|  |  |
| --- | --- |
| 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 , где

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 улучшение:

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:

1. 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.
2. Для каждой полученной стационарной позиции рассчитать её супремум-норму, что в результате даст матрицу супремум-норм размерности 201 × 201.
3. Вывести на экран линии уровня с заполнением для полученной матрицы относительно значений .  
   (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 продемонстрировать 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 методом градиентного спуска, провести соответствующий анализ и сравнить свойства сходимости последних двух методов

# 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 –** найти аналитически все стационарные позиции заданной системы ОДУ
* **Задача 5 –** отметить на фазовом портрете найденные стационарные позиции
* **Задача 6 –** написать функцию newton(x\_0, f, J)
* **Задача 7 –** написать функцию gradient\_descent(x\_0, f, J)
* **Задача 8 –** использую функции из предыдущих двух пунктов, провести анализ
* **Задача 9 –** проанализировав полученные результаты, сравнить свойства сходимости методов Ньютона и градиентного спуска
* **Задача 10 –** найти решения модифицированной СЛАУ
* **Задача 11 -** 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 (Листинг 1):

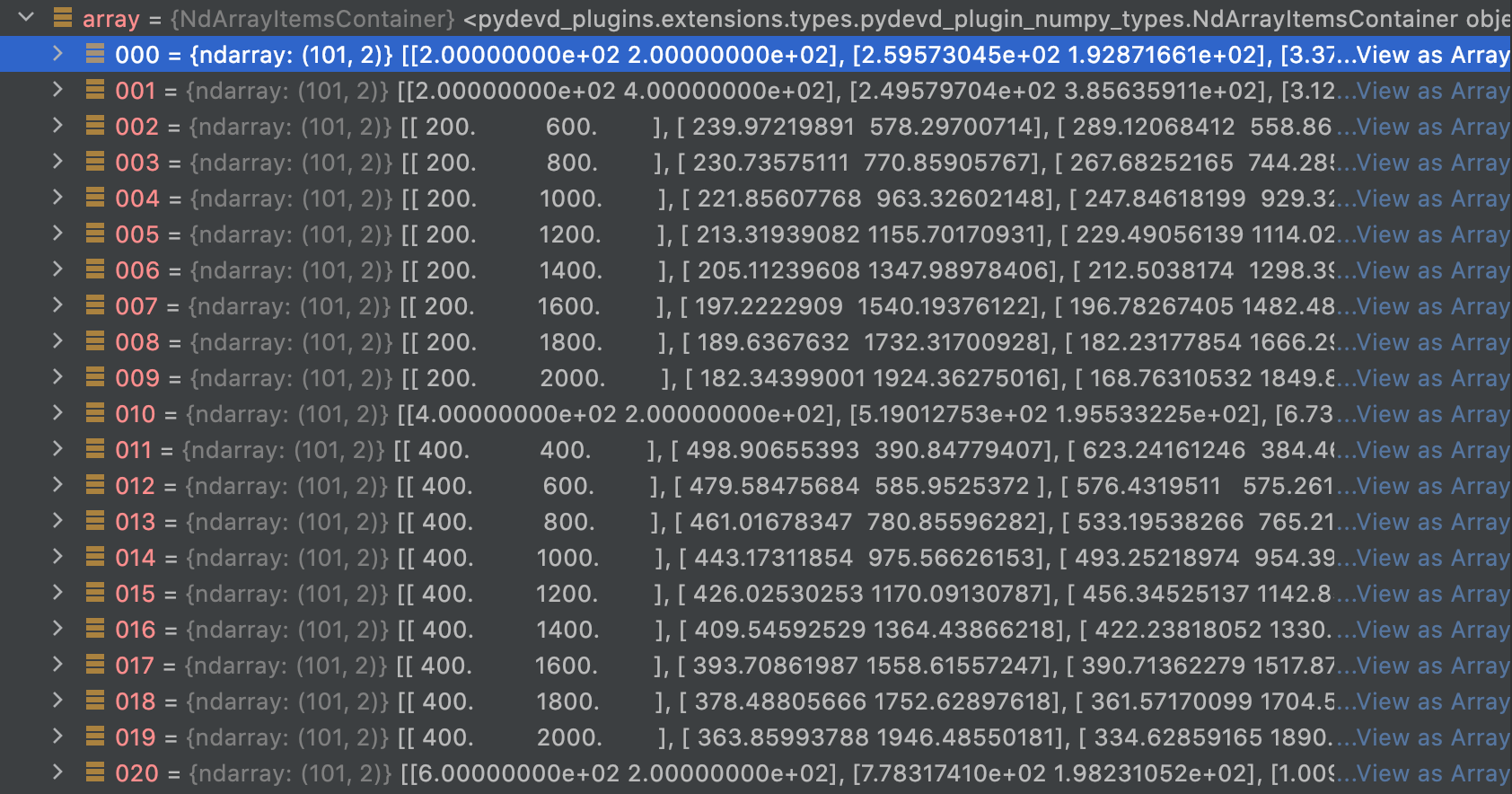
def rk4(x\_0, t\_n, f, h):  
 w = [] # np.zeros((t\_n+1,2))  
 w.append(x\_0)  
 t = [i \* h for i in range(t\_n + 1)]  
 for i in range(1, t\_n + 1):  
 wi = w[i - 1]  
 k1 = h \* f(w[i - 1])  
 k2 = h \* f(w[i - 1] + k1 / 2)  
 k3 = h \* f(w[i - 1] + k2 / 2)  
 k4 = h \* f(w[i - 1] + k3)  
 w.append(w[i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

