**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**Федеральное государственное автономное образовательное учреждение**

**высшего образования**

**«Национальный исследовательский**

**Нижегородский государственный университет им. Н.И. Лобачевского»**

Радиофизический факультет

Кафедра теории колебаний и автоматического регулирования

Направление «Фундаментальная информатика

и информационные технологии»

ОТЧЕТ ПО УЧЕБНОЙ ПРАКТИКЕ

Научно-исследовательская работа (получение первичных навыков научно-исследовательской работы)

**АППРОКСИМАЦИЯ НЕПРЕРЫВНОЙ МОДЕЛИ МУЛЬТИВИБРАТОРА МОДЕЛЬЮ С КОНЕЧНЫМ МНОЖЕСТВОМ СОСТОЯНИЙ**

Научный руководитель:  
доктор физико-математических наук

Радиофизический факультет

Кафедра теории колебаний и

автоматического регулирования Канаков Олег Игоревич

Студент 3-го курса Семиков Алексей Александрович

Нижний Новгород, 2023

**Введение**

Динамическая система представляет собой такую [математическую модель](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C) некоего объекта, процесса или явления, в которой пренебрегают «флуктуациями и всеми другими статистическими явлениями».

Динамическая система также может быть представлена как система, обладающая состоянием. При таком подходе, динамическая система описывает (в целом) динамику некоторого процесса, а именно: процесс перехода системы из одного состояния в другое. [Фазовое пространство](https://ru.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B7%D0%BE%D0%B2%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D1%81%D1%82%D1%80%D0%B0%D0%BD%D1%81%D1%82%D0%B2%D0%BE) системы — совокупность всех допустимых состояний динамической системы. Таким образом, динамическая система характеризуется своим начальным состоянием и законом, по которому система переходит из начального состояния в другое.

Различают системы с дискретным временем и системы с непрерывным временем.

В данной работе решается задача аппроксимации непрерывной динамической системы моделью с конечным множеством состояний. Аппроксимация позволяет исследовать числовые характеристики и качественные свойства объекта, сводя задачу к изучению более простых или более удобных объектов. Во-первых, это дает возможность более простого моделирования, а во-вторых, это инструмент исследования динамики исходной модели.

**Ход выполнения**

Для моделирования была взята система уравнений:

(1)

Для решения системы уравнений (1) был использован метод Эйлера. Выбираем шаг и итерационным методом решаем систему:

(2)

(3)

где функции и имеют вид:

(4)

Программная реализация данного метода для нашей системы, заключается в отыскании решений в прямоугольной область [-2.6, 2.6], [-4.3, 4.3] с шагом .

*Листинг 1.*

# Начальные условия:

segment = [0,10]

h = 0.001

n = int( (segment[1] - segment[0]) / h )

start\_point = calc\_start\_point(1000, [[-2.6, 2.6], [-4.3, 4.3,]])

mu = 0.1

**try**:

grid = [[[0, 0, 0, 0] **for** col **in** range(matrix\_size)] **for** row **in** range(matrix\_size)] # заполнение сетки нулями

# �-аполнение массивов X, Y

**for** j **in** range(len(start\_point)):

point = [[], []] # массив содержащий [X, Y]

**for** i **in** range(n):

**if**(i == 0):

point[0].append(start\_point[j][0] + func\_1(start\_point[j][0], start\_point[j][1]) \* h / mu)

point[1].append(start\_point[j][1] + func\_1(start\_point[j][0], start\_point[j][1]) \* h / mu)

point[0].append(point[0][i] + func\_1(point[0][i], point[1][i]) \* h / mu)

point[1].append(point[1][i] + func\_2(point[0][i], point[1][i]) \* h)

graficSystem(point[0], point[1])

Начальная точка при прохождение цикла *j* изменяется с каждой итерацией. Её расчет происходит в функции *calc\_start\_point(quantity, segment)* случайным образом при помощи библиотеки *random* в ограниченном отрезками пространстве: [-2.6, 2.6], [-4.3, 4.3].

*Листинг 2.*

**def** calc\_start\_point(quantity, segment):

start\_point = [[None, None]] \* quantity

**for** i **in** range(quantity):

start\_point[i] = [r.uniform(segment[0][0], segment[0][1]), r.uniform(segment[1][0], segment[1][1])]

**return** start\_point

Отобразим решение системы на графике для 5000 разных начальных точек:

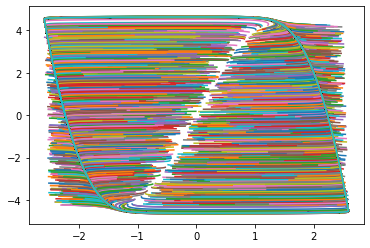


Рис. 3

Заменяем фазовую плоскость сеткой с конечным количеством состояний. Переходы системы между состояниями будем описывать матрицей. Для создания матрицы будем использовать функцию *create\_grid(matrix\_size, func\_1, func\_2, cell\_size=0)*. Функция принимает в качестве аргументов: размер матрицы, и соответственно, размер клетки (необязательно). Для удобства работы, поместим метод Эйлера в саму функцию.

