­­­МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ

РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное

образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

ЭКОНОМИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информатики и прикладной математики

Кафедра прикладной математики и экономико-математических методов

**КУРСОВАЯ РАБОТА**

по дисциплине:

**«Численные методы»**

Тема: «Метод Галлея для решения нелинейных уравнений и систем нелинейных уравнений»

Направление: 01.03.02 Прикладная математика и информатика

Студент: Матвеева Валерия Владимировна

Группа: ПМ-2001 Подпись\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Проверил: Хазанов Владимир Борисович

Должность: д. ф-м. н., профессор

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Дата: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Санкт-Петербург

2022 г.

**ОГЛАВЛЕНИЕ**

[**ВВЕДЕНИЕ** 4](#_Toc101954078)

[**1.** **МЕТОД ЖОРДАНА ДЛЯ РЕШЕНИЯ СИСТЕМЫ ЛИНЕЙНЫХ АЛГЕБРАИЧЕСКИХ УРАВНЕНИЙ (СЛАУ) СХЕМОЙ ЕДИНСТВЕННОГО ДЕЛЕНИЯ** 5](#_Toc101954079)

[**1.1.** **Описание метода с необходимыми формулами** 5](#_Toc101954080)

[**1.2.** **Программная реализация** 6](#_Toc101954081)

[**1.3.** **Анализ полученных результатов** 7](#_Toc101954082)

[**1.4.** **Оценка точности полученных результатов** 9](#_Toc101954083)

[**2.** **РЕШЕНИЕ СЛАУ ИТЕРАЦИОННЫМ МЕТОДОМ МИНИМАЛЬНЫХ НЕВЯЗОК С ИСПОЛЬЗОВАНИЕМ 2 ТРАНСФОРМАЦИИ ГАУССА** 11](#_Toc101954084)

[**2.1.** **Описание метода с необходимыми формулами** 11](#_Toc101954085)

[**2.2.** **Программная реализация** 12](#_Toc101954086)

[**2.3.** **Анализ и оценка точности полученных результатов** 13](#_Toc101954087)

[**3.** **РЕШЕНИЕ СИСТЕМЫ НЕЛИНЕЙНЫХ АЛГЕБРАИЧЕСКИХ УРАВНЕНИЙ (СЛНУ) С ПОМОЩЬЮ МЕТОДА ГАЛЛЕЯ (ХЭЛЛИ)** 16](#_Toc101954088)

[**3.1.** **Описание метода с необходимыми формулами** 16](#_Toc101954089)

[**3.2.** **Программная реализация** 19](#_Toc101954090)

[**3.3.** **Анализ полученных результатов** 21](#_Toc101954091)

[**3.4.** **Оценка точности полученных результатов** 22](#_Toc101954092)

[**4.** **РЕШЕНИЕ ПРОБЛЕМЫ СОБСТВЕННЫХ ЗНАЧЕНИЙ ДЛЯ МАТРИЦЫ МЕТОДОМ ОБРАТНЫХ ИТЕРАЦИЙ СО СДВИГОМ** 25](#_Toc101954093)

[**4.1.** **Описание метода с необходимыми формулами** 25](#_Toc101954094)

[**4.2.** **Программная реализация** 26](#_Toc101954095)

[**4.3.** **Анализ полученных результатов** 28](#_Toc101954096)

[**5.** **АППРОКСИМАЦИЯ ФУНКЦИИ ИНТЕРПОЛЯЦИОННОЙ ФОРМУЛОЙ БЕССЕЛЯ ДЛЯ РАВНООТСТОЯЩИХ УЗЛОВ** 31](#_Toc101954097)

[**5.1.** **Описание метода с необходимыми формулами** 31](#_Toc101954098)

[**5.2.** **Программная реализация** 33](#_Toc101954099)

[**5.3.** **Анализ и оценка точности полученных результатов** 35](#_Toc101954100)

[**6.** **АППРОКСИМАЦИЯ ФУНКЦИИ МЕТОДОМ НАИМЕНЬШИХ КВАДРАТОВ** 38](#_Toc101954101)

[**6.1.** **Описание метода** 38](#_Toc101954102)

[**6.2.** **Необходимые формулы** 39](#_Toc101954103)

[**6.3.** **Программная реализация** 39](#_Toc101954104)

[**6.4.** **Анализ и оценка точности полученных результатов** 40](#_Toc101954105)

[**7.** **ИНТЕГРИРОВАНИЕ ФУНКЦИИ КВАДРАТУРНОЙ ФОРМУЛОЙ ЭЙЛЕРА-МАКЛОРЕНА** 43](#_Toc101954106)

[**7.1.** **Описание метода с необходимыми формулами** 43](#_Toc101954107)

[**7.2.** **Программная реализация** 44](#_Toc101954108)

[**7.3.** **Анализ и оценка точности полученных результатов** 44](#_Toc101954109)

[**ЗАКЛЮЧЕНИЕ** 47](#_Toc101954110)

[**СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ** 48](#_Toc101954111)

# **ВВЕДЕНИЕ**

В курсовой работе содержатся теоретические описания методов решения задач численного анализа в областях линейной алгебры и математического анализа, их программные реализации на языке программирования Python с анализом полученных результатов (правильность, оценка точности).

Цель курсовой работы состоит в изучении и демонстрации принципа работы различных алгоритмов определенных численных методов алгебры и анализа с помощью языка программирования Python и Wolfram Mathematica.

Задачи курсовой работы:

* Определить необходимую теоретическую основу метода Галлея для нелинейных уравнений и систем нелинейных уравнений и других методов.
* Реализовать метод Галлея для систем нелинейных уравнений и прочие методы на языке программирования Python.
* Проверить корректность работы алгоритма и провести анализ результатов программы с помощью средств языков Wolfram Mathematica и Python.

# **МЕТОД ЖОРДАНА ДЛЯ РЕШЕНИЯ СИСТЕМЫ ЛИНЕЙНЫХ АЛГЕБРАИЧЕСКИХ УРАВНЕНИЙ (СЛАУ) СХЕМОЙ ЕДИНСТВЕННОГО ДЕЛЕНИЯ**

## **Описание метода с необходимыми формулами**

Метод Жордана является модификацией метода Гаусса, предназначенного для решения систем линейных алгебраических уравнений (СЛАУ). Нередко этот метод называют методом Жордана-Гаусса.

В данном методе решения СЛАУ работают с расширенной матрицей системы. Преобразования, допустимые в методе Гаусса-Жордана те же, что и в методе Гаусса.

Дано, что все главные миноры матрицы не равны нулю: .

Также дана правая часть , т. к. в данном методе рассматривается расширенная матрица системы. Вектор неизвестных переменных . В общем виде вид уравнения: .

На 1 шаге первую строку матрицы делят на элемент , а далее из всех нижележащих строк вычитают преобразованную первую строку, умноженную на первый элемент соответствующей строки, с целью получить первым элементом каждой строки, кроме первой, нуль:

и ,

где – -тая строка матрицы ).

Начиная со 2 шага, преобразованную строку (поделенную на соответствующий эл-т ) вычитают не только из нижележащих, но и из вышележащих строк:

) и ,

где .

