Лабораторная работа №7  
Гайворонская Екатерина Александровна   
010304 КМСб-о23

ЛАБОРАТОРНАЯ РАБОТА 7.

Применение инструментария Питон

Для решения вычислительных задач и обработки данных

Задачи:

1. Научиться разрабатывать подпрограммы на языке Python для решения некоторых задач вычислительной математики.

Вариант p = 0 q = 1

**Порядок выполнения работы**

**Задача1.** Разработайте программу для решения СЛАУ методом Крамера.

Порядок решения:

1. Метод Крамера.
2. Создам подпрограмму det\_all для поиска общего определителя
3. Создадим подпрограммы det\_x, det\_y, det\_z для поиска определителей этих матриц
4. Найдем значения x, y, z используя метод Крамера

Программа для решения задачи:

import copy

def det\_all(matrix):

    A11 = matrix[1][1] \* matrix[2][2] - matrix[2][1] \* matrix[1][2]

    A12 = matrix[1][0] \* matrix[2][2] - matrix[2][0] \* matrix[1][2]

    A13 = matrix[1][0] \* matrix[2][1] - matrix[2][0] \* matrix[1][1]

    d = matrix[0][0] \*  A11 - matrix[0][1] \* A12 + matrix[0][2] \* A13

    return d

def det\_x(matrix, constants):

    copy\_matryx = copy.deepcopy(matrix)

    b1 = constants[0]

    b2 = constants[1]

    b3 = constants[2]

    copy\_matryx[0][0], copy\_matryx[1][0], copy\_matryx[2][0] = b1,  b2,  b3

    matrix\_x = det\_all(copy\_matryx)

    return matrix\_x

def det\_y(matrix, constants):

    copy\_matryx = copy.deepcopy(matrix)

    b1 = constants[0]

    b2 = constants[1]

    b3 = constants[2]

    copy\_matryx[0][1], copy\_matryx[1][1], copy\_matryx[2][1] = b1,  b2,  b3

    matrix\_y = det\_all(copy\_matryx)

    return matrix\_y

def det\_z(matrix, constants):

    copy\_matryx = copy.deepcopy(matrix)

    b1 = constants[0]

    b2 = constants[1]

    b3 = constants[2]

    copy\_matryx[0][2], copy\_matryx[1][2], copy\_matryx[2][2] = b1,  b2,  b3

    matrix\_z = det\_all(copy\_matryx)

    return matrix\_z

matrix = [[7, 2, 3], [10, -6, 4], [5, -7, 3]]

constants = [15, 30, 21]

x = det\_x(matrix, constants) / det\_all(matrix)

y = det\_y(matrix, constants) / det\_all(matrix)

z = det\_z(matrix, constants) / det\_all(matrix)

print('x =', x, 'y =', y, 'z =', z)

Результаты выполнения программы:



**Задача2.**

1)Дополнить полученную программу процедурой перестановки строк. Добавить возможность задания коэффициентов системы уравнений с клавиатуры для произвольного значения n.

2) Решить методом Гаусса систему уравнений

Порядок решения 1-ой задачи:

1. Создаем подпрограмму под названием gaussian\_elimination
2. Приводим к общему знаменателю и делим строку
3. Перемножаем и отнимаем строки для обнуления коэффициентов
4. Подставляем полученное значение для поиска оставшихся
5. Вводим длину массива и заполняем массив данными

Программа для решения задачи:

def gaussian\_elimination(a):

    n = len(a)

    x = [0] \* n

    # обратный ход

    for k in range(n):

        # Проверка на нулевой ведущий элемент и перестановка строк

        if a[k][k] == 0:

            for i in range(k+1, n):

                if a[i][k] != 0:

                    a[k], a[i] = a[i], a[k]

                    break

        # Нормализация строки(приводим к общему знаменателю)

        p = a[k][k] #общий знаменатель

        for j in range(k, n+1):

            a[k][j] /= p

        # Обнуление столбца    ниже текущей строки мы перемножаем строку

        #с единичной строкой,для того что бы потом их отнять и получить 0

        for i in range(k+1, n):

            s = a[i][k]

            for j in range(k, n+1):

                a[i][j] -= s \* a[k][j]

