)} + \sum_{j=1}^{n} b_{ij} x_j^{(k)} + g_i \,.$$

Условия остановки процесса итерации:

$$\max_{i=1..n}\left(\left|x_i^{(k)} - x_i^{(k-1)}\right|\right) \le \varepsilon \,.$$

Решение второй системы уравнений в MathCad и в программе представлено на рисунках 3 и 4.

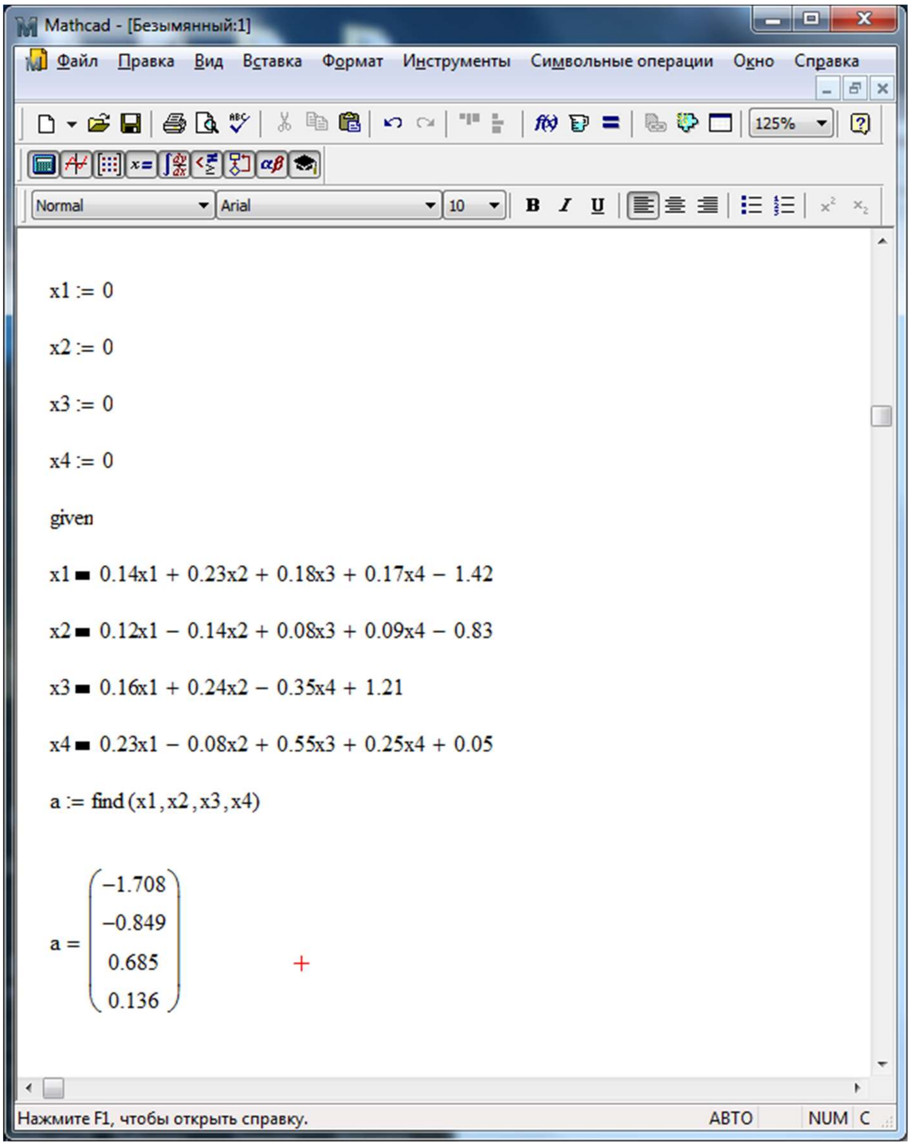

Рисунок 3 – Решение второй системы уравнений в MathCad.