Таким образом, на месте исходной матрицы получают единичную матрицу и тогда компоненты правой части – значения неизвестных:

,

где .

## **Программная реализация**

Входные данные: матрица и столбец правых частей .  
Реализация метода Жордана схемой единственного деления представлена на Рис. 1 и выполнена с помощью языка программирования Python и его средств, преимущественно библиотеки NumPy.

Выходные данные: вектор из значений соответствующих переменных вектора .

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 1. Программная реализация прямого метода Жордана схемой единственного деления

Помимо самого метода была написана небольшая функция для приятного глазу вывода в консоли, представленная на Рис. 2.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2. Функция для более приятного вывода в консоль

## **Анализ полученных результатов**

Для проверки правильности работы написанной функции были выбраны несколько примеров матриц и правых частей с числами с плавающей точкой. Примеры тестовых данных представлены на Рис. 3.

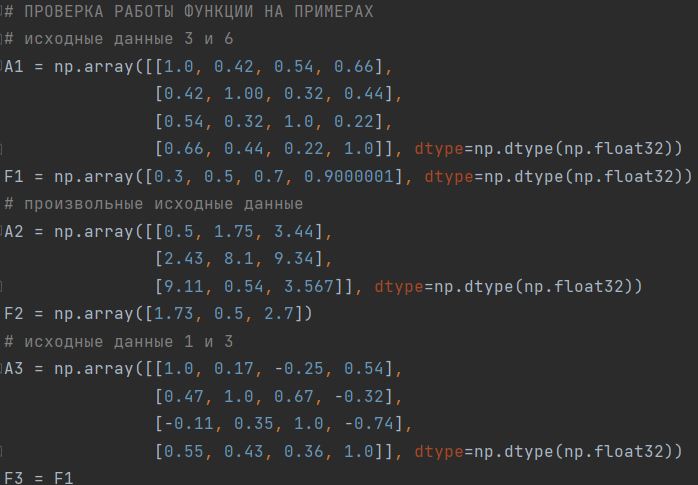


Рисунок 3. Тестовые наборы данных

Результаты работы функции представлены на рисунках 4, 5, 6.

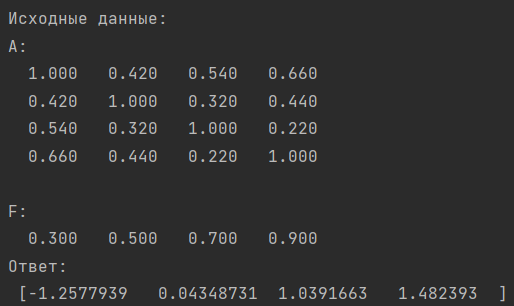


Рисунок 4. Пример работы функции (1)

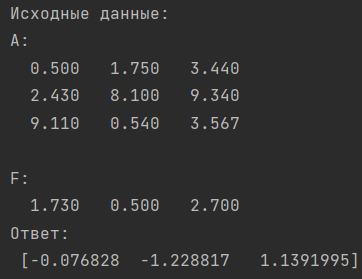


Рисунок 5. Пример работы функции (2)

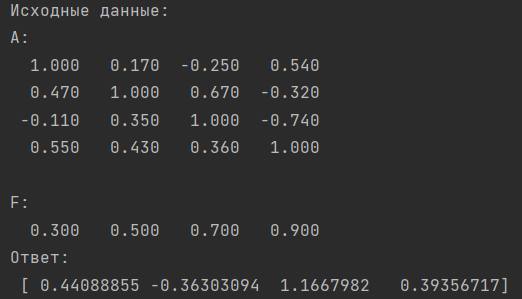


Рисунок 6. Пример работы функции (3)

Правильность полученных решений была проверена с помощью встроенной функции NumPy.linalg – solve. Результаты сравнений результатов встроенной функции и написанной функции представлены на рисунках 7, 8.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 7. Проверка на правильность полученных результатов

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 8. Вывод проверки правильности результатов

Из полученных результатов можно сделать вывод, что алгоритм и написанная функция работают довольно точно.

## **Оценка точности полученных результатов**

Была произведена оценка точности полученного значения посредством нахождения нормы вектора невязки. Норма данного вектора находится с помощью подставления полученных результатов вместо значений переменных и сравнение полученных значений с правой частью линейного неоднородного уравнения (в нашем случае). Невязка – расхождение значений правой и левой частей. Для ее вычисления была написана функция, представленная на рисунке 9, которая вычисляет норму вектора невязки. Результаты представлены на рисунках 10, 11.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 9. Функция для вычисления нормы вектора невязки

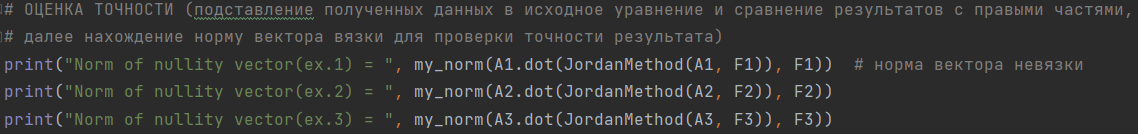


Рисунок 10. Оценка точности полученного решения

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 11. Полученные результаты погрешностей

# **РЕШЕНИЕ СЛАУ ИТЕРАЦИОННЫМ МЕТОДОМ МИНИМАЛЬНЫХ НЕВЯЗОК С ИСПОЛЬЗОВАНИЕМ 2 ТРАНСФОРМАЦИИ ГАУССА**

## **Описание метода с необходимыми формулами**

Итерационный метод минимальных невязок определяется следующим образом. Пусть последовательные приближения строятся следующим образом: {\mathbf{u}}_{k+ 1} = {\mathbf{u}}_k - \tau_k{\mathbf{r}}_k, и вектор невязки вычисляется, как {\mathbf{r}}_k = \mathbf{Au}_k - \mathbf{f}. Итерационный параметр \tau _{k} на каждой итерации выбирается так, чтобы минимизировать евклидову норму невязки {\mathbf{r}_{k+1}}. Заметим, что итерационный процесс {\mathbf{u}}_{n + 1} = {\mathbf{u}}_n + \tau_n{\mathbf{r}}_{n} может быть представлен в равносильном виде в терминах невязки {\mathbf{r}}_{n + 1} = {\mathbf{r}}_{n} + \tau_n{\mathbf{Ar}}_n. Тогда для квадрата евклидовой (третьей) нормы невязки получаем условие

({\mathbf{r}}_{{k+ 1}},{\mathbf{r}}_{{k+ 1}}) = ({\mathbf{r}}_k,{\mathbf{r}}_k) - 2\tau_k({\mathbf{Ar}}_k,{\mathbf{r}}_k) + \tau_k^2 ({\mathbf{Ar}}_k,{\mathbf{Ar}}_k).

Для отыскания минимума невязки на следующей итерации приравняем нулю производную последнего выражения по итерационному параметру \tau _{k}. Получим равенство

- 2({\mathbf{Ar}}_k,{\mathbf{r}}_k) + 
 2\tau_k({\mathbf{Ar}}_k,{\mathbf{Ar}}_k) = 0.