    # Обратный ход

    # Мы получаем значение последней переменной СЛАУ и далее

    # подставляем ее в оставшиеся уровнения тем самы рассчитав оставшиеся переменные

    x[n-1] = a[n-1][n] / a[n-1][n-1]

    for i in range(n-2, -1, -1):

        s = a[i][n]

        for j in range(i+1, n):

            s -= a[i][j] \* x[j]

        x[i] = s

    return x

    # Ввод коэффициентов системы уравнений

n = int(input("Введите количество уравнений (n): "))

a = []

for i in range(n):

    row = list(map(float, input(f"Введите коэффициенты для уравнения {i+1}, разделенные пробелами: ").split()))

    a.append(row)

    # Решение системы уравнений методом Гаусса

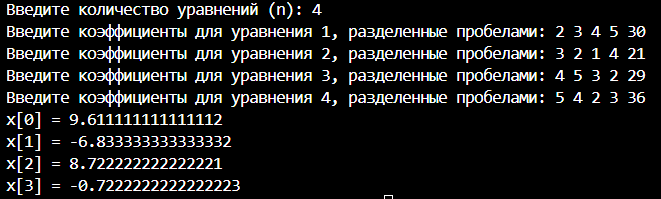
solution = gaussian\_elimination(a)

    # Вывод решений

for j in range(n):

    print(f'x[{j}] = {solution[j]}')

Результаты выполнения программы:



Порядок решения 2-ой задачи:

Протестируем ранее написанную функцию на моем варианте

Программа для решения задачи:

def gaussian\_elimination(a):

    n = len(a)

    x = [0] \* n

    # обратный ход

    for k in range(n):

        # Проверка на нулевой ведущий элемент и перестановка строк

        if a[k][k] == 0:

            for i in range(k+1, n):

                if a[i][k] != 0:

                    a[k], a[i] = a[i], a[k]

                    break

        # Нормализация строки(приводим к общему знаменателю)

        p = a[k][k] #общий знаменатель

        for j in range(k, n+1):

            a[k][j] /= p

        # Обнуление столбца    ниже текущей строки мы перемножаем строку

        #с единичной строкой,для того что бы потом их отнять и получить 0

        for i in range(k+1, n):

            s = a[i][k]

            for j in range(k, n+1):

                a[i][j] -= s \* a[k][j]

    # Обратный ход

    # Мы получаем значение последней переменной СЛАУ и далее

    # подставляем ее в оставшиеся уровнения тем самы рассчитав оставшиеся переменные

    x[n-1] = a[n-1][n] / a[n-1][n-1]

    for i in range(n-2, -1, -1):

        s = a[i][n]

        for j in range(i+1, n):

            s -= a[i][j] \* x[j]

        x[i] = s

    return x

a = [[7, 2, 3, 15], [10, -6, 4, 30], [5, -7, 3, 21]]

n = 3

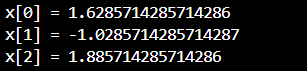
solution = gaussian\_elimination(a)

    # Вывод решений

for j in range(n):

    print(f'x[{j}] = {solution[j]}')

Результаты выполнения программы:



**Задача3.** 1) Напишите программу, которая вычисляет интеграл методом трапеций и методом Симпсона для n = 12. Сравните результаты с точечным значением интеграла.

2) Найти площадь фигуры, ограниченной линиями x = a = 1, x = b = 9, y = 0, y = f(x), используя методы прямоугольников, трапеций и Симпсона. Найти точечное значение интеграла. Сравнить полученные значения. Какой из методов окажется точнее?

Порядок решения 1):

Метод Симпсона(метод парабол) – способ при котором график подынтегральной функции приближается не ломанной линией, а маленькими параболами.

