|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ *Робототехники и комплексной автоматизации*

КАФЕДРА *Системы автоматизированного проектирования (РК-6)*

**ОТЧЕТ О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ**

по дисциплине: «Вычислительная математика»

|  |  |  |
| --- | --- | --- |
| Студент |  | Шашко Олег Владимирович |
| Группа |  | РК6-51Б |
| Тип задания |  | лабораторная работа |
| Тема лабораторной работы |  | Модель Лотки–Вольтерры |

Студент **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Шашко О.В.**

*подпись, дата фамилия, и.о.*

Преподаватель **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Соколов А.П.**

*подпись, дата фамилия, и.о.*

*Москва, 2022 г.*

Оглавление

[Задание на лабораторную работу 3](#_Toc32399212)

[Цель выполнения лабораторной работы 3](#_Toc32399213)

[Выполненные задачи 3](#_Toc32399214)

[1. @Название раздела в соответствии с задачей 1@ 4](#_Toc32399215)

[2. @Название раздела в соответствии с задачей 2@ 4](#_Toc32399216)

[Заключение 4](#_Toc32399217)

[Список использованных источников 4](#_Toc32399218)

# Задание на лабораторную работу

Задача 18 (модель Лотки–Вольтерры)

Дана модель Лотки–Вольтерры в виде системы ОДУ где

(25)

где 𝑥 – количество “жертв”, 𝑦 – количество “хищников”, 𝛼 = 3 – коэффициент рождаемости “жертв”, 𝛽 = 0.002 – коэффициент убыли “жертв”, 𝛿 = 0.0006 – коэффициент рождаемости “хищников”, 𝛾 = 0.5 – коэффициент убыли “хищников”.

Требуется (базовая часть):

1. Написать функцию rk4(x\_0, t\_n, f, h), возвращающая дискретную траекторию системы ОДУ с правой частью, заданную функцией f, начальным условием x\_0, шагом по времени h и конечным временем t\_n, полученную с помощью метода Рунге–Кутты 4-го порядка.

2. Найти траектории для заданной системы для ряда начальных условий 𝑥(0) =

200𝑖,𝑦(0) = 200𝑗, где 𝑖,𝑗 = 1,...,10. 𝑗

3. Вывести все полученные траектории на одном графике в виде фазового портрета. Объясните, какой вид имеют все полученные траектории. В качестве подтверждения выведите на экран совместно графики траекторий 𝑥(𝑡) и 𝑦(𝑡) для одного репрезентативного случая.

Требуется (продвинутая часть):  
1. Найти аналитически все стационарные позиции заданной системы ОДУ.  
2. Отметить на фазовом портрете, полученном в базовой части, найденные стационарные позиции. Объясните, что происходит с траекториями заданной системы

при приближении к каждой из стационарных позиций.  
3. Написать функцию newton(x\_0, f, J), которая, используя метод Ньютона, воз-

вращает корень векторной функции f с матрицей Якоби J и количество проведенных

итераций. Аргументы f и J являются функциями, принимающими на вход вектор x

и возвращающими соответственно вектор и матрицу. В качестве критерия остановки

следует использовать ограничение на относительное улучшение71,72: ||𝑥𝑘+1 −𝑥𝑘||∞ < −8

𝜖,где𝜖=10 .  
4. Написать функцию gradient\_descent(x\_0, f, J), которая, используя метод градиентного спуска, возвращает корень векторной функции f с матрицей Якоби J и количество проведенных итераций. Используйте тот же критерий остановки, что и в предыдущем пункте.

5. Используя каждую из функций newton() и gradient\_descent(), провести следующий анализ:

(a) Найти стационарные позиции как нули заданной векторной функции 𝑓(𝑥) для ряда начальных условий 𝑥(0) = 15𝑖,𝑦(0) = 15𝑗, где 𝑖,𝑗 = 0,1,...,200.

𝑖𝑗

(b) Длякаждойполученнойстационарнойпозициирассчитатьеёсупремум-норму, что в результате даст матрицу супремум-норм размерности 201 × 201.

