МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Костромской государственный университет»

(КГУ)

ИАСТ

Кафедра автоматизированных систем и технологий

09.03.02

Направление подготовки/Специальность Информационные системы и технологии

Дисциплина Численные методы

# Лабораторная №4.

# Решение СНАУ (Вариант 19).

Выполнил студент

Копосов Лев Владимирович

Группа 22-ИСбо-1б

Проверил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подпись преподавателя \_\_\_\_\_\_\_\_\_\_\_\_

Кострома

**Постановка задачи.**

Решить заданную систему нелинейных уравнений с помощью 1) метода наискорейшего градиентного спуска и 2) метода Ньютона с точностью до 0.001.

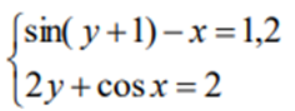
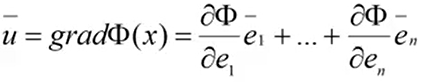
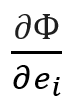
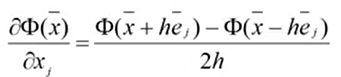
****

рис.1. Задача N19

**Краткая теория используемых методов.**

1. Метод наискорейшего градиентного спуска.

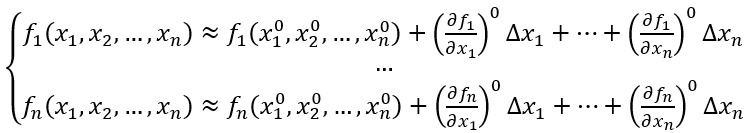
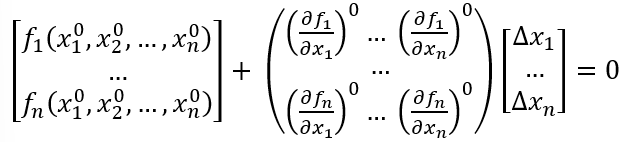
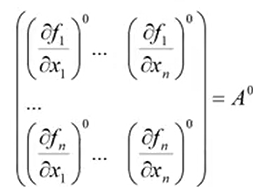
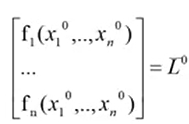
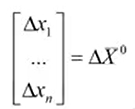
Метод наискорейшего градиентного спуска - это метод оптимизации, используемый для поиска минимума функции.

Основная идея метода заключается в том, чтобы двигаться в направлении антиградиента функции , чтобы постепенно приближаться к её минимуму. Метод наискорейшего градиентного спуска заключается в поиске целевой функции (1.1) и последовательном вычислении градиента функции в данной точке (1.2), где - производная целевой функции, - единичный вектор и находится по формуле(1.5). На каждой итерации выбирается новый шаг  для поиска следующей точки (1.3).

Потом выбираем начальное приближение  и продолжаем процесс итерации, пока не будет достигнута необходимая точность. Решение будет считаться приближенным, если выполняется условие  (1.4), где Ф - целевые функции на k и k+1 шаге, ε - заданная точность.

1. Метод Ньютона.

Метод Ньютона - это итерационный метод для решения систем нелинейных уравнений.

Основная идея метода заключается в построении последовательности приближенных решений, которая сходится к точному решению. Метод Ньютона заключается в задании начальной точки, которую можно найти с помощью методов спуска и преобразовании системы  к векторному виду  (2.1) или , где  - производная функции f и находится по той же формуле (1.5);  (2.2),  (2.3),  (2.4). Затем последовательно вычисляем точку  (2.5) и по этой точке строим новую систему линейных уравнений.

Потом выбираем начальное приближение  и продолжаем процесс итерации, пока не будет достигнута необходимая точность. Решение будет считаться приближенным, если выполняется условие  (2.6), где |ΔXk|- ограниченность длины вектора на k шаге, ε - заданная точность.

**Алгоритм решения поиска минимума функции.**

1. **Метод наискорейшего градиентного спуска.**

Шаг 1. Задаем начальную точку, и точность вычисления ε;

Шаг 2. Вычисляем значение целевой функции по формуле (1.1);

Шаг 3. Вычисляем градиент целевой функции по формуле (1.2);

Шаг 4. Вычисляем значение для h, с выбранным шагом, где целевая функция будет принимать минимальное значение, h принадлежит (0, 1);

Шаг 5. По найденному градиенту находим новую точку (1.3);

Шаг 6. Если модуль разности целевых функций на предыдущей и текущей итерации меньше заданной точности ε (1.4), то выводим решение xk+1, Конец.

Иначе Переходим к Шаг 2;

1. **Метод Ньютона.**

Шаг 1. Задаем шаг h и точность вычисления ε;

Шаг 2. Вычисляем начальную точку x0, используя метод спуска （метод наискорейшего градиентного спуска);

Шаг 3. Преобразуем систему к виду (2.1), вычисляя матрицы (2.2), (2.3);

Шаг 4. Решаем полученную систему и получаем вектор (2.4);

Шаг 5. Вычисляем новую точку xk+1 по формуле (2.5);

Шаг 6. Если условие (2.6) выполняется, то выводим решение xk+1, Конец.

Иначе Переходим к Шаг 3;

**Вывод результата решения задачи.**

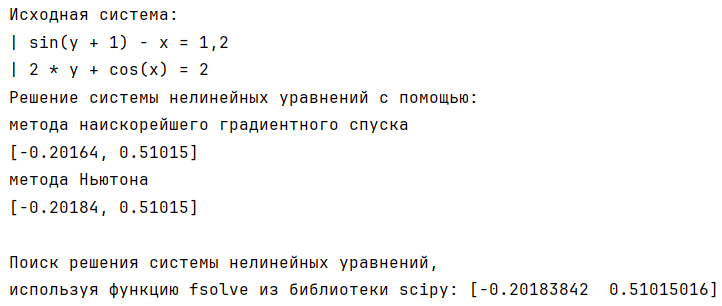
****

рис.2. Вывод результата задачи

**Проверка правильности решения.**

Правильность решений была проверена с помощью unit тестов.

