Министерство образования и науки РФ

ФГБПОУ ВПО Тульский государственный университитет

КАФЕДРА АВТОМАТИКИ И ТЕЛЕМЕХАНИКИ

СРЕДНЕКВАДРАТИЧЕСКОЕ ПРИБЛИЖЕНИЕ ФУНКЦИЙ

Лабораторная работа № 3

по курсу «Вычислительный практикум»

Вариант № 4

Выполнил: студент группы 220601 ________ Белым А.А.

(подпись)

Проверил: к. т. н., доцент ________ Карцева А.С.

(подпись)

Тула 2011

Цель работы

Цель работы заключается в том, чтобы изучить методы среднеквадратического приближения и написать программу, реализующий один из таких методов.

Задание на работу

Выполнить среднеквадратическое приближение по тригонометрическому базису для функции:

$$f(x) = x^4 + 3x - 1$$

Теоретическая справка

Пусть значения приближаемой функции $f(x)$ заданы в $N+1$ узлах: $f(x_0), ..., f(x_N)$.

Аппроксимирующую функцию будем выбирать из некоторого параметрического семейства $F(x, \vec{c})$, где

$\vec{c} = (c_0, c_1, ..., c_n)^T$ - вектор параметров, $N > n$.

Принципиальным отличием задачи среднеквадратического приближения от задачи интерполяции является то, что число узлов превышает число параметров. В данном случае практически всегда не найдется такого вектора параметров, для которого значения аппроксимирующей функции совпадали бы со значениями аппроксимируемой функции во всех узлах.

В этом случае задача аппроксимации ставится как задача поиска такого вектора параметров $\vec{c} = (c_0, c_1, ..., c_n)^T$, при котором значения аппроксимирующей функции $F(x, \vec{c})$ как можно меньше отклонялись бы от значений аппроксимируемой функции в совокупности всех узлов.

Можно записать различные критерии близости аппроксимируемой и аппроксимирующей функции.

Наиболее известными и часто используемыми являются

критерий равномерного приближения:

$$J(\vec{c}) = \sum_{i=0}^{N} (f(x_i) - F(x, \vec{c}) \to \min \qquad (1)$$

критерий среднеквадратического приближения (метод наименьших квадратов):

$$J(\vec{c}) = \sqrt{\sum_{i=0}^{N}(f(x_i) - F(x,\vec{c})^2} \to \min \qquad (2)$$

В обоих случаях коэффициенты многочлена выбирают исходя из условия

$$\vec{c} = arg_{\vec{c}} min J(\vec{c})$$

.

Подкоренное выражение в функции (2) представляет собой квадратичную функцию относительно коэффициентов аппроксимирующего многочлена. Она непрерывна и дифференцируема по $c_0, c_1, ..., c_n$. Очевидно, что ее минимум находится в точке, где все частные производные равны нулю

$\frac{\partial J(\vec{c})}{\partial c_m}, m = 0, 1..., n$.

Приравнивая к нулю частные производные, получим систему линейных алгебраических уравнений относительно неизвестных (искомых) коэффициентов многочлена наилучшего приближения.

Метод наименьших квадратов может быть применен для различных параметрических функций, но часто в инженерной практике в качестве аппроксимирующей функции используются многочлены по какому-либо линейно независимому базису $\{\phi_k(x), k = 0, 1, ..., n\}$:

$F(x,\vec{c}) = \sum_{k=0} n c_k \phi_k(x)$.

В этом случае СЛАУ для определения коэффициентов будет иметь вполне определенный вид:

$$\begin{cases} c_1 a_0 0 & + & c_2 a_0 1 & + & ... & + & c_n a 0_n & = b_1 \\ c_1 a_1 0 & + & c_2 a_1 1 & + & ... & + & c_n a 1_n & = b_2 \\ & ... & & & ... & & & ... \\ c_1 a_n 0 & + & c_2 a_n 1 & + & ... & + & c_n a n_n & = b_n \end{cases}$$

,

$a_k j = \sum_{i=0} N \phi_k(x_i)\phi_j(x_i), b_j = \sum_{i=0} N \phi_j(x_i) f(x_i)$.

Чтобы эта система имела единственное решение необходимо и достаточно, чтобы определитель матрицы А (определитель Грама) был отличен от нуля. Для того, чтобы система имела единственное решение необходимо и достаточно чтобы система базисных функций $\phi_k(x), k = 0, 1, ..., n$ была линейно независимой на множестве узлов аппроксимации. Очень распространено среднеквадратическое приближение многочленами по степенному базису $\{x_k, k = 0, 1, ..., n\}$ и по тригонометрическому

$$\{\phi_0(x) = 1, \phi_1(x) = \sin(x), \phi_2(x) = \cos(x), \phi_3(x) = \sin(2x), \phi_4(x) = \cos(2x), ...\}$$

или

$$\phi_k = \begin{cases} \cos(nx), & k = 2n, \\ \sin((n+1)x), & k = 2n+1 \end{cases} \quad k = 0, 1, ..., N.$$

Схема алгоритма

На рисунке 1 представлена схема алгоритма расчета коэффициентов аппроксимирующей функции на основе тригонометрического базиса.