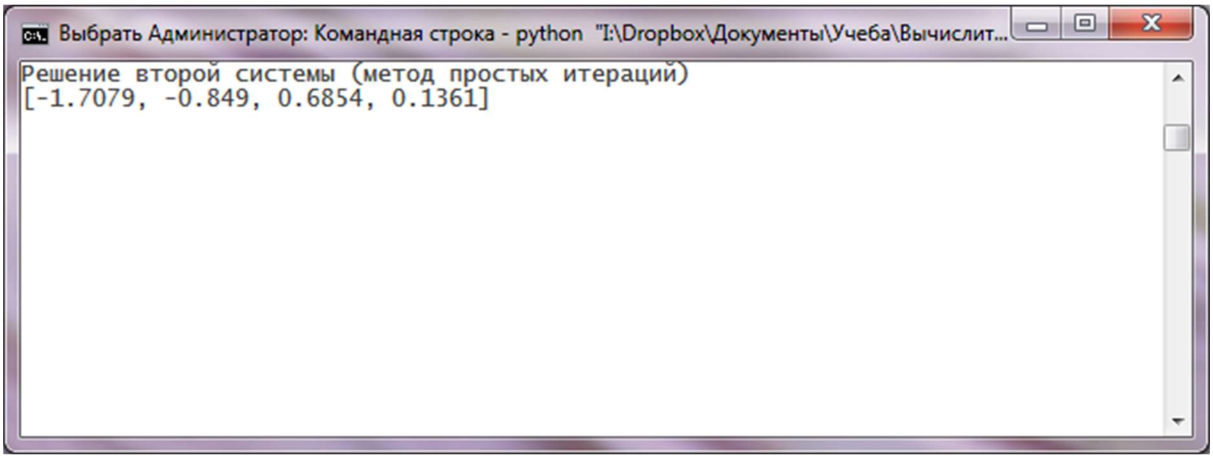

Рисунок 4 – Решение системы в программе.

Метод обратной матрицы

Суть метода состоит в умножении левой и правой части матричной записи уравнения на матрицу, обратную матрице коэффициентов перед неизвестными (основной матрице). При этом в левой части получается вектор, содержащий неизвестные, а в правой части значения этого вектора.

$$AX = B \mid \times A^{-1}$$

$$A^{-1}AX = A^{-1}B$$

$$X = A^{-1}B$$

Само собой разумеется, что на этот метод накладывается ограничение $|A| \neq 0$, в противном случае найти обратную матрицу будет невозможно. Обратная матрица получается, как транспонированная матрица алгебраических дополнений основной матрицы, умноженная на $1/|A|$.

Решение третьей системы уравнений в MathCad и в программе приведено на рисунках 5 и 6.