1. Метод Трапеции - вычисления происходят по формуле
2. Метод Симпсона – вычисления происходят по формуле
3. Создаем продпрограммы trapezoidal\_rul, simpsons\_rule.
4. Приведенные ниже шаги относятся к обоим программам

* Разбиение интервала
* Вычисление шага
* Итерация по точкам
* Вычисление значений функций

Программа для решения задачи:

def trapezoidal\_rule(a, b, func, n = 12):

    h = (b - a) / n #Вычисляем шаг интегрирования

    integral = 0.5 \* (func(a) + func(b)) #вычисления части формулы

    for i in range(1, n):

        integral += func(a + i \* h)

    integral \*= h # Умножаем значение функции на шаг

    return integral

def simpsons\_rule(a, b, func, n = 12):

    # Проверяем, что количество разбиений четное

    if n % 2 != 0:

        print("Количество разбиений должно быть четным")

    else:

        h = (b - a) / n  # Вычисляем шаг интегрирования

        x = a

        integral = func(a) + func(b)  # функция в граничных точках

        # Суммируем значения функции для точек с четными и нечетными индексами

        for i in range(1, n):

            # Определяем коэффициент для текущего значения функции

            if i % 2 == 0:

                coefficient = 2

            else:

                coefficient = 4

            x += h

            integral += coefficient \* func(x)  # Добавляем значение функции, умноженное на соответствующий коэффициент

        integral \*= h / 3  # Умножаем сумму значений функции на шаг и на 1/3

        return integral

def my\_function(x):

    return (3 \* x) / (x \*\* 2 + 1)

my\_integral = trapezoidal\_rule(0, 4, my\_function)

simpson\_integral = simpsons\_rule(0, 4, my\_function)

print("Приближенное значение интеграла(метод трапеции):", my\_integral)

print("Приближенное значение интеграла(Метод Симпсона):", simpson\_integral)

Результаты выполнения программы:



Порядок решения 2):

Дополним раннее написанный мной код

Метод прямоугольников – Отрезок делится на равные части. После чего происходят вычисления.

Программа для решения задачи:

import math

def trapezoidal\_rule(a, b, func, n = 12):

    h = (b - a) / n #Вычисляем шаг интегрирования

    integral = 0.5 \* (func(a) + func(b)) #вычисления части формулы

    for i in range(1, n):

        integral += func(a + i \* h)

    integral \*= h # Умножаем значение функции на шаг

    return integral

def simpsons\_rule(a, b, func, n = 12):

    # Проверяем, что количество разбиений четное

    if n % 2 != 0:

        print("Количество разбиений должно быть четным")

    else:

        h = (b - a) / n  # Вычисляем шаг интегрирования

        x = a

        integral = func(a) + func(b)  # функция в граничных точках

        # Суммируем значения функции для точек с четными и нечетными индексами

        for i in range(1, n):

            # Определяем коэффициент для текущего значения функции

            if i % 2 == 0:

                coefficient = 2

            else:

                coefficient = 4

            x += h

            integral += coefficient \* func(x)  # Добавляем значение функции, умноженное на соответствующий коэффициент

        integral \*= h / 3  # Умножаем сумму значений функции на шаг и на 1/3

        return integral

def rectangle\_rule(a, b, func, n = 12):

    h = (b - a) / n  # Вычисляем шаг интегрирования

    integral = 0

    x = a #идем от этой точки

    for i in range(n):

        integral += func(x) \* h  # Площадь прямоугольника - значение функции \* ширина прямоугольника

        x += h  # Переходим к следующему прямоугольнику

    return integral

def my\_function(x):

    return math.sqrt(6 \* x - 5)

my\_integral = trapezoidal\_rule(1, 9, my\_function)

simpson\_integral = simpsons\_rule(1, 9, my\_function)

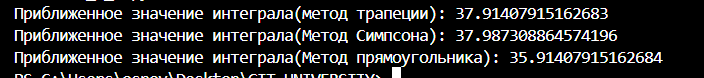
rectangle\_integral = rectangle\_rule(1, 9, my\_function)

print("Приближенное значение интеграла(метод трапеции):", my\_integral)

print("Приближенное значение интеграла(Метод Симпсона):", simpson\_integral)

print("Приближенное значение интеграла(Метод прямоугольника):", rectangle\_integral)

Результаты выполнения программы:



Если мы увеличим количество точек разбиения на отрезке, погрешность во всех методах уменьшится