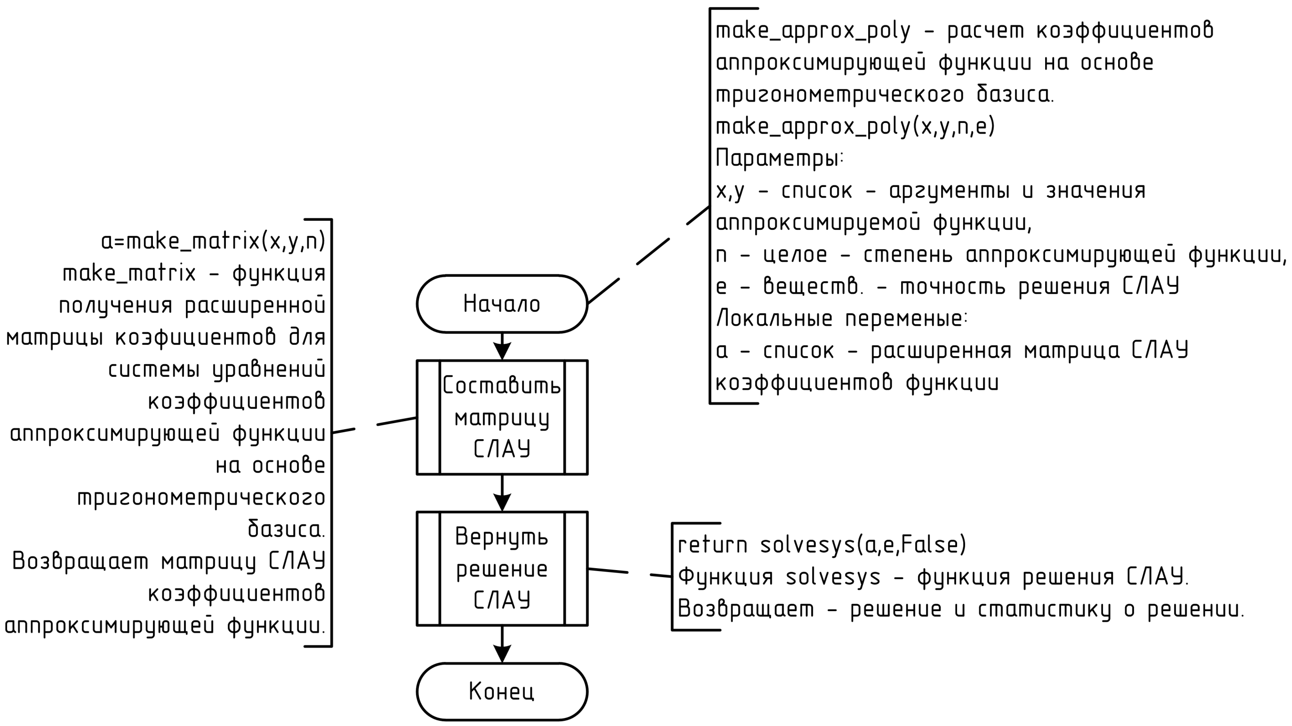

Рисунок 1 - Схема алгоритма расчета коэффициентов аппроксимирующей функции

На рисунке 2 представлена схема алгоритма расчета значения члена тригонометрического базиса.

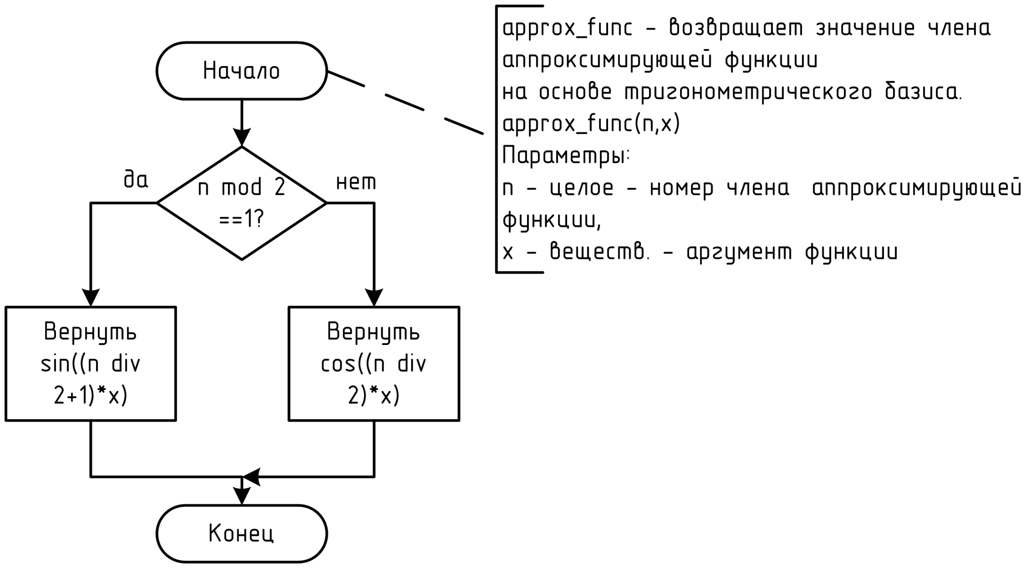

Рисунок 2 - Схема алгоритма расчета члена тригонометрического базиса

На рисунке 3 представлена схема алгоритма получения матрицы СЛАУ коэффициентов аппроксимирующей функции на основе тригонометрического базиса.

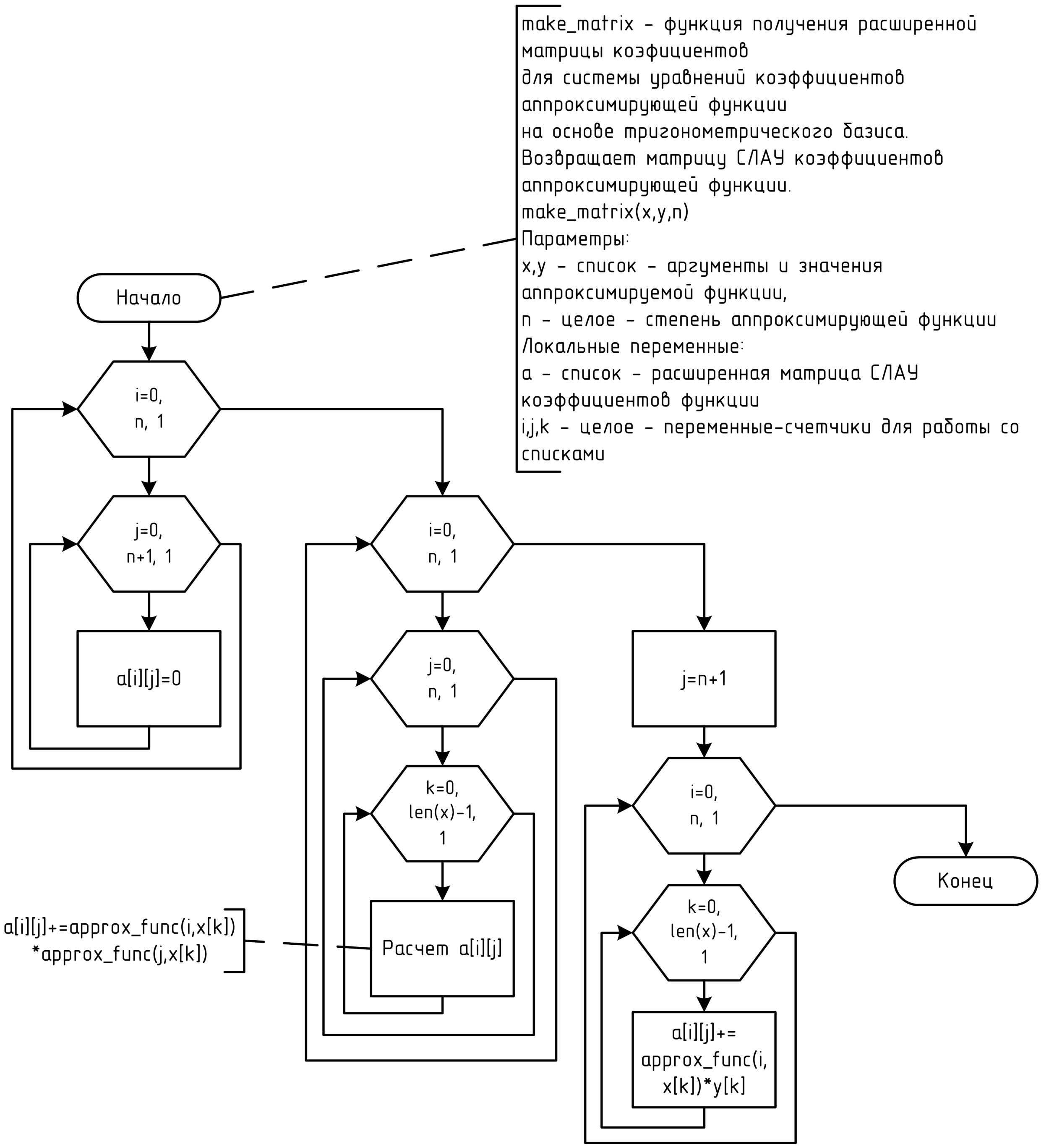

Рисунок 3 - Схема алгоритма получения матрицы СЛАУ

На рисунке 4 представлена схема обобщенного алгоритма решения системы линейных алгебраических уравнений.