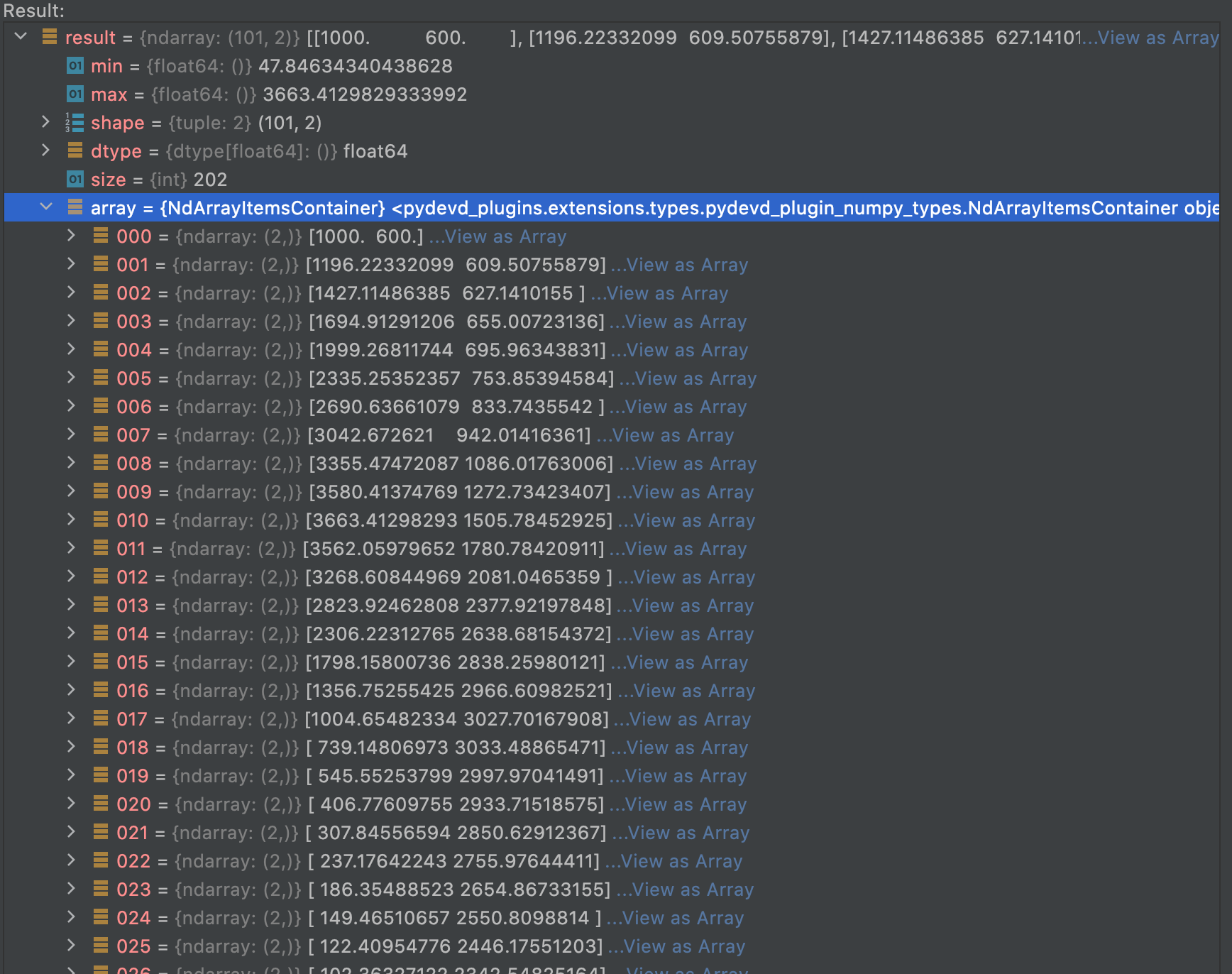
Рассчитаем в цикле вектора использую функцию, представленную на листинге.