Из последнего соотношения находим значение итерационного параметра

$ \tau_k= \frac{({\mathbf{Ar}}_k,{\mathbf{r}}_k)}{({\mathbf{Ar}}_k,{\mathbf{Ar}}_k)}. $

Тут же стоит упомянуть 2 трансформацию Гаусса: система

Изображение выглядит как текст

Автоматически созданное описание

где – матрица , заменяется системой

Изображение выглядит как текст

Автоматически созданное описание

где - транспонированная к матрица, и полученная система оказывается совместной и имеет единственное решение. Нормальное решение исходной системы получается по формуле



Эта трансформация помогает сделать матрицу системы гарантированно симметричной (использование свойств и => Изображение выглядит как текст

Автоматически созданное описание), что как раз и будет подходить для метода минимальных невязок.

## **Программная реализация**

Входные данные: матрица , столбец правых частей , кол-во итераций .

Выходные данные: на каждом шаге выводится номер итерации, приближенное значение вектора , а также норма вектора невязки. После выполнения всех итераций выводится «конечный» ответ.

Реализация метода минимальных невязок с использованием 2 трансформации Гаусса представлена на рисунке 1 и выполнена с помощью языка программирования Python и его средств, преимущественно библиотеки NumPy.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 1. Реализация метода минимальных невязок на языке Python

## **Анализ и оценка точности полученных результатов**

Входные тестовые данные представлены на рисунке 2.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2. Тестовые наборы данных

Результат работы представлен на рисунках 3, 4, 5, 6. Изначально выводятся входные данные, далее на каждом шаге итерации выводится номер итерации, приближенное решение. Также на каждом шаге выводится норма вектора невязки, соответствующая каждому последовательному приближенному решению, благодаря которой можно оценить точность полученного решения, полученного на каждой итерации. Т.к. критерий сходимости данного метода довольно мал, вместо вывода результата каждой итерации, на примере выводятся шаги итераций, кратные 10, чтобы лучше увидеть динамику уменьшения нормы вектора.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3. Изначальный вывод входных данных

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 4. Пример работы программы на примере (результаты на первых 30-ти итерациях)

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 5.Пример работы программы на примере (результаты на 40-70 итерациях)

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 6. Пример работы программы на примере (результаты на 80-100 итерациях и конечный результат)

# **РЕШЕНИЕ СИСТЕМЫ НЕЛИНЕЙНЫХ АЛГЕБРАИЧЕСКИХ УРАВНЕНИЙ (СЛНУ) С ПОМОЩЬЮ МЕТОДА ГАЛЛЕЯ (ХЭЛЛИ)**

## **Описание метода с необходимыми формулами**

В основе метода Галлея для решения систем нелинейных уравнений лежит вид квадратичной интерполяции векторной функции  обратной к . Каждая функция в системе должна иметь непрерывную 2 производную.

 .

Возьмем разложение функции  в ряд Тейлора второго порядка: ,

где 1 слагаемое – вектор, 2 слагаемое – матрица Якоби, умноженная на вектор, а  – матрица Гессе , и в 3 слагаемом при умножении 3-мерной матрицы 2 раза на вектор получаем вектор.

Обозначим  – начальное приближение, , ,. Также для простоты последующих обозначений введем параметры  – Ньютоновскую поправку и  – Чебышевскую поправку. Таким образом, получим формулу

, .

Стоит упомянуть, что существует модифицированный метод, в котором меняется лишь Ньютоновская поправка  и вычисляется она, как .

Метод также существует для функции одной действительной переменной с непрерывной 2 производной. Метод Галлея точно находит корни линейно-линейного приближения Паде к функции, в отличие от метода Ньютона или метода секущих, которые линейно аппроксимируют функцию.

Возьмем ряд Тейлора второго порядка



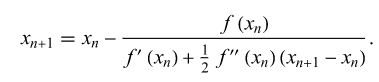
Корень *f(x)* удовлетворяет *f(x)=0*, таким образом, получаем



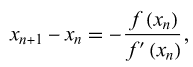
Теперь напишем



из чего получим

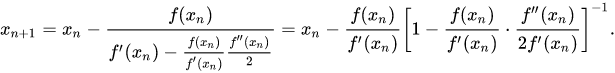


Используя результат метода Ньютона, получаем

Изображение выглядит как текст

Автоматически созданное описание.

Результат можно упростить и написать в иной формулировке, которая показывает сходство метода Галлея с методом Ньютона



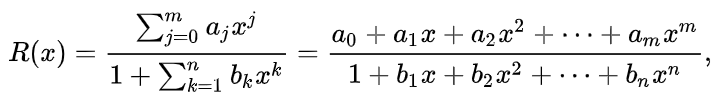
Здесь стоит объяснить Паде-аппроксимацию, так метод Галлея основан как раз таки на ней. Аппроксимация Паде — классический метод рациональной аппроксимации аналитических функций, названный по имени французского математика Анри Паде. Метод заключается в представлении функции в виде отношения двух полиномов, коэффициенты которых определяются коэффициентами разложения функции в ряд Тейлора.

Пусть задан степенной ряд

*,*

представляющий функцию 𝑓(*x*). Данное разложение является исходным пунктом любого анализа, использующего аппроксимацию Паде.

Для функции *f* и двух целых чисел *m* ≥ 0 и *n* ≥ 1 аппроксимация Паде порядка *[ m / n ]* является рациональной функцией



что согласуется с *f ( x )* в максимально возможном порядке, который составляет

Изображение выглядит как текст

Автоматически созданное описание

Стоит упомянуть, что в основе метода Галлея лежит Паде-аппроксимация функции *x=g(y)* обратнойк функции *y=f(x)*. Метод Галлея предполагает аппроксимацию, в которой используется совпадение разложений вплоть до 2 порядка (в числителе и в знаменателе стоят полиномы 1 степени, т. е. x=[1/1]g(y), где в зависимости от метода х и у либо векторные функции, либо функции 1 переменной).

Аппроксимация Паде единственна для данных *m* и *n*, то есть коэффициенты  могут быть определены однозначно. По причинам уникальности член нулевого порядка в знаменателе *R ( x )* был выбран равным 1(*b0*=1), иначе числитель и знаменатель *R ( x )* были бы уникальными только с точностью до умножения на константу.

Эквивалентно, если *R ( x )* раскрывается в ряд Маклорена (ряд Тейлора в 0), его первые *m+n* членов аннулируют первые *m+n* членов *f ( x )*, из этого следует

.

## **Программная реализация**

Входные данные: массив функций (система нелинейных уравнений), начальное приближение, кол-во итераций и  для критерия остановки .

Выходные данные: массив длины n, где n – кол-во переменных, с приближенным корнем уравнения. Дополнительно печатается поданное на вход кол-во итераций. Также для анализа результатов выдается начальное приближение и текущее приближение вектора х на каждой итерации. По завершении итерационного процесса печатается вектор невязки, который получается с помощью вычисления значений каждой из функций системы в полученной точке.

Программная реализация метода Галлея представлена на рис. 1, 2 и выполнена с помощью языка программирования Python и его средств, преимущественно библиотеки NumPy и SciPy.