(c) Вывести на экран линии уровня73 с заполнением для полученной матрицы (0) (0)

относительно значений 𝑥𝑖 , 𝑦𝑗 .  
(d) Описать наблюдения, исходя из подобной визуализации результатов.  
(e) Найти математическое ожидание и среднеквадратическое отклонение количества итераций.  
(f) Выбрать некоторую репрезентативную начальную точку из 𝑥(0),𝑦(0) и про-

демонстрировать степень сходимости метода с помощью соответствующего loglog-графика.

6) Проанализировав полученные результаты, сравнить свойства сходимости метода Ньютона и метода градиентного спуска.

# Цель выполнения лабораторной работы

**Цель выполнения лабораторной работы** – освоить решение ОДУ методом Рунге-Кутта, методом Ньютона и градиентного спуска.

# Выполненные задачи

* **Задача 1 –** написать функцию rk4(x\_0, t\_n, f, h)
* **Задача 2 –** найти траектории для заданной системы для ряда начальных условий
* **Задача 3 –** вывести все полученные траектории на одном графике в виде фазового портрета
* **Задача 4 –** отметить на фазовом портрете стационарные точки
* **Задача 5 –** написать функцию newton(x\_0, f, J)
* **Задача 6 –** написать функцию gradient\_descent(x\_0, f, J)
* **Задача 7 –** использую функции из предыдущих двух пунктов, провести анализ
* **Задача 8 –** проанализировав полученные результаты, сравнить свойства сходимости методов Ньютона и градиентного спуска
* **Задача 9 –** найти решения модифицированной СЛАУ
* **Задача 10 -** построить log-log график зависимости относительной погрешности

**1. Базовая часть**

1. Задача 1 – написать функцию rk4(x\_0, t\_n, f, h)

По условию задачи, функция должна возвращать дискретную траекторию(обозначим ее y), тогда:

рассчитывается следующим образом:

*(1)*

Таким образом, каждое новое значение w будет находиться при помощи формулы (1).

Переходим к реализации:

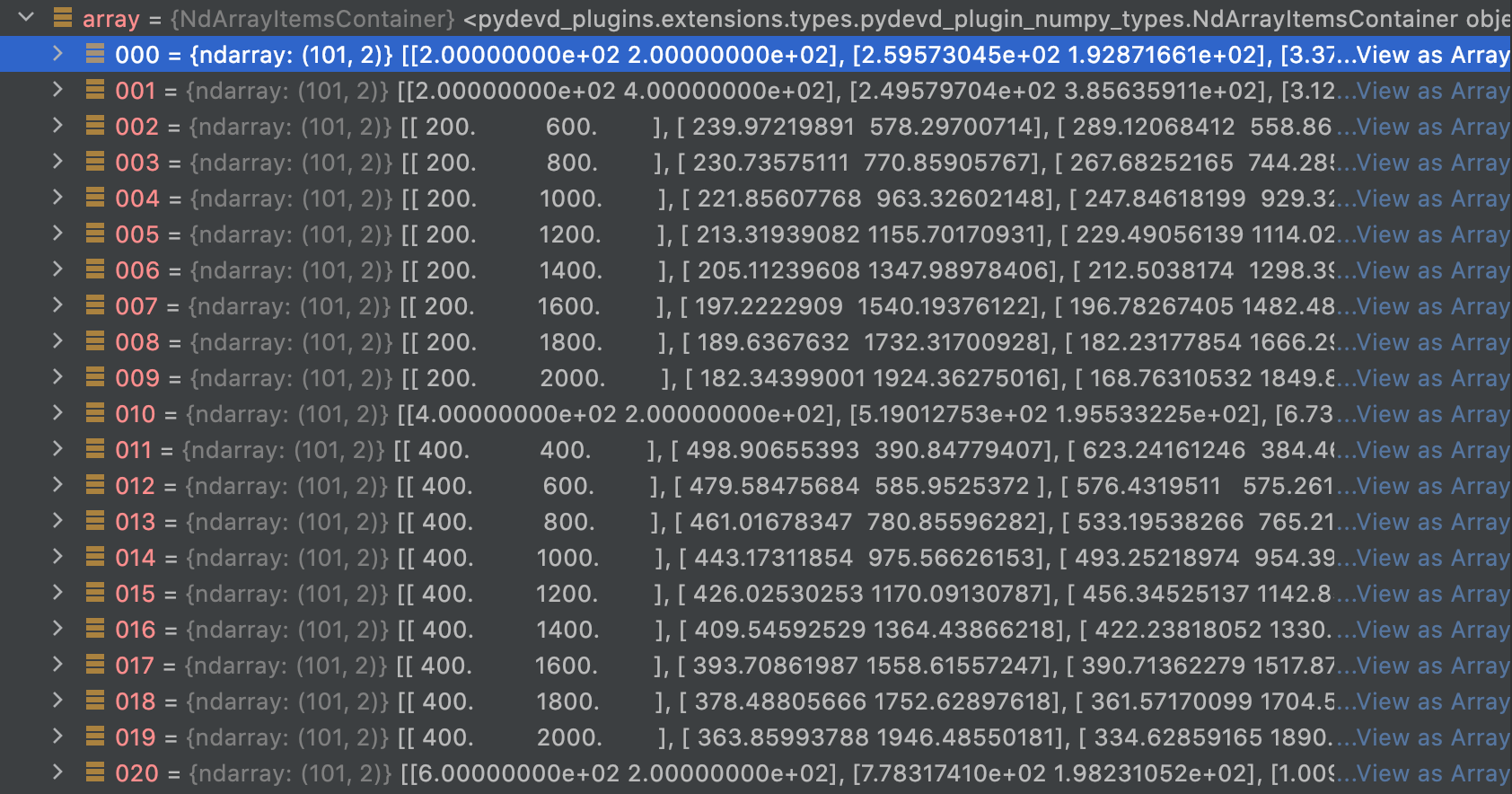
def rk4(x\_0, t\_n, f, h):  
 w = [] # np.zeros((t\_n+1,2))  
 w.append(x\_0)  
 t = [i \* h for i in range(t\_n + 1)]  
 for i in range(1, t\_n + 1):  
 wi = w[i - 1]  
 k1 = h \* f(w[i - 1])  
 k2 = h \* f(w[i - 1] + k1 / 2)  
 k3 = h \* f(w[i - 1] + k2 / 2)  
 k4 = h \* f(w[i - 1] + k3)  
 w.append(w[i - 1] + (k1 + 2 \* k2 + 2 \* k3 + k4) / 6)  
 return w