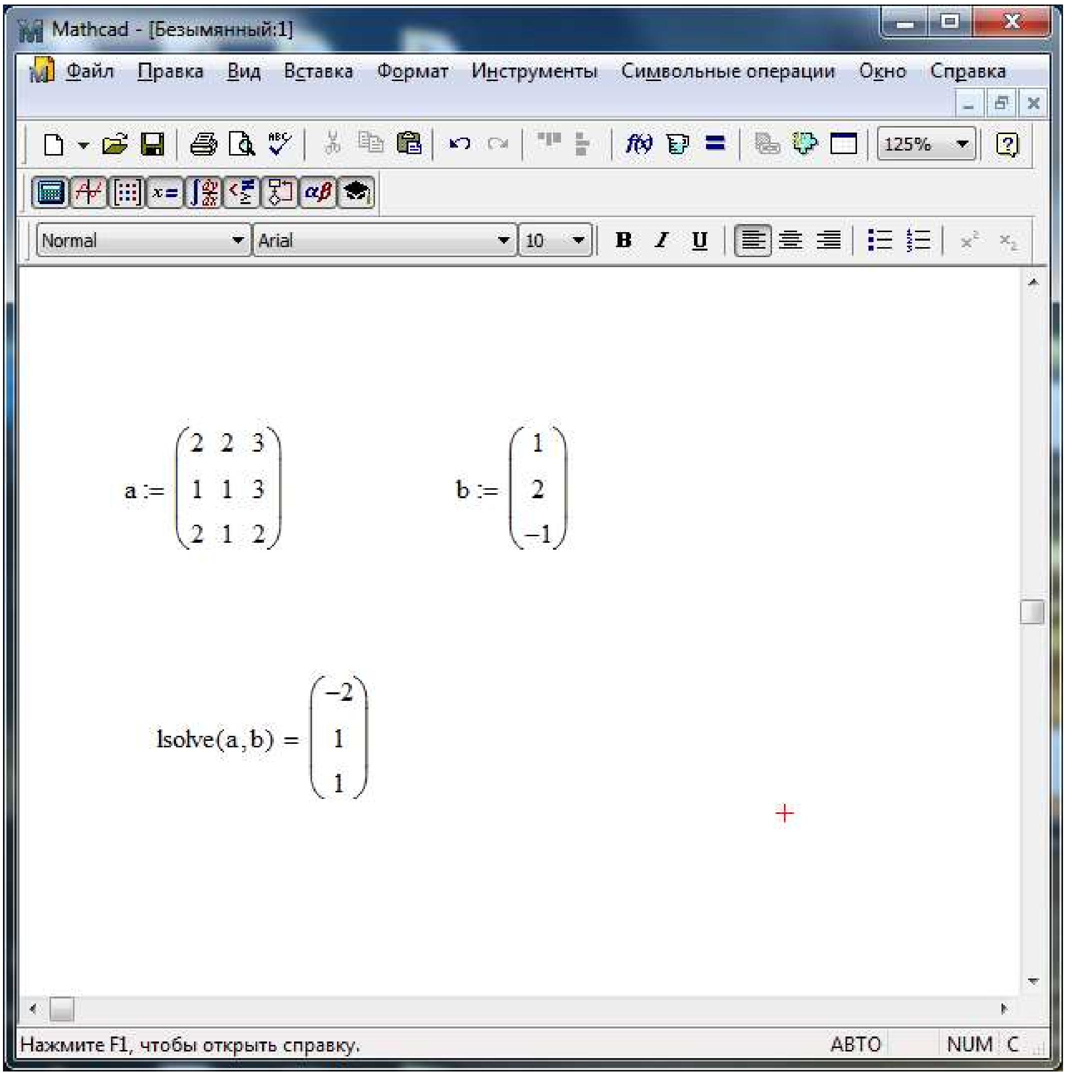

Рисунок 5 – Решение третьей системы уравнений в MathCad.

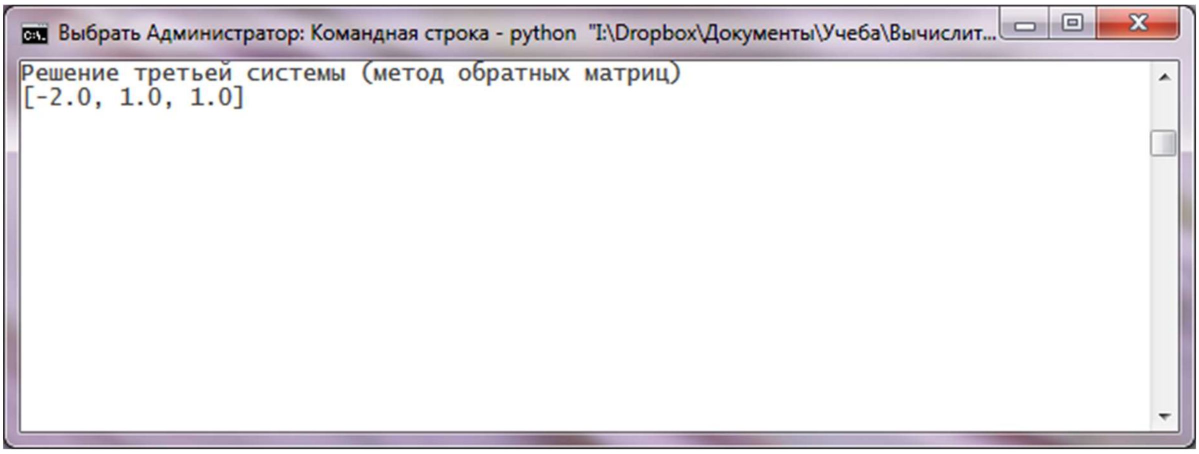

Рисунок 6 – Решение системы в программе.

Метод Ньютона

Метод Ньютона для решения систем нелинейных уравнений является многомерным случаем метода Ньютона для решения нелинейных уравнений. Для системы из вдух уравнений каждое последующиее приближение получается по формулам:

$$x^{(k+1)} = x^{(k)} - \frac{d_x^{(k)}}{d^{(k)}}$$

$$y^{(k+1)} = y^{(k)} - \frac{d_y^{(k)}}{d^{(k)}}$$

$$d^{(k)} = \begin{vmatrix} f'_x(x^{(k)}, y^{(k)}) & f'_y(x^{(k)}, y^{(k)}) \\ g'_x(x^{(k)}, y^{(k)}) & g'_y(x^{(k)}, y^{(k)}) \end{vmatrix}$$

$$d_x^{(k)} = \begin{vmatrix} f(x^{(k)}, y^{(k)}) & f'_y(x^{(k)}, y^{(k)}) \\ g(x^{(k)}, y^{(k)}) & g'_y(x^{(k)}, y^{(k)}) \end{vmatrix}$$

$$d_y^{(k)} = \begin{vmatrix} f'_x(x^{(k)}, y^{(k)}) & f(x^{(k)}, y^{(k)}) \\ g'_x(x^{(k)}, y^{(k)}) & g(x^{(k)}, y^{(k)}) \end{vmatrix}$$

Итерация продолжается до тех пор, пока не будет выполнено условие $m = \sqrt{(x^{(k)} - x^{(k-1)})^2 + (y^{(k)} - y^{(k-1)})^2} \le \varepsilon$.