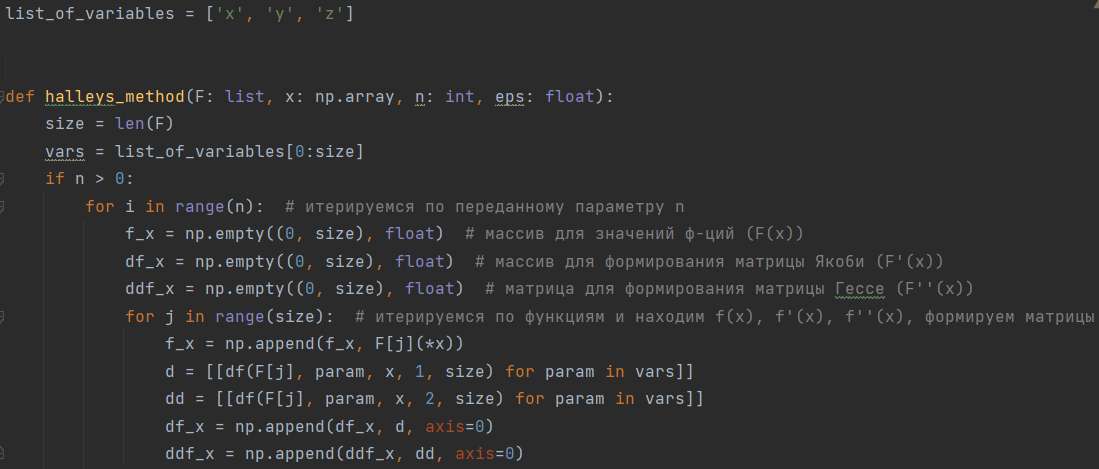


Рисунок 1. Программная реализация метода Галлея (1)

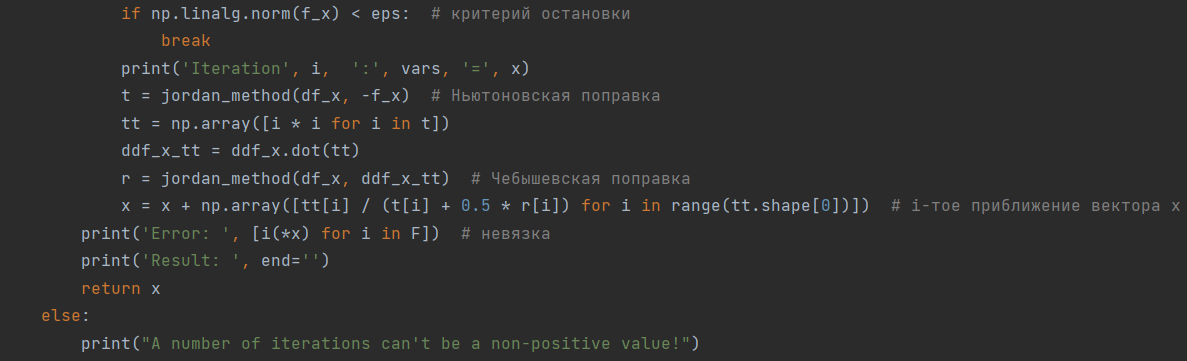


Рисунок 2. Программная реализация метода Галлея (2)

Для вычисления производной n-ого порядка по параметру, n=1,2,3, была написана функция df(), представленная на рис. 3. В ней на вход подается функция, параметр, по которому необходимо дифференцировать, порядок производной и кол-во переменных в уравнении.

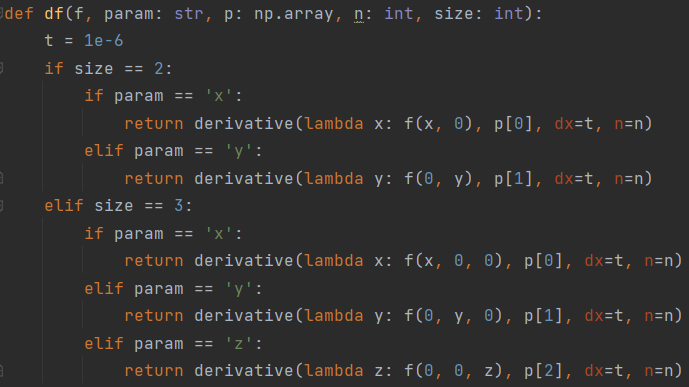


Рисунок 3. Вспомогательная функция вычисления производной

Также использовался ранее представленный в параграфе 1 немного модифицированный метод Жордана для решений систем уравнений, использующийся для нахождения Ньютоновской и Чебышевской поправок.

## **Анализ полученных результатов**

Несколько примеров работы функции представлено на рис. 4, 5, 6, 7.   
 во всех случаях бралось равным .

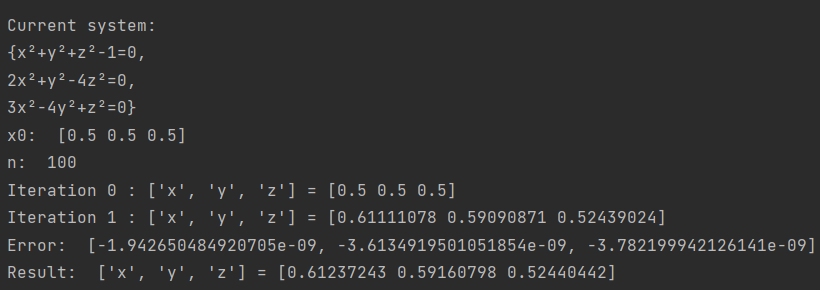


Рисунок 4. Пример работы функции (1)

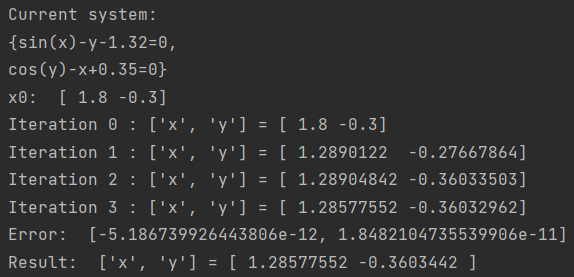


Рисунок 5. Пример работы функции (2)

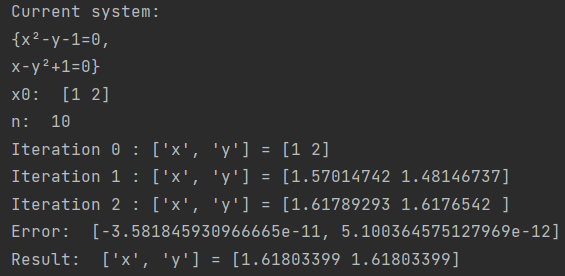


Рисунок 6. Пример работы функции (3)

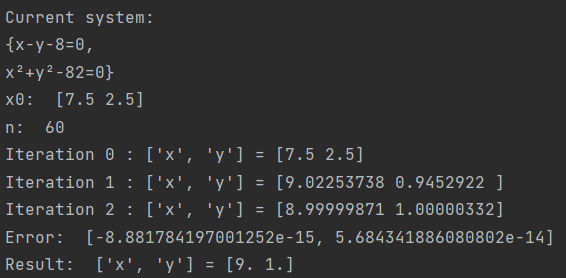


Рисунок 7. Пример работы функции (4)

## **Оценка точности полученных результатов**

Оценить точность полученных результатов можно с помощью встроенных функций Wolfram Mathematica и увидеть результаты с помощью графиков. На рисунках 8, 9, 10 представлены результаты решения некоторых систем нелинейных уравнений встроенной функцией NSolve(), а также визуализированы полученные результаты: поданные на вход кривые системы, и полученная точка.