*Листинг 1 – реализация функции rk4*

1. Задача 2 – найти траектории для заданной системы для ряда начальных условий

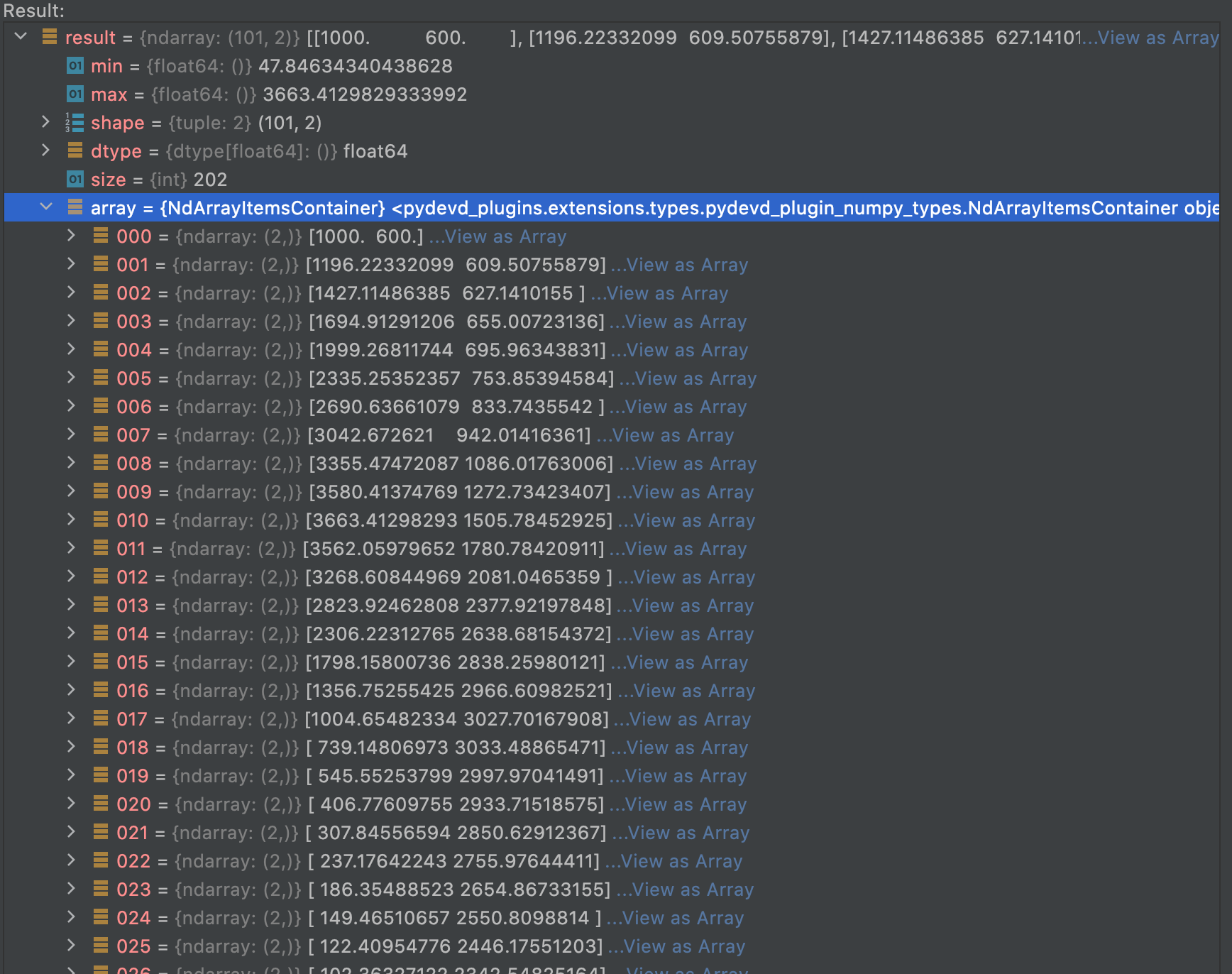
Рассчитаем в цикле вектора использую функцию, представленную на листинге.

Таким образом, у нас получится 100 различных траекторий. Массив, хранящий эти значения представлен на рисунке 1:



*Рисунок 1 – массив значений рассчитанных траекторий*

Рассмотрим одну из траекторий(например, траекторию с индексом 42) более подробно(рисунок 2):



*Рисунок 2 – одна из найденных траекторий*

3. Задача 3 – решить СЛАУ с помощью функции solve и затем сравнить полученный результат с точным решением

Из условия задачи, точное решение этой СЛАУ –

В результате работы программы получен результат

Сравнивая полученное решение с эталонным, можно сделать вывод, что метод LU – разложения является достаточно точным.

**2. Продвинутая часть**

2. Задача 4 – доказать необходимость перестановок строк

Для применения прямого хода метода Гаусса, который лежит в основе LU – разложения, необходимым условием является неравенство нулю угловых миноров. Для примера, возьмем СЛАУ (19) и посчитаем ее угловые миноры:

*(19)*

Мы получили угловой минор, равный нулю. Таким образом, данную СЛАУ нельзя решить без перестановки строк.

5. Задача 5 – модифицировать функцию lu(A, permute)

В задаче 1 базовой части мы реализовали функцию lu, которая возвращала нам матрицы L и U, вычисленные без выбора главного элемента. Таким образом, для случая, когда permute=False, нам полностью подходит реализация, представленная на листинге 1, за исключением return – тут мы будем возвращать помимо матриц L и U единичную матрицу P, которую можно получить с помощью библиотечной функции np.identity().

Для случая, когда permute=True, нам необходимо выбрать условие, при котором будет производится выбор главного элемента. Рационально будет использовать условие, при котором абсолютное значение диагонального элемента не превосходит 10e-10.