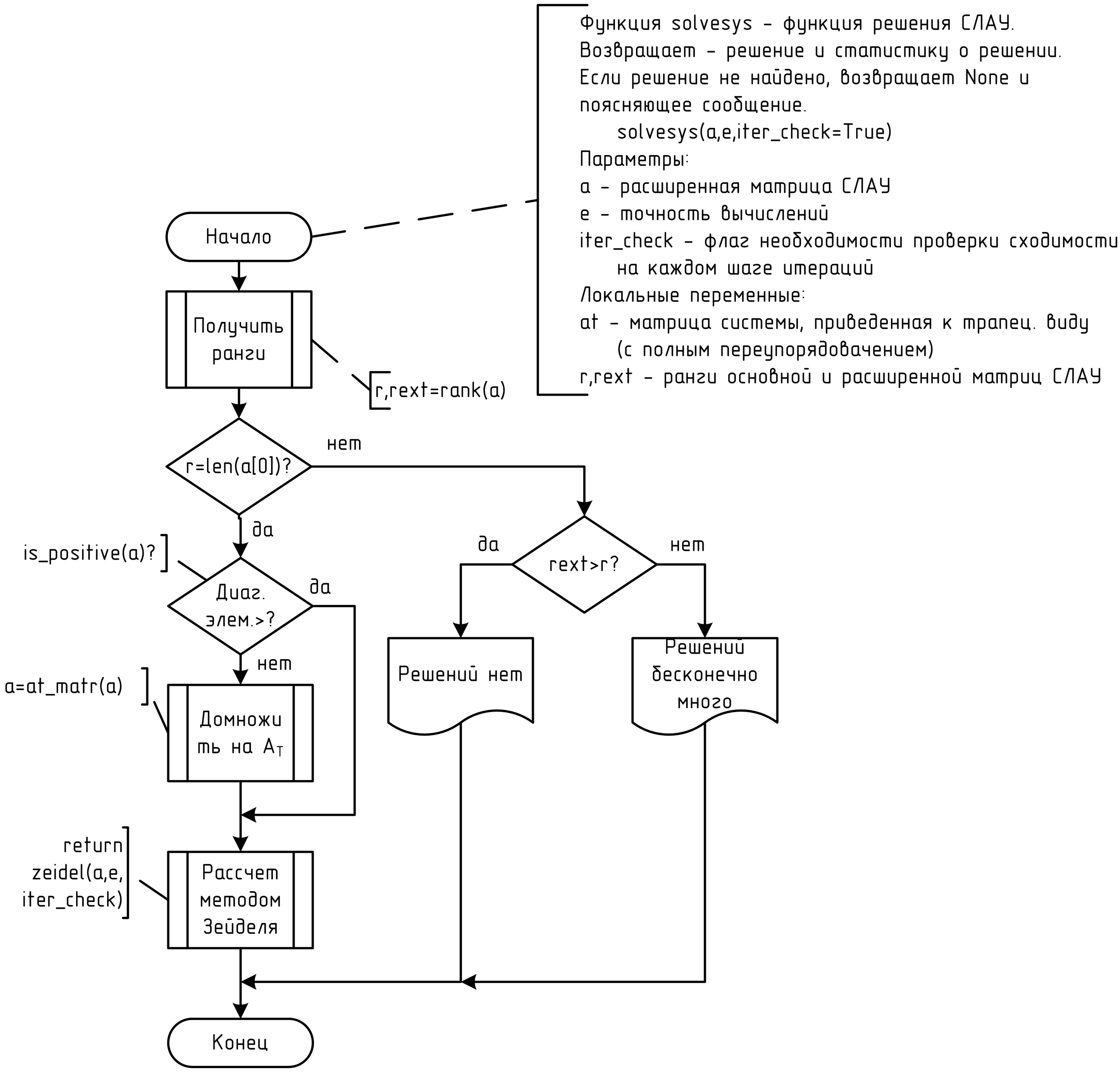

Рисунок 4 - Схема обобщенного алгоритма решения СЛАУ

На рисунке 5 представлена схема алгоритма расчета рангов основной и расширенной матрицы СЛАУ.

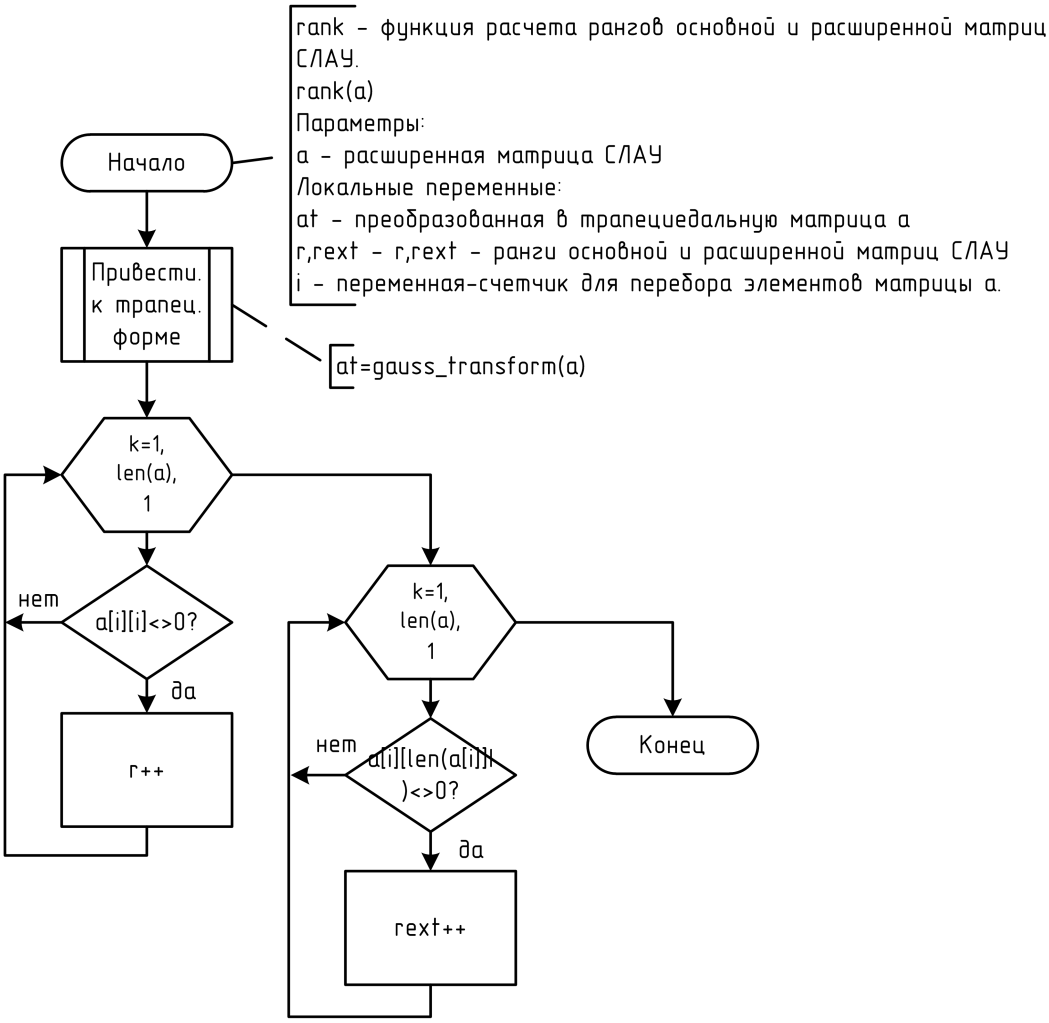

Рисунок 5 - Схема алгоритма расчета рангов

На рисунке 6 представлена схема алгоритма преобразования матрицы СЛАУ к верхнетрапециедальному виду с полным переупорядовачиванием.