Изображение выглядит как текст

Автоматически созданное описание

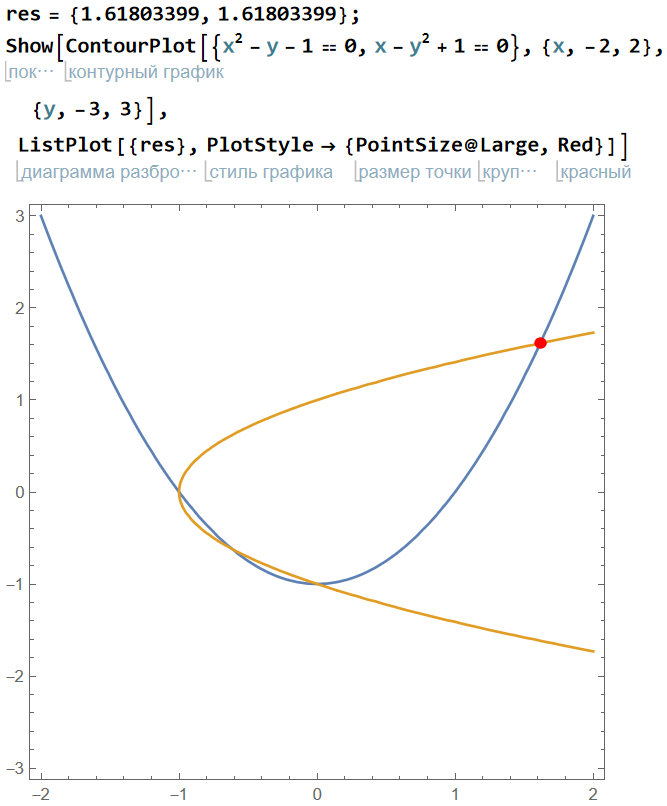


Рисунок 8. Оценка полученных результатов на примере 1

Изображение выглядит как текст

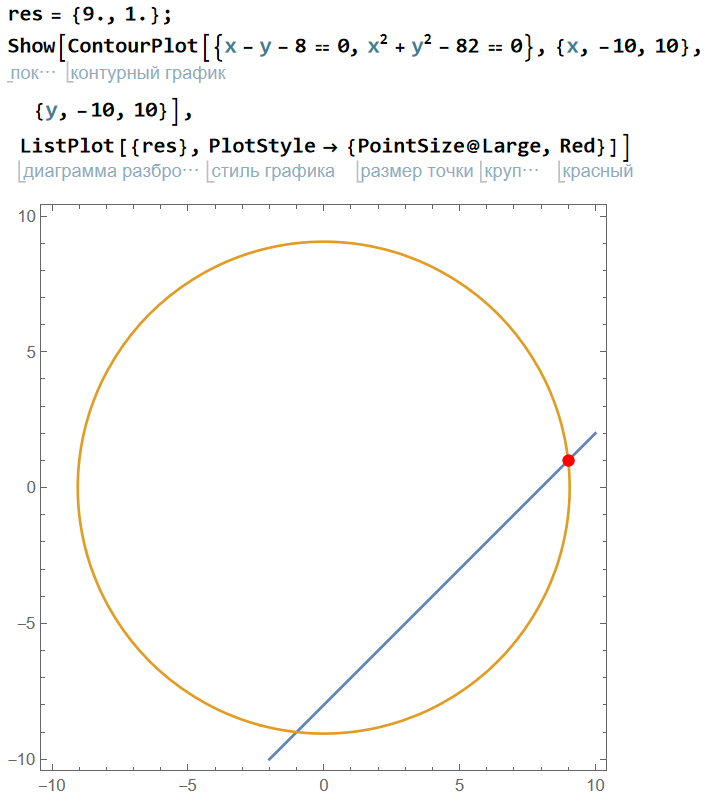
Автоматически созданное описание

Рисунок 9. Оценка полученных результатов на примере 2

Изображение выглядит как карта

Автоматически созданное описание

Рисунок 10. Оценка полученных результатов на примере 3

# **РЕШЕНИЕ ПРОБЛЕМЫ СОБСТВЕННЫХ ЗНАЧЕНИЙ ДЛЯ МАТРИЦЫ МЕТОДОМ ОБРАТНЫХ ИТЕРАЦИЙ СО СДВИГОМ**

## **Описание метода с необходимыми формулами**

Напишем итерационный процесс, обратный по отношению к степенному процессу:



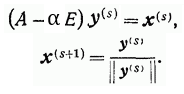
Он сходится к наибольшему по модулю собственному значению матрицы , т. е. к наименьшему по модулю собственному значению матрицы (ибо собственные значения матриц А и обратны друг другу). Сходимость в данном случае будет довольно медленной.

Однако положение можно существенно улучшить методом сдвига, который заключается в следующем. Пусть нам приближенно известно некоторое, не обязательно наименьшее, собственное значение α. Тогда так называемая сдвинутая матрица будет иметь собственные значения λ-α. У этой матрицы интересующее нас собственное значение  будет намного меньше по модулю, чем остальные. Поэтому обратные итерации со сдвинутой матрицей



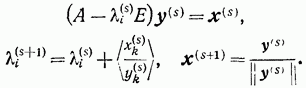
будут быстро сходиться и определят требуемое нам собственное значение .

После каждой итерации надо нормировать вектор, чтобы избежать переполнений. С учетом этого получим последовательность формул



Если исходное приближение было хорошим, то иногда процесс сходится за несколько, итераций; тогда выгодно непосредственно решать линейную систему . Если же требуемое число итераций велико, то лучше обратить матрицу .

Если сдвиг постоянный, то итерации сходятся линейно. Можно получить квадратичную сходимость, если уточнять сдвиг в ходе расчета следующим образом:



Для матриц, имеющих ортогональную систему собственных векторов (например, эрмитовых матриц), сходимость вблизи корня будет даже кубической. Заметим, что допускать слишком точное совпадение  с собственным значением нельзя, ибо матрица системы становится плохо обусловленной. Поэтому, когда в ходе итераций у величины устанавливаются (т. е. перестают меняться) 5—7 знаков, то итерации следует прекращать.

## **Программная реализация**

Входные данные: матрица А, сдвиг alpha, кол-во итераций.

Выходные данные: в написанной функции выдается собственный вектор матрицы A и ее собственное значение, соответствующее полученному собственному вектору.

Программная реализация метода обратных итераций со сдвигом представлена на рис. 1, а функция получения собственного значения get\_eigenvalue() представлена на рис. 2 (разделила на 2 функции лишь для удобства). Также использовался метод для «приятного глазу» вывода, представленный в параграфе 1.2. Также выводится норма вектора невязки, вычисляемого, как и номер последней итерации (т. е. номер итерации, когда у нормы вектора невязки устанавливаются (т. е. перестают меняться) 5 знаков).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 1. Реализация метода обратных итераций со сдвигом

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2. Реализация функции получения собственного значения, соответствующего переданному собственному значению

Примеры работы программы можно увидеть на рис. 3, 4, 5.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3. Пример 1

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 4.Пример 2

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 5.Пример 3

## **Анализ полученных результатов**

Полученный результаты можно сравнить с результатами встроенной функции Wolfram Mathematica. Результаты представлены на рис. 6, 7, 8. Полученные написанной функцией собственный вектор и значение выделены светло-голубым цветом.