Решение четвёртой системы уравнений в MathCad и программе приведено на рисунках 7 и 8.

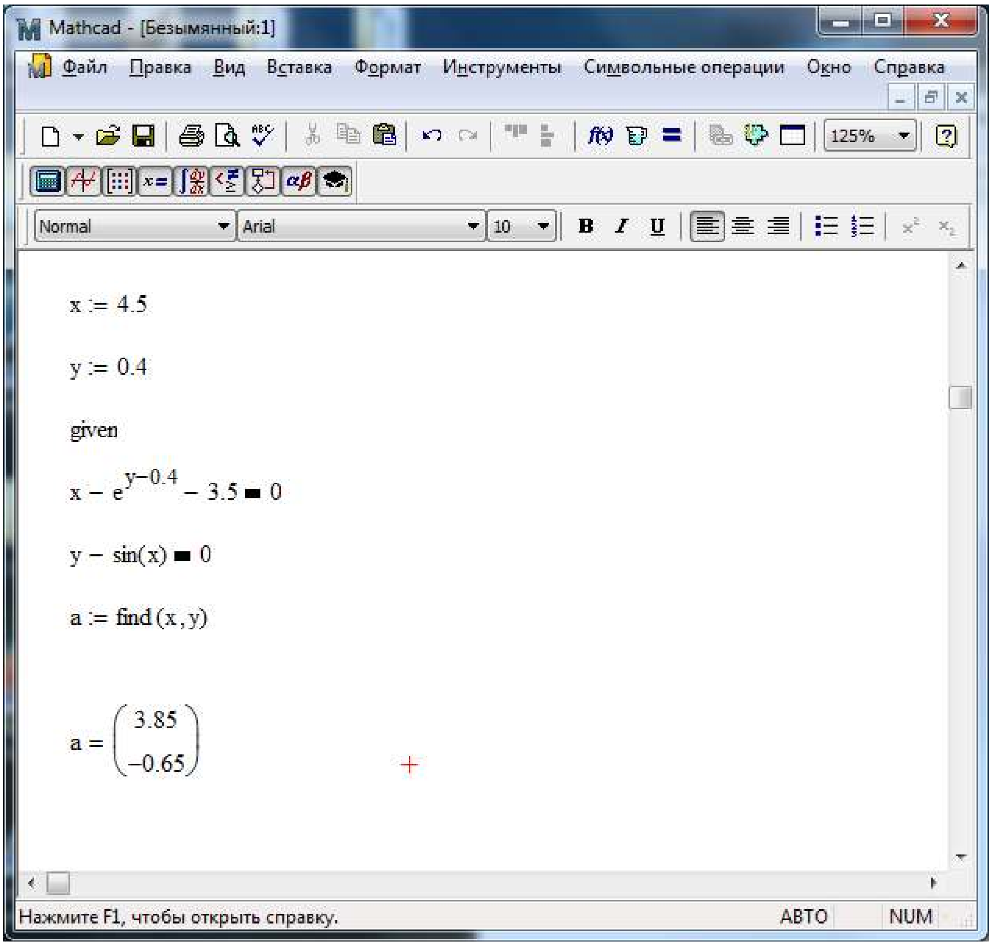

Рисунок 7 – Решение четвёртой системы уравнений в MathCad.