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



*Рисунок 1 – массив значений рассчитанных траекторий*

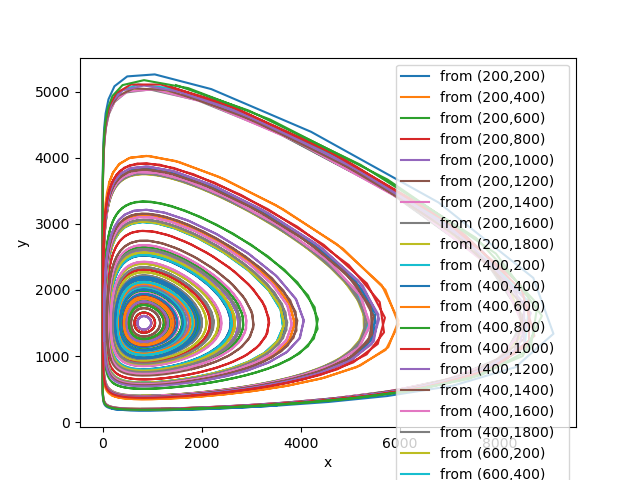
Рассмотрим одну из траекторий (например, траекторию с индексом 42) более подробно (рисунок 2):



*Рисунок 2 – одна из найденных траекторий*

3. Задача 3 – вывести все полученные траектории на одном графике в виде фазового портрета

Построим в цикле графики полученных траекторий. Всего у нас будет 81 график (представлены на рисунке 3)



*Рисунок 3 – графики полученных траекторий*

**2. Продвинутая часть**

2. Задача 4 – найти аналитически все стационарные позиции заданной системы ОДУ

Для нахождения стационарных точек системы ОДУ необходимо, чтобы функция являлась константой. Таким образом:

*(1.1)*

Исходя из (1.1), приравниваем правую часть ОДУ к нулю и вычисляем стационарные точки (1.2):

*(1.2)*

Мы нашли стационарные точки для заданной системы ОДУ

3. Задача 5 – отметить на фазовом портрете найденные стационарные позиции

Для определения характера точек нам необходимо линеаризовать нашу систему.