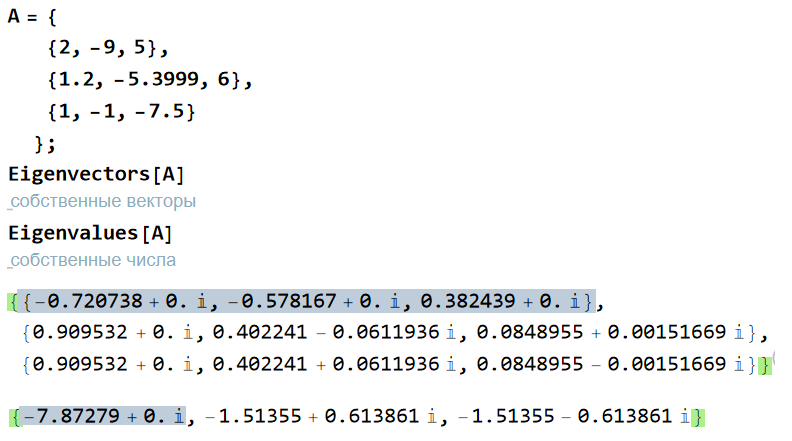


Рисунок 6. Пример 1

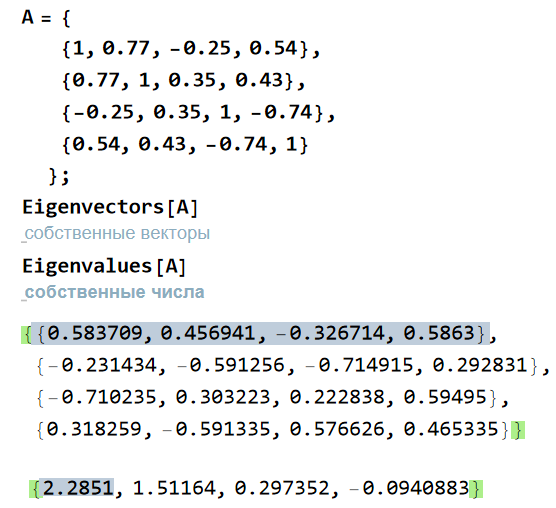


Рисунок 7.Пример 2

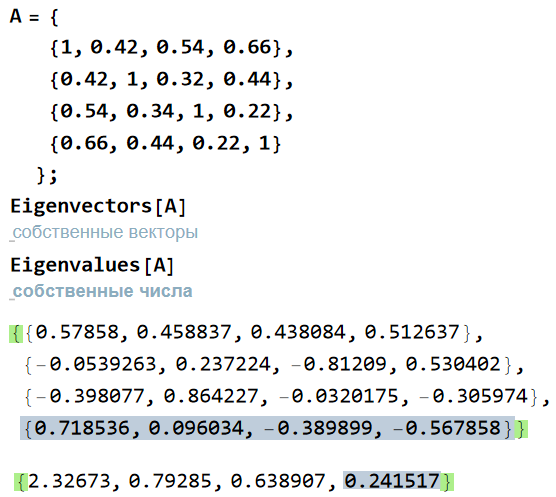


Рисунок 8.Пример 3

# **АППРОКСИМАЦИЯ ФУНКЦИИ ИНТЕРПОЛЯЦИОННОЙ ФОРМУЛОЙ БЕССЕЛЯ ДЛЯ РАВНООТСТОЯЩИХ УЗЛОВ**

## **Описание метода с необходимыми формулами**

Пусть есть равноотстоящих узлов интерполирования

где , и известны значения аппроксимируемой функции в этих точках:

Выбрав начальными значениями используем вторую интерполяционную формулу Гаусса

Изображение выглядит как текст

Автоматически созданное описание

где . Теперь, приняв за начальные значения , используем первую интерполяционную формулу Гаусса

Изображение выглядит как текст

Автоматически созданное описание

Взяв среднее арифметическое этих формул, после несложных преобразований получим интерполяционную формулу Бесселя

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст, кроссворд, квитанция

Автоматически созданное описаниегде - конечные разности, вычисляемые по таблице ниже.

## **Программная реализация**

Входные данные: массив равноотстоящих узлов, массив значений в этих точках, массив точек, в которых требуется посчитать значение аппроксимированной функции или одна точка.

Выходные данные: массив значений, посчитанных формулой.

Программная реализация аппроксимации функции интерполяционной формулой Бесселя для равноотстоящих узлов, вспомогательные функции и основная реализация функции интерполирования в точке или в точках (зависит от последнего входного значения) представлена на Рисунок 1, Рисунок 2.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 1. Реализация аппроксимации интерполяционной формулой Бесселя для равноотстоящих узлов и 2 вспомогательные функции

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2. Обобщенная функция интерполирования

Также отдельно написаны тестовые функции для проверки работы функции и прорисовки графиков функций и приближений к ним. Они представлены на Рисунок 3, Рисунок 4.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3. Тестовые функции (1, 2)

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 4. Тестовые функции (3)

## **Анализ и оценка точности полученных результатов**

Проверим точность полученного результата, посчитав значение интерполяционного полинома в тех же точках, по которым он строился и вычислим норму ошибки. Тестовая функция для данной реализации и результат (т. е. норма невязки) представлены на Рисунок 5.

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 5. Тестовая функция (4) и выдаваемый результат

Как видно есть небольшая ошибка, но она скорее связана с тем, что для 21 точки с вычисляемыми значениями будет происходить потеря точности при арифметических операциях (к примеру, вычисляемый факториал по формуле будет порядка 1019). Также видно, что не возникло никаких ошибок и все работает корректно.

Протестируем функцию на случайных равноудаленных точках и выведем получившиеся графики (test\_1()) (*Рисунок 6*). На этом и всех последующих рисунках оранжевый график – исходная функция, синий – то, что получилось при интерполяции.

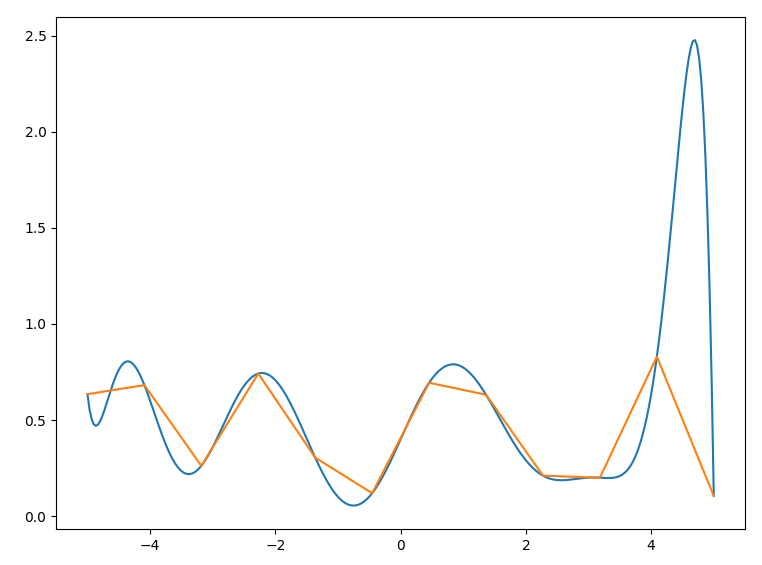


Рисунок 6. Визуализация полученных результатов (1)

Видно, что получившаяся кривая проходит через все узловые точки.