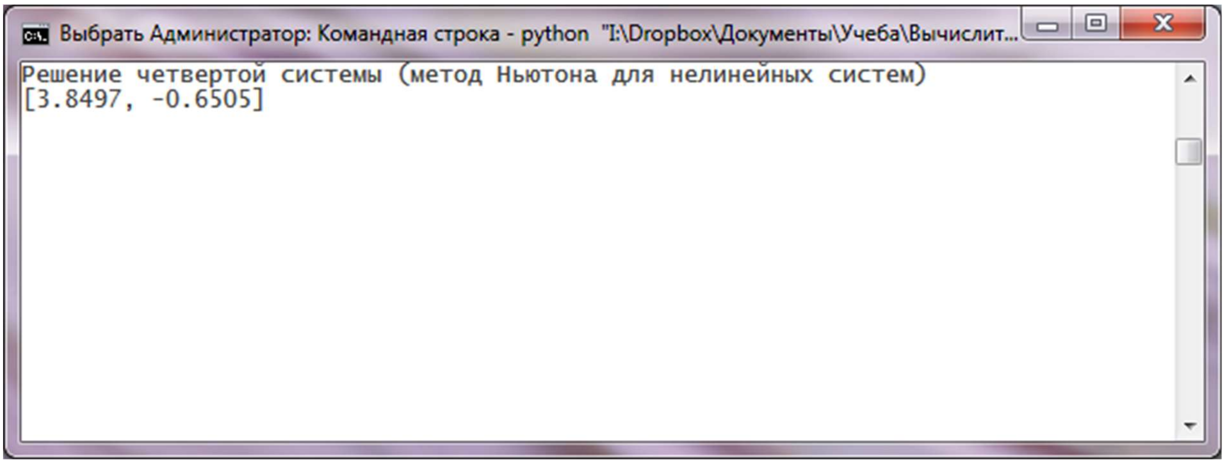

Рисунок 8 – Решение системы уравнений в программе.

3 Вывод

В ходе данной лабораторной работы были изучены численные методы решения систем линейных и нелинейных уравнений, изученых их преимущества и недостатки. Для всех приведённых в задании методов была написана программа, реализующая их, листинг программы приведён в приложении А.

Приложение А
(обязательное)
Листинг кода программы

```
from math import *


def simple_iterations(a, b, eps):
    x0 = b[:]
    x1 = b[:]
    n = len(b)

    while True:
        for i in range(n):
            x1[i] = b[i]
            for j in range(n):
                x1[i] += x0[j] * a[i][j]

        if all((abs(v2 - v1) <= eps for v2, v1 in zip(x1, x0))):
            break
        else:
            x0 = x1[:]

    check = [0] * n

    for i in range(n):
        check[i] = b[i]
        for j in range(n):
            if i == j:
                check[i] += x0[j] * (a[i][j] - 1)
            else:
                check[i] += x0[j] * a[i][j]

    return [round(x, int(-log10(eps))) for x in x0], all((abs(c) <= eps for c in
check))


def reverse_matrix(a, b, eps):
    def det(m):
        return m[0][0] * (m[1][1] * m[2][2] - m[2][1] * m[1][2]) -\
               m[0][1] * (m[1][0] * m[2][2] - m[2][0] * m[1][2]) +\
               m[0][2] * (m[1][0] * m[2][1] - m[2][0] * m[1][1])

    n = len(b)
    d0 = det(a)
    d = [0] * n

    for i in range(n):
        for j in range(n):
            a[j][i], b[j] = b[j], a[j][i]
        d[i] = det(a)
        for j in range(n):
            a[j][i], b[j] = b[j], a[j][i]

    return ([round(di / d0, int(-log10(eps))) for di in d], True) if abs(d0) > eps
else ([-1] * n, False)


def gauss(a, b, eps):
    n = len(b)
    for i in range(n):
        z = i
        while z < n and a[z][i] == 0:
            z += 1
        a[i], a[z] = a[z], a[i]
```

```
        val = a[i][i]
        for j in range(n):
            a[i][j] /= val
        b[i] /= val

        for j in range(i + 1, n):
            val = a[j][i]
            for k in range(i, n):
                a[j][k] -= a[i][k] * val
            b[j] -= b[i] * val

    x = [0] * n
    for i in range(n - 1, -1, -1):
        x[i] = b[i]
        for j in range(i + 1, n):
            x[i] -= a[i][j] * x[j]

    return [round(i, int(-log10(eps))) for i in x], True


def newton(funcs, eps, s):
    f1, f2, f1x, f1y, f2x, f2y = funcs
    x0, y0 = s

    while True:
        a = [
            [f1x(x0, y0), f1y(x0, y0)],
            [f2x(x0, y0), f2y(x0, y0)]
        ]

        b = [-f1(x0, y0), -f2(x0, y0)]

        d = (a[0][0] * a[1][1]) - (a[0][1] * a[1][0])

        dx = (b[0] * a[1][1] - b[1] * a[0][1]) / d
        dy = (a[0][0] * b[1] - b[0] * a[1][0]) / d

        if abs(dx) <= eps and abs(dy) <= eps:
            break
        else:
            x0 += dx
            y0 += dy

    return [round(i, int(-log10(eps))) for i in [x0, y0]], True
```