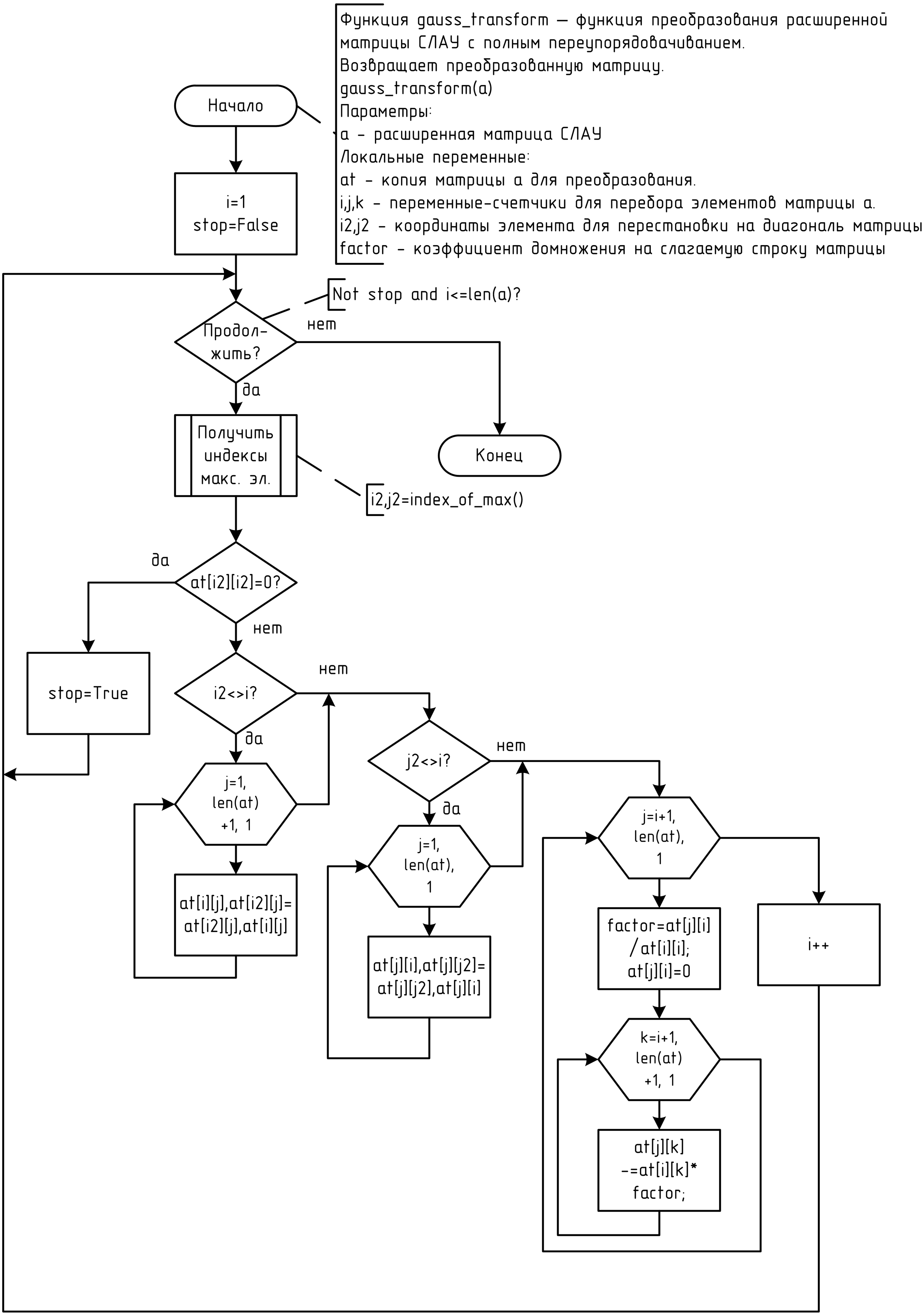

Рисунок 6 - Схема алгоритма преобразования матрицы СЛАУ

На рисунке 7 представлена схема алгоритма нахождения индексов максимального элемента в подматрице (i,i,n,n) основной матрицы СЛАУ A_{NxN}.