Теперь протестируем функцию на конкретных функциях (test\_2(), test\_3()).

1. Функция:

Результат:

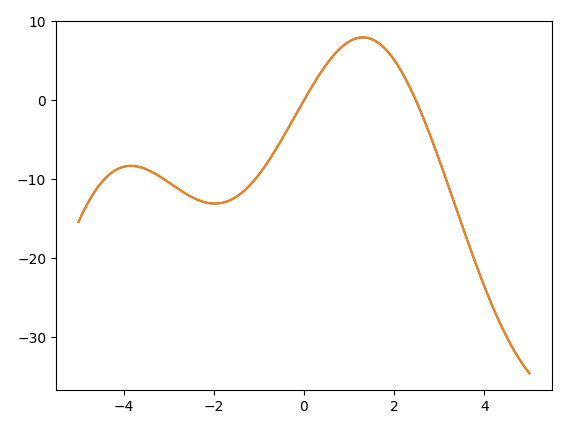


Рисунок 7. Визуализация полученных результатов (2)

1. Функция:

Результат:

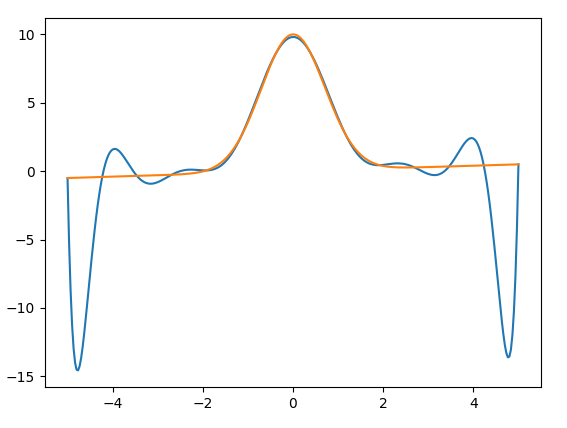


Рисунок 8. Визуализация результатов (3)

Во 2 примере (Рисунок 7) видно, что функция достаточно хорошо аппроксимирует заданную функцию так, что на рисунке даже не видно наличие двух графиков.

Однако в 3 примере интерполяция дает хороший результат только вблизи середины графика и очень плохую на концах. Это связано с тем, что набор линейно независимых функций, используемый в аппроксимации в данном алгоритме – полиномы, которые так и должны себя вести. Именно в этом примере функция должна асимптотически сходиться к прямой , однако полином n-й степени, n ≠ 1, как известно, не будет себя так вести. Поэтому в данном случае интерполяция не будет давать точных результатов, как и для достаточно больших по модулю аргументов.

# **АППРОКСИМАЦИЯ ФУНКЦИИ МЕТОДОМ НАИМЕНЬШИХ КВАДРАТОВ**

## **Описание метода**

Есть *n* точек и значения функции в этих точках. Также имеется набор линейно независимых функций. Нужно найти аппроксимацию данной функции используя метод наименьших квадратов.

Нужная функция будет линейной комбинацией данного набора линейно независимых функций. В общем случае их может быть сколько угодно. Пусть их *n* штук. Тогда значение этой функции в точке *x* будет равно:

где φi – линейно независимая функция. Чтобы пользоваться этой формулой, нужно найти *ai*.

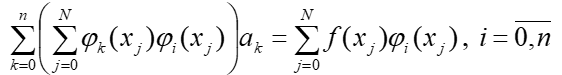
Используя метод наименьших квадратов, приходим к выражению:

Изображение выглядит как текст, часы, датчик

Автоматически созданное описание

Другими словами, нужный набор *ak* будет таков, что сумма квадратов разности исходной функции и получившейся по всем *xj* будет минимальной. В этом и заключается идея нахождения навой функции.

Из получившегося выше выражения будет вытекать следующее:



Можно увидеть, что это система линейных уравнений, которая имеет единственное решение, так как матрица здесь является матрицей Грамма, построенной из линейно независимых функций, и в таком случае она невырожденная.

Остается только решить эту систему и получить набор *ak*.

## **Необходимые формулы**

Элементы матрицы грамма:

Столбец свободных коэффициентов:

Набор линейно независимых функций будет вида

*,*

где *k* пробегает значения от 0 до *N*-1.

## **Программная реализация**

Программная реализация метода наименьших квадратов представлена на *Рисунок 1*.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 1. Программная реализация метода наименьших квадратов

## **Анализ и оценка точности полученных результатов**

Рассмотрим функцию:

и возьмем 10 полиномов. Тестовая функция представлена на *Рисунок 2.*

*Изображение выглядит как текст

Автоматически созданное описание*

Рисунок 2. Тестовая функция (1)

Результат, представленный на *Рисунок 3* достаточно хороший.

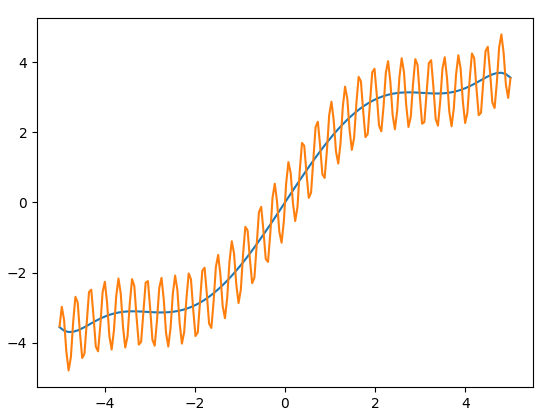


Рисунок 3. Результат отработки функции с использованием 10 полиномов (1)

Теперь возьмем чуть меньше полиномов – 5. Результат представлен на *Рисунок 4*.

Изображение выглядит как текст, музыка

Автоматически созданное описание

Рисунок 4. Результат отработки функции с использованием 5 полиномов (1)

Как видно на обоих примерах, функция хорошо аппроксимируется.

Рассмотрим другой пример функции:

Тестовая функция представлена на *Рисунок 5*.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 5. Тестовая функция (2)

Возьмем так же 10 и 5 полиномов. Результаты работы представлены на *Рисунок 6* и *Рисунок 7*.