1) Точка (0, 0):

*(2.1)*

*(2.2)*

*(2.3)*

2) Точка :

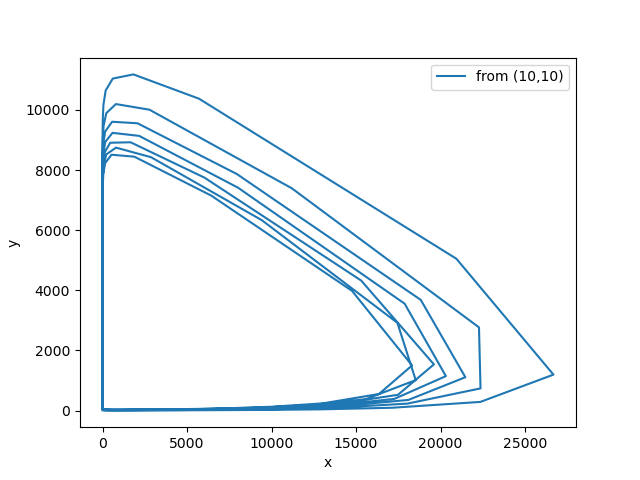
Находим собственные значения матриц и :

Для случая , расположение траекторий в окрестности точки (0, 0) имеют форму седла.

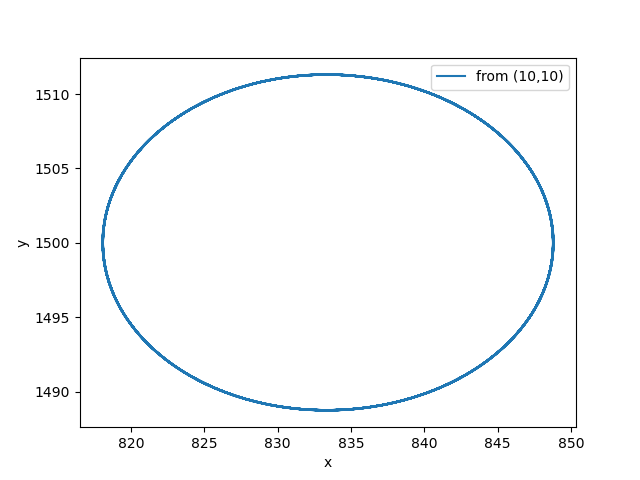
Для случая , расположение траекторий в окрестности точки имеют форму центра.

Вернемся к последнему пункту базовой части и продемонстрируем на графике:

Для приближения к точке (0, 0) используем начальные условия (10, 10) График представлен на рисунке 4:

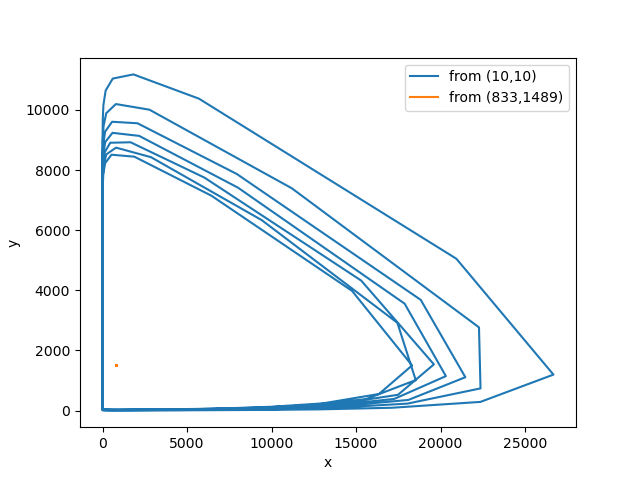
****

*Рисунок 4 – графики в окрестности точки (0,0)*

Аналогично для второй точки, начальные условия возьмем (830, 1489): (Рисунок 5)

*Рисунок 5 – график в окрестности точки (834, 1500)*

Теперь построим графики 4-5 на одном графике (рисунок 6)



*Рисунок 6 – объединённый график*

4. Задача 6 – написать функцию newton(x\_0, f, J)

Для вычисления матрицы Якоби воспользуемся подсказкой 72 из методических указаний к данной лабораторной работе.