Если размер клетки не был указан, то он рассчитывается автоматически с помощью начальных условий:

*Листинг 3.*

**if**(cell\_size == 0):

cell\_size = calculation\_cell\_size(point[0], point[1], matrix\_size) # размер клетки

**print**('Размер клетки:', cell\_size)

Функция *calculation\_cell\_size(X, Y, matrix\_size)* производит расчет размера матрицы исходя из размера массивов X, Y, полученных из метода Эйлера, и размера матрицы:

*Листинг 4.*

**def** calculation\_cell\_size(X, Y, matrix\_size):

# Расчет размера клетки по размеру матрицы

cell\_size = [((np.max(X) - np.min(X)) / (matrix\_size - 5)), ((np.max(Y) - np.min(Y)) / (matrix\_size - 5))]

**return** np.max(cell\_size)

Так как границы в сетке состояний не совпадают с целочисленными значениями координат, то необходимо находить координаты для графика относительно центра. Найду координаты относительно левого верхнего угла матрицы. Для этого необходимо умножить размер клетки на столбец (*n*) или строку (*m*):

Из этих формул можно найти номер столбца и строки, в которую попала наша координата. После пересчитываем координаты относительно центра матрицы:

*Листинг 5.*

x\_temp = int( (matrix\_size - 1) / 2 ) + int( point[0][0] / cell\_size ) # Координата X первой точки

y\_temp = int( (matrix\_size - 1) / 2 ) + int( point[1][0] / cell\_size ) # Координата У первой точки

# заполнение матрицы по траектории движения

**for** i **in** range(2, n):

**if**(error\_len(x\_temp, y\_temp, grid)):

x\_coor = int( (matrix\_size + 1) / 2 ) + int( point[0][i] / cell\_size ) # Координата Х следующей точки

y\_coor = int( (matrix\_size + 1) / 2 ) - int( point[1][i] / cell\_size ) # Координата У следующей точки

**if**(error\_len(x\_coor, y\_coor, grid)):

**if**([x\_temp, y\_temp] != [x\_coor, y\_coor]):

save\_coord\_point([x\_temp, y\_temp]) # Сохранение точек в массив

**else**:

**continue**

x\_temp = x\_coor # Переприсваиваем Х

y\_temp = y\_coor # Переприсваиваем У

**else**:

**continue**

Далее определяю траекторию движения и расставляю цифры в матрицы по траектории движения: [1, 2, 3, 4]. Определяем из какой точки мы пришли по номеру в массиве: 0 – вверх, 1 – вправо, 2 – вниз, 3 – влево. Цифра стоящая в массиве – следующая точка: 1 – вверх, 2 – вправо, 3 – вниз, 4 – влево. Для этого используется функция *comparison\_point(coord\_point\_array, grid)*.

*Листинг 6.*

**def** comparison\_point(arr, grid):

# Принимает: массив с точками, матрицу

**for** i **in** range(len(arr)-1):

**if**(grid[arr[i][1]][arr[i][0]] == 0):

grid[arr[i][1]][arr[i][0]] = [0, 0, 0, 0]

# заполнение массива точками

**for** i **in** range(1, len(arr)-1):

**if**(arr[i-1][1] < arr[i][1]):

k = 0

**if**(arr[i-1][1] > arr[i][1]):

k = 2

**if**(arr[i-1][0] < arr[i][0]):

k = 3

**if**(arr[i-1][0] > arr[i][0]):

k = 1

**if**(arr[i+1][1] < arr[i][1]):

grid[arr[i][1]][arr[i][0]][k] = 1

**if**(arr[i+1][1] > arr[i][1]):

grid[arr[i][1]][arr[i][0]][k] = 3

**if**(arr[i+1][0] < arr[i][0]):

grid[arr[i][1]][arr[i][0]][k] = 4

**if**(arr[i+1][0] > arr[i][0]):

grid[arr[i][1]][arr[i][0]][k] = 2

Распечатанная матрица приведена в текстовом файле *5000.txt*

Матрица описывает правила перехода между состояниями в системе с конечным множеством состояний. Для того чтобы обеспечить возможность сохранения правил, описывающих ансамбль траекторий, реализовано запоминание в каждой клетки матрицы предыдущего состояния системы.

**Заключение**

Мы представили нашу динамическую систему в виде автомата с конечным числом состояний. Исследовав качественно полученный автомат, мы можем качественно оценить динамику системы. Использование компьютера дает приближенное решение дифференциальных уравнений на конечном отрезке времени, что позволяет качественно понять поведение фазовых траекторий в целом.

Отследив переход из одного состояния в другое с учетом предыдущего, мы добились однозначности эволюционных правил этого автомата в тех случаях, когда изменение невозможно определить только текущей клеткой. И это позволило создать правила динамики конечного автомата, который воспроизводит динамику исходной системы.

**Список литературы**

1. *Пухов А. А.* Лекции по колебаниям и волнам: учеб. пособие. В двух частях. Ч. 1. Колебания / А. А. Пухов. – Москва: МФТИ, 2019 – 208с.