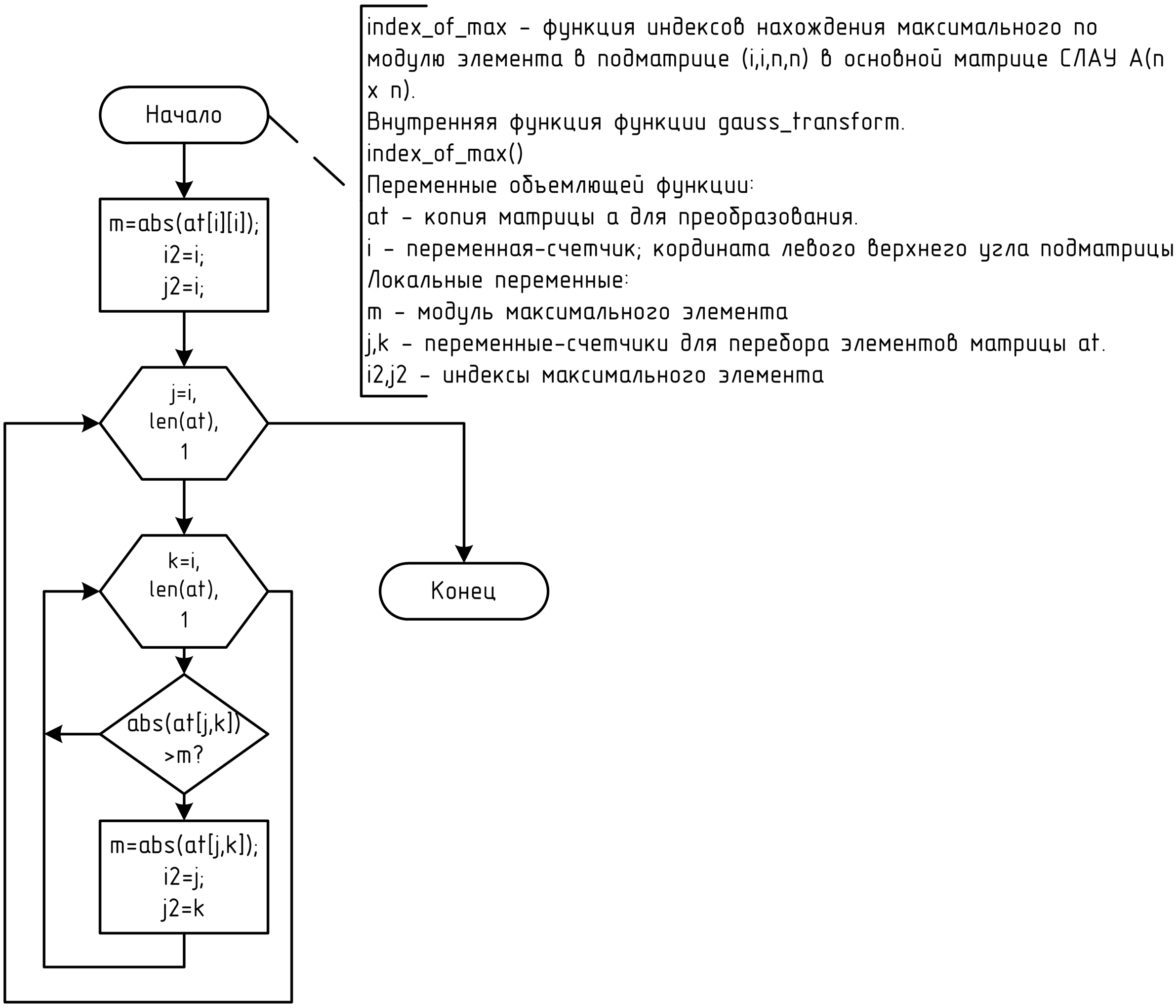

Рисунок 7 - Схема алгоритма нахождения индексов максимального элемента

На рисунке 8 представлена схема алгоритма проверки матрицы на диагональное преобладание.

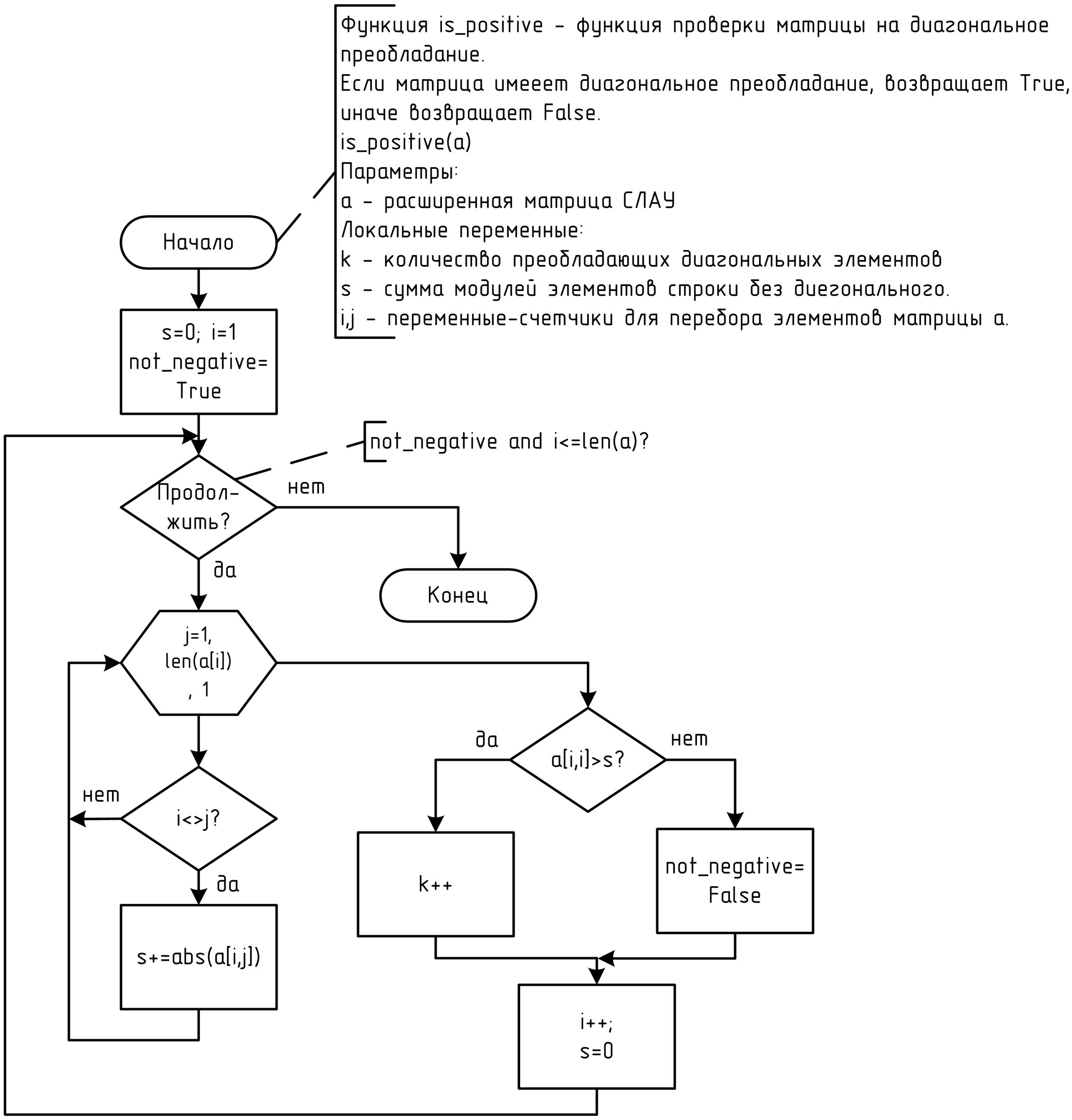

Рисунок 8 - Схема алгоритма проверки матрицы на диагональное преобладание

На рисунке 9 представлена схема алгоритма домножения расширенной матрицы СЛАУ на транспонированную основную.