Корень векторной функции методом Ньютона находится по формуле:

*(3.1)*

Вектор – результат решения СЛАУ . В нашей реализации для нахождения этого вектора воспользуемся библиотечной функцией np.linalg.solve()

*(3.2)*

*(3.3)*

*(3.4)*

Критерий для остановки в нашем случае -

Переходим к реализации метода Ньютона (листинг 2):

def newton(x\_0, f, J):  
 y1 = x\_0  
 slau\_sol = np.linalg.solve(J(y1), f(y1))  
 y2 = y1 - slau\_sol  
 n = 1  
 while np.linalg.norm(y2 - y1, ord=np.inf) >= 10e-8:  
 y1 = y2  
 y2 = y1 - np.linalg.solve(J(y1), f(y1))  
 n += 1  
 return y2, n

*Листинг 2 – реализация функции newton*

5. Задача 7 – написать функцию gradient\_descent(x\_0, f, J)

Для начала, нам необходимо найти минимум скалярного произведения векторов . (4.1)

*(4.1)*

для данного метода может быть найден по формуле (4.2)

*(4.2)*

для предыдущей формулы вычисляется по формуле (4.3):

*(4.3)*

Коэффициенты находятся по формулам (4.4 - 4.7):

*(4.4)*

*(4.5)*

*(4.6),*

где

Для формулы (4.2) z вычисляется следующим образом (4.7):

*(4.7)*

Нам необходимо найти минимум функции где , причем

Задача нахождения минимума функции сводится к задаче нахождения производной этой функции и приравнивании ее к нулю.

, причем

Теперь можно переходить к реализации функции, но для упрощения задачи и избежание дублирования кода, реализуем вспомогательные функции: solve\_t3 для нахождения коэффициента и h для нахождения скалярного произведения векторов. Реализация этих функций представлена на листингах 3-4:

def h(y, z, t, f):  
 f\_fix = f(y - t / np.linalg.norm(z) \* z)  
 g = np.dot(f\_fix, f\_fix)  
 return g

*Листинг 3 – реализация функции h*

def solve\_t3(h, y, z, f):  
 t = 1  
 while h(y, z, t, f) >= h(y, z, 0, f):  
 t /= 2  
 return t

*Листинг 4 – реализация функции solve\_t3*

Теперь можем переходить непосредственно к реализации функции градиентного спуска (листинг 5):

def gradient\_descent(x\_0, f, J):  
 y1 = x\_0  
 z = np.matmul(np.matrix.transpose(J(y1)), f(y1))  
 t3 = solve\_t3(h, y1, z, f)  
 t2 = t3 / 2  
 a = (h(y1, z, 0, f) / (t2 \* t3))  
 b = (h(y1, z, t2, f) / (t2 \* (t2 - t3)))  
 c = (h(y1, z, t3, f) / (t3 \* (t3 - t2)))  
 t = (a \* (t2 + t3) + b \* t3 + c \* t2) / (2 \* (a + b + c))  
 y2 = y1 - 2 \* t \* z / np.linalg.norm(z)  
 n = 1  
 while np.linalg.norm(y2 - y1, ord=np.inf) >= 10e-8:  
 y1 = y2  
 z = np.matmul(np.matrix.transpose(J(y1)), f(y1))  
 t3 = solve\_t3(h, y1, z, f)  
 t2 = t3 / 2  
 a = (h(y1, z, 0, f) / (t2 \* t3))  
 b = (h(y1, z, t2, f) / (t2 \* (t2 - t3)))  
 c = (h(y1, z, t3, f) / (t3 \* (t3 - t2)))  
 t = (a \* (t2 + t3) + b \* t3 + c \* t2) / (2 \* (a + b + c))  
 y2 = y1 - 2 \* t \* z / np.linalg.norm(z)  
 n += 1  
 return y2, n

*Листинг 5 – реализация функции gradient\_descent*

# Заключение

Мы освоили решение ОДУ методом Рунге-Кутта, методом Ньютона и методом градиентного спуска, провели соответствующий анализ и сравнили свойства сходимости последних двух методов, а также научились реализовать их 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)*