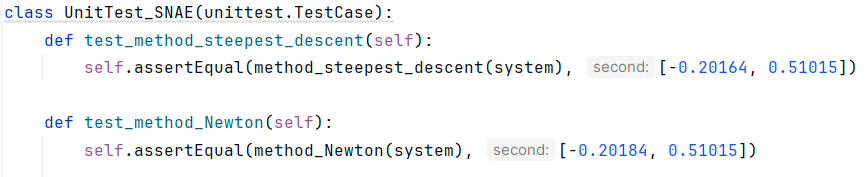


рис.3. Проверка корректности программы с помощью unit тестов

**Выводы.**

Решили заданную систему нелинейных уравнений с помощью 1) метода наискорейшего градиентного спуска и 2) метода Ньютона с точностью до 0.001. Проверили решение с помощью unit тестов.

**Приложение: код программы.**

| **import math**  **import numpy as np**  **from scipy.optimize import fsolve**  **def system(arg):**  ***"""Исходная система"""***  **x, y = arg**  **r = [0] \* 2**  **r[0] = math.sin(y + 1) - x - 1.2**  **r[1] = 2 \* y + math.cos(x) - 2**  **return r**  **def method\_steepest\_descent(sys, arg=None,**  **accuracy=1e-3, max\_iter=100):**  ***"""***  ***Решение системы нелинейных уравнений с помощью***  ***метода наискорейшего градиентного спуска***  ***"""***  **def target\_function(sys\_f, arg\_vect):**  ***""" метод для поиска целевой функции """***  **return sum([func \*\* 2 for func in sys\_f(arg\_vect)])**  **def derivative1(sys\_f, arg\_vect, h\_step, i):**  ***""" метод для поиска производной """***  **e = [0] \* len(arg\_vect)**  **e[i] = 1**  **dif\_f = target\_function(sys\_f, [arg\_vect[index] - e[index] \* h\_step for index in range(len(e))])**  **sum\_f = target\_function(sys\_f, [arg\_vect[index] + e[index] \* h\_step for index in range(len(e))])**  **derivative = (sum\_f - dif\_f) / (2 \* h\_step)**  **return derivative**  **def grad(sys\_f, arg\_vect, h\_step):**  ***""" метод для поиска градиента """***  **u\_grad = [0] \* 2**  **for i in range(len(arg\_vect)):**  **e = [0] \* 2**  **e[i] = 1**  **dtarg = derivative1(sys\_f, arg\_vect, h\_step, i)**  **u\_grad[i] += dtarg \* e[i]**  **return u\_grad**  **if arg is None:**  **arg = [0, 0]**  **iteration\_count = 0**  **while iteration\_count < max\_iter:**  **target = target\_function(sys, arg)**  **h = 0.001**  **while h < 1:**  **u = grad(sys, arg, h)**  **arg\_min = arg[0] - h \* u[0], arg[1] - h \* u[1]**  **h += 0.001**  **if target\_function(sys, arg\_min) < target\_function(sys, arg):**  **arg = arg\_min**  **target\_cur = target\_function(sys, arg)**  **if abs(target - target\_cur) < accuracy:**  **break**  **iteration\_count += 1**  **return [round(argi, 5) for argi in arg]**  **def method\_Newton(sys, arg=None,**  **accuracy=1e-3, max\_iter=100):**  ***"""***  ***Решение системы нелинейных уравнений с помощью***  ***метода Ньютона***  ***"""***  **def Jacobian\_matrix(sys\_f, arg\_vect,**  **accuracy\_jacobian=1e-3):**  ***""" метод для поиска матрицы Якоби """***  **n = len(arg\_vect)**  **A\_matrix = [[]] \* n**  **for i in range(n):**  **row = [0] \* n**  **for j in range(n):**  **x\_accuracy = arg\_vect.copy()**  **x\_accuracy[j] += accuracy\_jacobian**  **sum\_f = sys\_f(x\_accuracy)**  **dif\_f = sys\_f(arg\_vect)**  **derivative = (sum\_f[i] - dif\_f[i]) / accuracy\_jacobian**  **row[j] = derivative**  **A\_matrix[i] = row**  **return A\_matrix**  **if arg is None:**  **arg = method\_steepest\_descent(sys)**  **iteration\_count = 1**  **while iteration\_count < max\_iter:**  **L = np.array(sys(arg))**  **A = np.array(Jacobian\_matrix(sys, arg))**  **delta\_arg = np.linalg.solve(-A, L)**  **arg = delta\_arg + arg**  **if all(abs(d\_arg) < accuracy for d\_arg in sys(arg)):**  **break**  **iteration\_count += 1**  **return [round(argi, 5) for argi in arg]**  **if \_\_name\_\_ == "\_\_main\_\_":**  **print(f"Исходная система:\n"**  **f"| sin(y + 1) - x = 1,2\n"**  **f"| 2 \* y + cos(x) = 2")**  **print("Решение системы нелинейных уравнений с помощью:")**  **print("метода наискорейшего градиентного спуска")**  **res1 = method\_steepest\_descent(system)**  **print(res1)**  **print("метода Ньютона")**  **res2 = method\_Newton(system)**  **print(res2)**  **correct\_result = fsolve(system, [0, 0])**  **print(f"\nПоиск решения системы нелинейных уравнений, \n"**  **f"используя функцию fsolve "**  **f"из библиотеки scipy: {correct\_result}")** |
| --- |