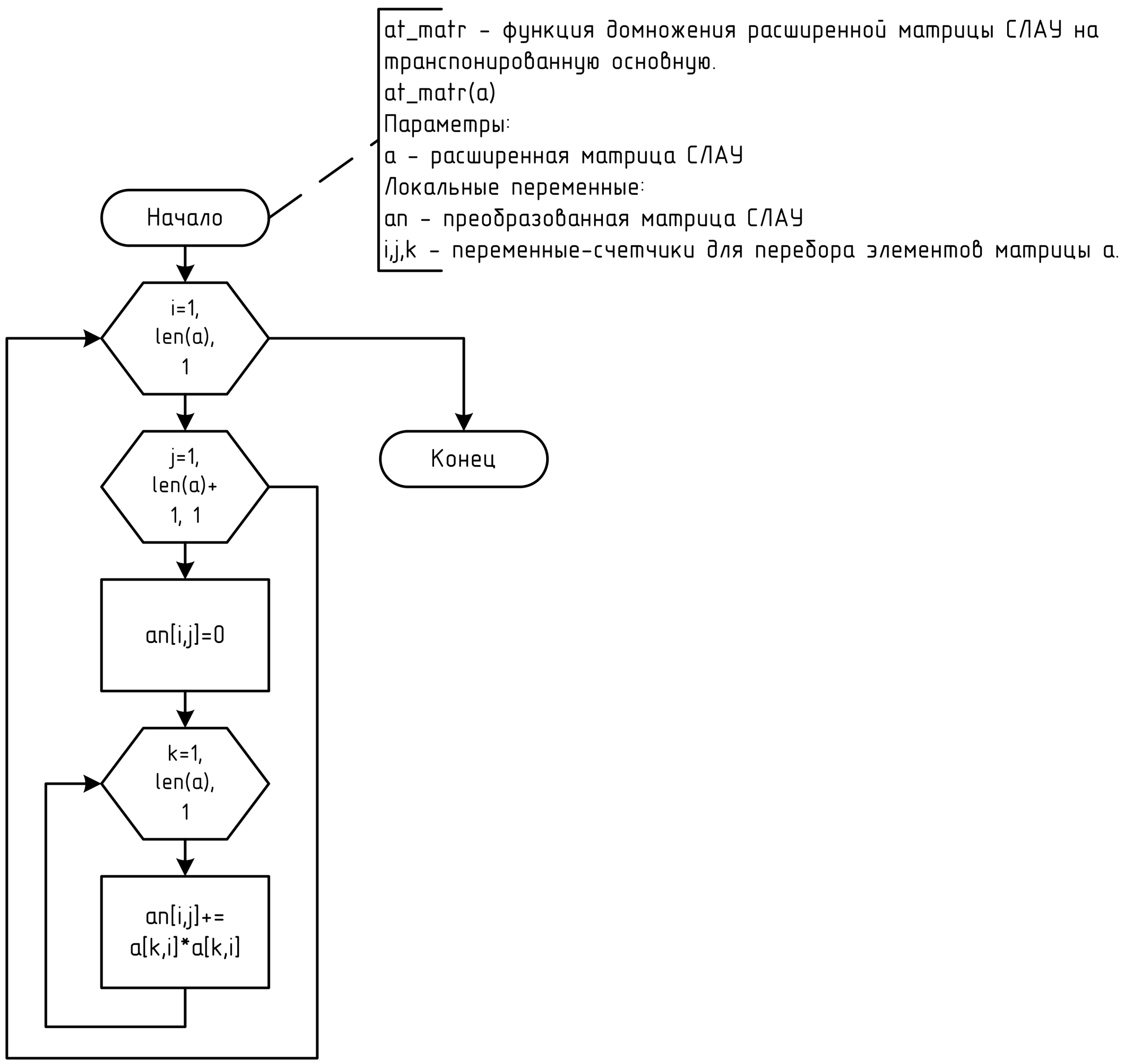

Рисунок 9 - Схема алгоритма домножения расширенной матрицы СЛАУ

На рисунке 10 представлена схема алгоритма решения СЛАУ по методу Гаусса-Зейделя.

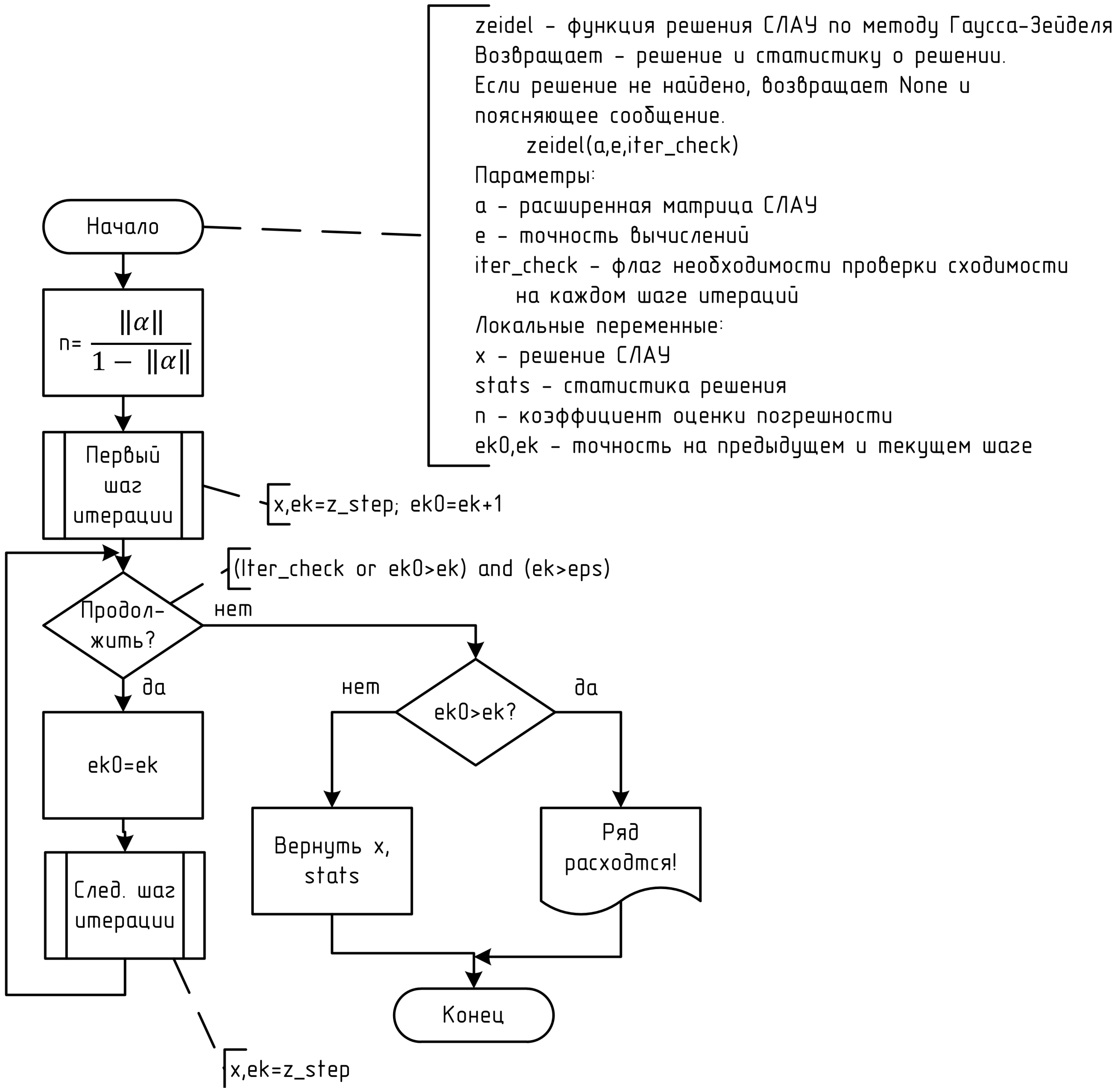

Рисунок 10 - Схема алгоритма решения СЛАУ по методу Гаусса-Зейделя

На рисунке 11 представлена схема алогритма шага итерации по методу Гаусса-Зейделя.