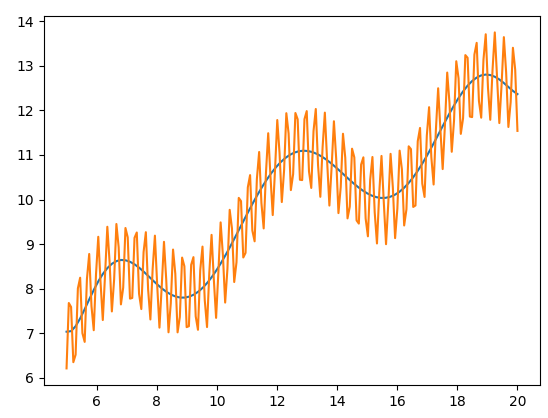


Рисунок 6. Результат отработки функции с использованием 10 полиномов (2)

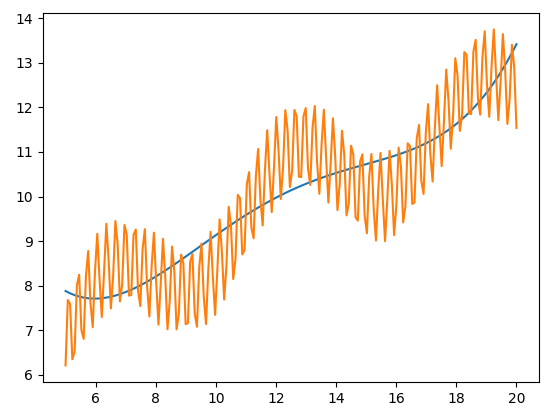


Рисунок 7. Результат отработки функции с использованием 5 полиномов (2)

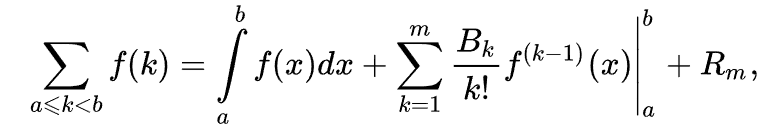
На последнем графике видно, что функция аппроксимируется не так хорошо, как в предыдущем примере. Из этого можно сделать вывод, что по мере построения реализации нужно делать выбор в пользу времени работы или качества получаемого решения.

Как видно, реализация работает и дает хороший результат для данного набора полиномов.

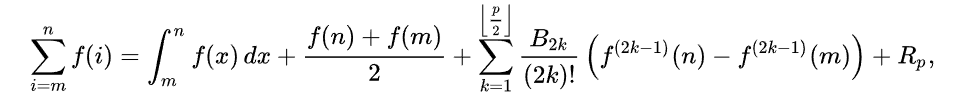
# **ИНТЕГРИРОВАНИЕ ФУНКЦИИ КВАДРАТУРНОЙ ФОРМУЛОЙ ЭЙЛЕРА-МАКЛОРЕНА**

## **Описание метода с необходимыми формулами**

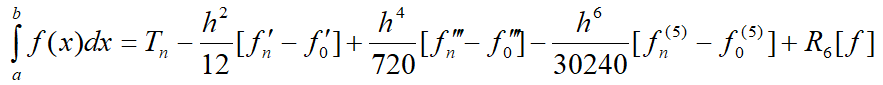
Формула Эйлера — Маклорена имеет вид



где – натуральное число, – числа Бернулли, – достаточно гладкая функция, чтобы иметь производные – остаточный член.

Данная формула часто записывается с индексом, принимающим только четные значения, так как нечетные числа Бернулли равны нулю, за исключением B1. В этом случае мы имеем

Эта формула дает возможность считать приближенные значения сумм, однако ее можно использовать в обратную сторону, чтобы считать приближенные значения интегралов. Для этой задачи она примет вид



где – пределы интегрирования, так что .

Тогда, используя эту формулу, можно считать приближенные значения интеграла функции.

## **Программная реализация**

Входные данные: интегрируемая впоследствии функция, промежуток интегрирования и кол-во шагов.

Выходные данные: численный результат интеграла функции на заданном промежутке.

Изображение выглядит как текст, электроника

Автоматически созданное описаниеПрограммная реализация метода интегрирования функции квадратурной формулой Эйлера-Маклорена представлена на *Рисунок 1*.

Рисунок 1. Программная реализация метода интегрирования функции квадратурной формулой Эйлера-Маклорена

## **Анализ и оценка точности полученных результатов**

Будем сравнивать результаты написанной функции с встроенной функцией языка Python, который интегрирует функцию на промежутке (scipy.integrate.quad()).

Рассмотрим функцию

Первая тестовая функция представлена на Рисунок 2.

Изображение выглядит как текст

Автоматически созданное описание  
Рисунок 2. Тестовая функция (1)

Для начала примем n = . Результат работы тестовой функции представлен на Рисунок 3.

Изображение выглядит как текст

Автоматически созданное описание  
Рисунок 3. Результат тестовой функции при n=1000

Как видно, погрешность большая.

Увеличим n до Результат представлен на Рисунок 4.

  
Рисунок 4. Результат тестовой функции при n=10000

В этом случае погрешность намного меньше и можно сделать вывод, что алгоритм работает правильно, так как при увеличении числа точек значение интеграла стало довольно точным.

Теперь рассмотрим функцию

Вторая тестовая функция представлена на Рисунок 5.

Изображение выглядит как текст

Автоматически созданное описание  
Рисунок 5. Тестовая функция (2)

Так же, как и в предыдущем примере, примем n = . Результат

работы тестовой функции представлен на *Рисунок 6*.

  
Рисунок 6. Результат тестовой функции при n=1000

Увеличим n до Результат представлен на Рисунок 7.

  
Рисунок 7. Результат тестовой функции при n=10000

В данном случае наблюдаем аналогичную ситуацию – при увеличении кол-ва точек растет точность.

Подведя итоги, можно сделать вывод, что алгоритм действительно работает правильно. Реализация на языке программирования Python дает хорошие результаты, а полученные результаты согласуются с теоретическими данными.

# **ЗАКЛЮЧЕНИЕ**

Были выполнены поставленные задачи, а именно:

* изучить теорию по данным методам;
* написать программную реализацию методов с использованием теоретических данных;
* проверить правильность работы методов;
* проанализировать полученные результаты.

В дальнейшей перспективе целесообразно освоение других вычислительных методов и применение их на практике.

# **СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ**

1. В. Б. Хазанов. Конспект лекций «Численные методы алгебры»
2. В. Б. Хазанов. Конспект лекций «Численные методы анализа»
3. В. В. Воеводин. Численные методы алгебры. Теория и алгоритмы. Изд-во «Наука»,  
   Москва, 1966
4. У. Г. Пирумов [и др.] Численные методы: учебник и практикум под ред. У. Г. Пирумова. — 5-е изд., перераб. и доп. — М.: Издательство Юрайт, 2018
5. Калиткин Н. Н. Численные методы. – СПб.: БХВ, 2011
6. Калиткин Н. Н., Альшина Е. А. Численные методы. Книга 1 Численный анализ. – М.:  
   Академия, 2013
7. Бахвалов И. С., Жидков Н. П., Кобельков Е. З. Численные методы. – М.: Изд. МГУ, 2007
8. Вержбицкий В. М. Численные методы. Линейная алгебра и нелинейные уравнения. – М.: Директ-Медиа, 2013
9. QR-алгоритм [электронный ресурс] / режим доступа: <https://www.hmong.press/wiki/Pad%C3%A9_approximant> , свободный. — Загл. с экрана. — Яз. рус.
10. Halley's Method [Электронные ресурс] / режим доступа: <https://mathworld.wolfram.com/HalleysMethod.html> , свободный. — Загл. с экрана — Яз. англ.