Поскольку , то PA

Переходим к реализации:

def lu(a1, permute):  
 a = a1  
 n = len(a)  
  
 p = np.identity(n)  
  
 if not permute:  
  
 n = len(a)  
  
 u = a  
 l = np.identity(n)  
 for j in range(0, n):  
 m\_j = np.identity(n)  
 for i in range(j + 1, n):  
 m\_j.itemset((i, j), -a[i, j] / a[j, j])  
 l.itemset((i, j), a[i, j] / a[j, j])  
 a = np.matmul(m\_j, a)  
 u = np.matmul(m\_j, u)  
 return [l, u, p]  
 else:  
 u = a  
 l = np.identity(n)  
 for j in range(0, n):  
 m\_j = np.identity(n)  
 for i in range(j + 1, n):  
 if abs(a[j, j]) > 10e-10:  
 m\_j.itemset((i, j), -a[i, j] / a[j, j])  
 l.itemset((i, j), a[i, j] / a[j, j])  
  
 else:  
 print('check')  
 max\_ind = index\_of\_max\_element(a[:][j])  
 print(f"max\_ind = ", max\_ind)  
 a = swap(a, j, max\_ind)  
 p = swap(p, j, max\_ind)  
 m\_j.itemset((i, j), -a[i, j] / a[j, j])  
 l.itemset((i, j), a[i, j] / a[j, j])  
 a = np.matmul(m\_j, a)  
 u = np.matmul(m\_j, u)  
 return [np.matmul(np.matmul(p, l), np.linalg.inv(p)), np.matmul(p, u), p]

*Листинг 3 – реализация функции lu с возможностью частичного выбора главного элемента*

6. Задача 6 – модифицировать функцию solve(L, U, P, b)

Для того, чтобы функция корректно вычисляла значения неизвестных, необходимо умножить вектор b, передаваемый в аргументах функции, на матрицу перестановок P. Программная реализация представлена на листинге 4:

def solve(l, u, vec1, p):  
 vec = np.matmul(vec1, p)  
 n = len(vec)  
 y = []  
 for k in range(0, n):  
 sum = 0  
 for i in range(0, k):  
 sum += l[k, i] \* y[i]  
 print(sum)  
 y.append(vec[k] - sum)  
 x = [0.0] \* n  
 for k in range(0, n):  
 sum = 0  
 for i in range(k + 1, n):  
 sum += u[k, i] \* x[i]  
 print(sum)  
 x[k] = y[k] / u[k, k] - sum / u[k, k]  
 return x

*Листинг 4 – реализация функции solve для решения СЛАУ с выбором главного элемента*

7. Задача 7 – найти решение СЛАУ(19) с помощью функции, реализованной в предыдущей задаче.

В результате работы функции были получены значения

Сравним с точным решением этой СЛАУ (точное решение посчитаем с помощью сервиса Wolfram Alpha) - . Таким образом, методом LU – разложения для данной матрицы мы получили точное решение системы.

8. Задача 8 – доказать, что при добавлении к первому элементу матрицы и вектора малого числа, решение этой СЛАУ не поменяется

Добавим к элементу матрицы и вектору число Затем, используя функцию solve, реализованную раннее, решим новое СЛАУ:

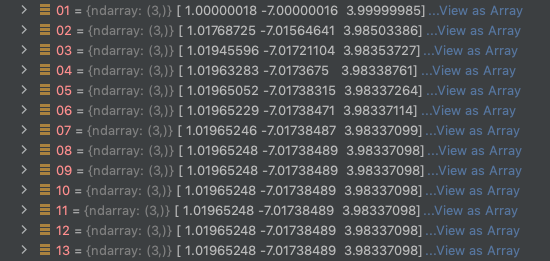
Сохраним решение исходной и модифицированной СЛАУ в переменные и сравним полученные результаты (рис 1):