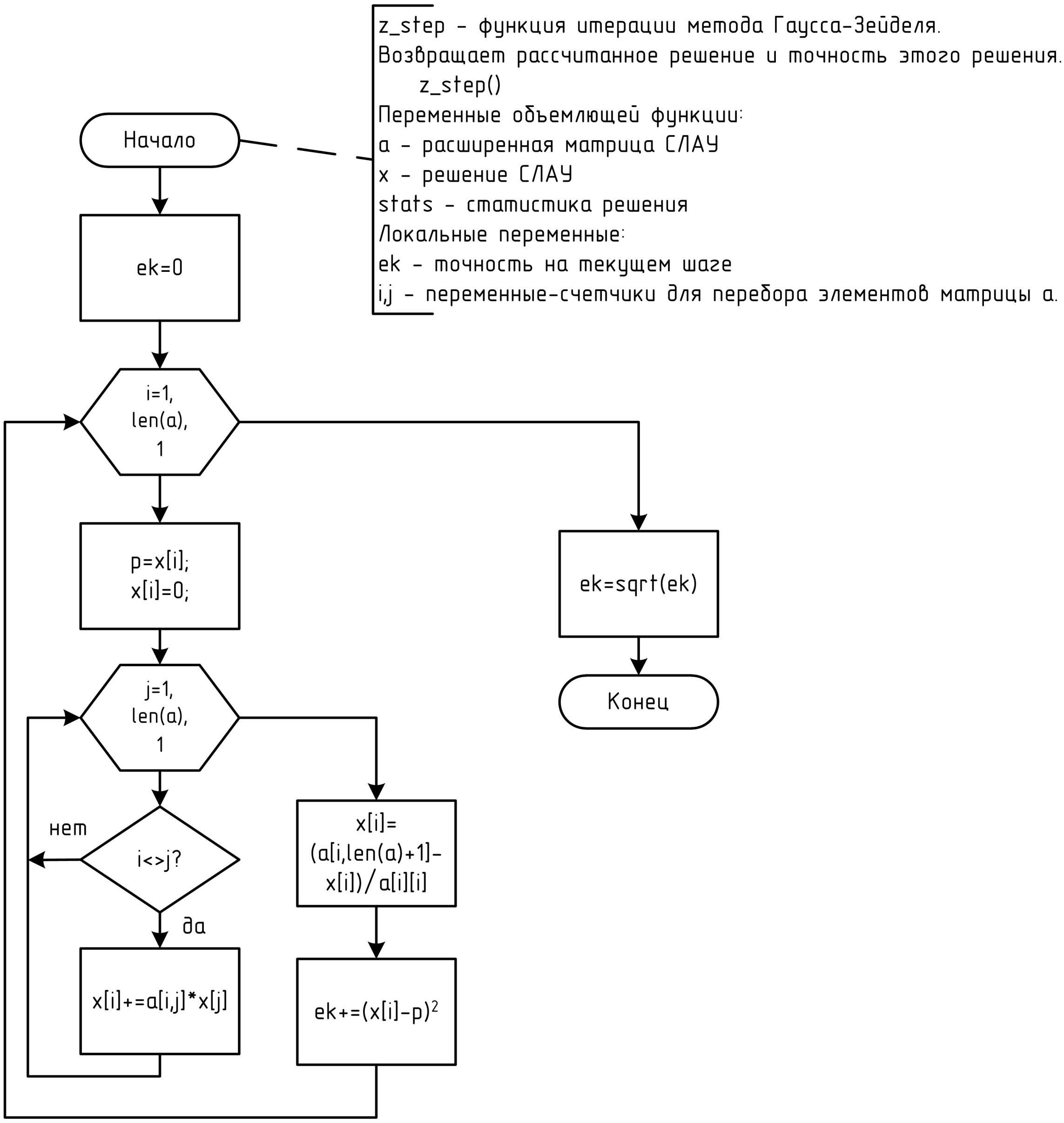

Рисунок 11 - Схема алгоритма шага итерации по методу Гаусса-Зейделя

Инструкция пользователя

Программа осуществляет построение аппроксимирующей функции тригонометрического базиса второй степени для функции $f(x) = x^4 + 3x - 1$.

Для работе программе необходимы списки аргументов и значений функции. Аргументы функции добавляются в соотвествующий список в левой части окна; значения функции рассчитываются автоматически. Уже введенные значения можно удалять из списка. После этого нужно указать точность решения системы линейных уравнений и степень аппроксимирующей функции, и нажать кнопку "Посчитать". Программа построит графики аппроксимируемой и аппроксимирующей функции и выведет вид последней на форму. Также можно посчитать значения обеих функций от требуемого аргумента.

Инструкция программиста

При разработке программы построения аппроксимирующей функции были написаны следующие процедуры и функции:

1. approx_func - рассчитывает и возвращает значение члена аппроксимирующей функции на основе тригонометрического базиса.

approx_func(n,x)

Параметры функции представлены в таблице 1 :

Таблица 1 - Параметры функции рассчета члена аппроксимирующей функции

имя	**тип**	**предназначение**
n	целое	номер члена аппроксимирующей функции
x	веществ.	аргумент функции

2. make_matrix - функция получения расширенной матрицы коэфициентов для системы уравнений коэффициентов аппроксимирующей функции на основе тригонометрического базиса.

Возвращает матрицу СЛАУ коэффициентов аппроксимирующей функции.

make_matrix(x,y,n)

Параметры функции представлены в таблице 2 :

Таблица 2 - Параметры функции получения расширенной матрицы коэфициентов

имя	тип	предназначение
x,y	список	аргументы и значения аппроксимируемой функции
n	целое	степень аппроксимирующей функции

Локальные переменные функции представлены в таблице 3 :

Таблица 3 - Локальные переменные функции получения расширенной матрицы коэфициентов

имя	тип	предназначение
a	список	расширенная матрица СЛАУ коэффициентов функции
i,j,k	целое	переменные-счетчики для работы со списками

3. make_approx_poly - расчет коэффициентов аппроксимирующей функции на основе тригонометрического базиса.

make_approx_poly(x,y,n,e)

Параметры функции представлены в таблице 4 :

Таблица 4 - Параметры функции расчета коэффициентов аппроксимирующей функции

имя	тип	предназначение
x,y	список	аргументы и значения аппроксимируемой функции
n	целое	степень аппроксимирующей функции
e	веществ.	точность решения СЛАУ

Локальные переменные функции представлены в таблице 5 :

Таблица 5 - Локальные переменные функции расчета коэффициентов аппроксимирующей функции

имя	тип	предназначение
a	список	расширенная матрица СЛАУ коэффициентов функции

4. Функция solvesys - функция решения СЛАУ. Возвращает - решение и статистику о решении. Если решение не найдено, возвращает None и поясняющее сообщение.

solvesys(a,e,iter_check=True)

Параметры функции представлены в таблице 6 :

Таблица 6 - Параметры функции решения СЛАУ

имя	**тип**	**предназначение**
a	список	расширенная матрица СЛАУ
e	веществ.	точность вычислений
iter_check	булев.	флаг необходимости проверки сходимости на каждом шаге итераций

Локальные переменные функции представлены в таблице 7 :