*Рисунок 1 – результаты решения исходной и модифицированной СЛАУ.*

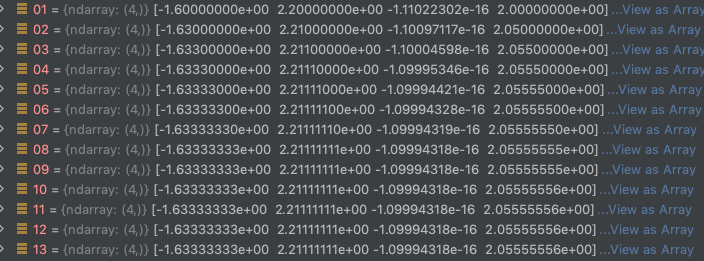
8. Задача 8 – найти решения СЛАУ для

1) Для случая с выбором главного элемента (решаем СЛАУ(19)) полученные решения представлены на рисунке 2:



*Рисунок 2 – результаты решения СЛАУ(19) с выбором главного элемента.*

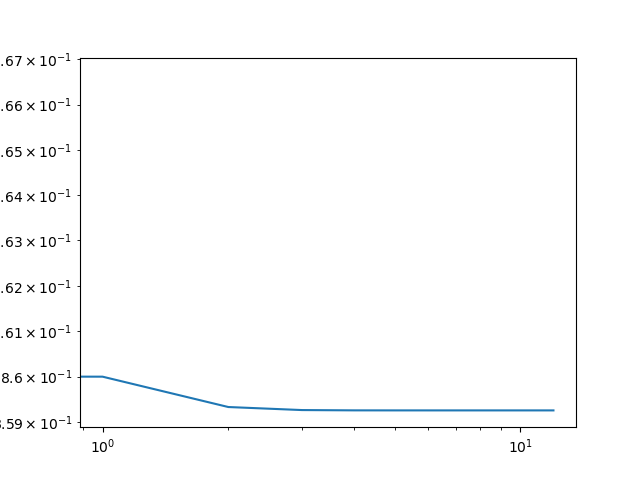
Теперь решим СЛАУ(18) без выбора главного элемента. Полученные значения представлены на рисунке 3:



*Рисунок 3 – результаты решения СЛАУ(18) без выбора главного элемента.*

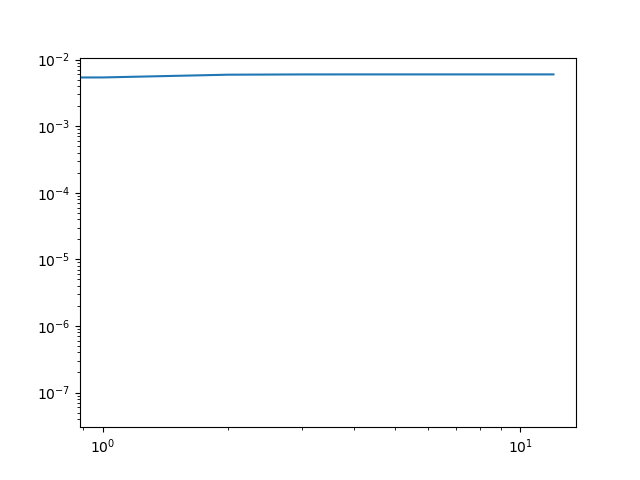
9. Задача 9 – для обоих случаев, рассмотренных в предыдущей задаче, построить log-log графики зависимости погрешности от p.

Для случая без выбора главного элемента график погрешности представлен на рисунке 4:



*Рисунок 4 – график зависимости погрешности от p для случая без выбора главного элемента*

Для случая с выбором главного элемента, график представлен на рисунке 5:



*Рисунок 5 – график зависимости погрешности от p для случая с выбором главного элемента*

# Заключение

Мы изучили метод LU-разложения как с частичным выбором главного элемента, так и без такого, для решения СЛАУ, а также научились реализовать его на языке программирования python.

# Список использованных источников

1. **Першин А.Ю.** *Лекции по курсу “Вычислительная математика”.* //

Москва, 2018-2021. С. 140.

2. *Официальная документация numpy* [Электронный ресурс] // Режим доступа: https://numpy.org/, свободный. Яз. англ.

*Шашко О.В. Отчет о выполнении лабораторной работы по курсу “Вычислительная математика”. [Электронный ресурс] – Москва: 2022 – 13 с. URL: https://sa2systems.ru:88 (система контроля версий кафедры РК6)*