Таблица 7 - Локальные переменные функции решения СЛАУ

имя	**тип**	**предназначение**
at	список	матрица системы, приведенная к трапец. виду (с полным переупорядовачением)
r,rext	целое	ранги основной и расширенной матриц СЛАУ

5. Функция gauss_transform - функция преобразования расширенной матрицы СЛАУ с полным переупорядовачиванием. Возвращает преобразованную матрицу.

gauss_transform(a)

Параметры функции представлены в таблице 8 :

Таблица 8 - Параметры функции преобразования расширенной матрицы СЛАУ

имя	**тип**	**предназначение**
a	список	расширенная матрица СЛАУ

Локальные переменные функции представлены в таблице 9 :

Таблица 9 - Локальные переменные функции преобразования расширенной матрицы СЛАУ

имя	тип	предназначение
at	список	копия матрицы а для преобразования.
i,j,k	целое	переменные-счетчики для перебора элементов матрицы а.
i2,j2	целое	координаты элемента для перестановки на диагональ матрицы
factor	веществ.	коэффициент домножения на слагаемую строку матрицы

6. index_of_max - функция нахождения индексов максимального по модулю элемента в подматрице (i,i,n,n) в основной матрице СЛАУ A(n x n).

Внутренняя функция функции gauss_transform.

index_of_max()

Переменные объемлющей функции представлены в таблице 10 :

Таблица 10 - Переменные объемлющей функции функции нахождения индексов максимального по модулю элемента

имя	тип	предназначение
at	список	копия матрицы а для преобразования.
i	целое	переменная-счетчик; кордината левого верхнего угла подматрицы

Локальные переменные функции представлены в таблице 11 :

Таблица 11 - Локальные переменные функции нахождения индексов максимального по модулю элемента

имя	тип	предназначение
m	веществ.	модуль максимального элемента
j,k	целое	переменные-счетчики для перебора элементов матрицы at.
i2,j2	целое	индексы максимального элемента

7. Функция is_positive - функции проверки матрицы на диагональное преобладание. Если матрица имееет диагональное преобладание, возвращает True, иначе возвращает False.

is_positive(a)

Параметры функции представлены в таблице 12 :

Таблица 12 - Параметры функции проверки матрицы на диагональное преобладание

имя	тип	предназначение
a	список	расширенная матрица СЛАУ

Локальные переменные функции представлены в таблице 13 :

Таблица 13 - Локальные переменные функции проверки матрицы на диагональное преобладание

имя	тип	предназначение
k	целое	количество преобладающих диагональных элементов
s	веществ.	сумма модулей элементов строки без диагонального.
i,j	целое	переменные-счетчики для перебора элементов матрицы a.

8. rank - функция расчета рангов основной и расширенной матриц СЛАУ.

rank(a)

Параметры функции представлены в таблице 14 :

Таблица 14 - Параметры функции расчета рангов основной и расширенной матриц

имя	тип	предназначение
a	список	расширенная матрица СЛАУ

Локальные переменные функции представлены в таблице 15 :

Таблица 15 - Локальные переменные функции расчета рангов основной и расширенной матриц

имя	тип	предназначение
at	список	преобразованная в трапециедальную матрица a
r,rext	целое	ранги основной и расширенной матриц СЛАУ
i	целое	переменная-счетчик для перебора элементов матрицы a.

9. at_matr - функция домножения расширенной матрицы СЛАУ на транспонированную основную.

at_matr(a)

Параметры функции представлены в таблице 16 :

Таблица 16 - Параметры функции домножения расширенной матрицы на транспонированную

имя	**тип**	**предназначение**
a	список	расширенная матрица СЛАУ

Локальные переменные функции представлены в таблице 17 :

Таблица 17 - Локальные переменные функции домножения расширенной матрицы на транспонированную

имя	**тип**	**предназначение**
an	список	преобразованная матрица СЛАУ
i,j,k	целое	переменные-счетчики.

10. norm - функция расчета нормы матрицы α для методов простых итераций и Гаусса-Зейделя.

norm(a)

Параметры функции представлены в таблице 18 :

Таблица 18 - Параметры функции расчета нормы матрицы

имя	**тип**	**предназначение**
a	список	расширенная матрица СЛАУ

Локальные переменные функции представлены в таблице 19 :

Таблица 19 - Локальные переменные функции расчета нормы матрицы

имя	тип	предназначение
s	веществ.	сумма квадратов строки матрицы α
line	список	строка матрицы alpha
a_ij	веществ.	текущий элемент alpha
i,j	целое	переменные-счетчики.

11. zeidel - функция решения СЛАУ по методу Гаусса-Зейделя Возвращает - решение и статистику о решении. Если решение не найдено, возвращает None и поясняющее сообщение.

zeidel(a,e,iter_check)

Параметры функции представлены в таблице 20 :

Таблица 20 - Параметры функции решения СЛАУ по методу Гаусса-Зейделя

имя	тип	предназначение
a	список	расширенная матрица СЛАУ
e	веществ.	точность вычислений
iter_check	булев.	флаг необходимости проверки сходимости на каждом шаге итераций

Локальные переменные функции представлены в таблице 21 :

Таблица 21 - Локальные переменные функции решения СЛАУ по методу Гаусса-Зейделя

имя	тип	предназначение
x	список	решение СЛАУ
stats	словарь	статистика решения
n	веществ.	коэффициент оценки погрешности
ek0,ek	веществ.	точность на предыдущем и текущем шаге

12. z_step - функция итерации метода Гаусса-Зейделя. Внутрення функция функции zeidel. Возвращает рассчитанное решение и точность этого решения.

z_step()

Переменные объемлющей функции(zeidel) представлены в таблице 22 :

Таблица 22 - Переменные объемлющей функции функции итерации метода Гаусса-Зейделя

имя	тип	предназначение
a	список	расширенная матрица СЛАУ
x	список	решение СЛАУ
stats	словарь	статистика решения

Локальные переменные функции представлены в таблице 23 :

Таблица 23 - Локальные переменные функции итерации метода Гаусса-Зейделя

имя	тип	предназначение
ek	веществ.	точность на текущем шаге
i,j	целое	переменные-счетчики для перебора элементов матрицы a.

Текст программы

Реализиция задачи построения интерполяционного многочлена Ньютона с разделенными разностями написана на языке Python 3.2 и состоит из трех частей.

Первая часть, файл mat3.py, содержит подпрограммы построения аппроксимирующей функции на основе тригонометрического базиса. Исходный текст этого модуля приводится ниже.

```
from math import cos,sin
from mat1 import solvesys

def approx_func(n,x):
    '''
    approx_func — возвращает значение члена аппроксимирующей функции
    на основе тригонометрического базиса.
    Параметры:
    n — целое — номер члена  аппроксимирующей функции,
    x — веществ. — аргумент функции
    '''
    if n%2:
        return sin((n//2+1)*x)
    else:
        return cos((n//2)*x)

def func(x):
    return x**4+3*x—1

def make_matrix(x,y,n):
    '''
    make_matrix — функция получения расширенной матрицы коэфициентов
    для системы уравнений коэффициентов аппроксимирующей функции
    на основе тригонометрического базиса.
    Возвращает матрицу СЛАУ коэффициентов аппроксимирующей функции.
    Параметры:
    x,y — список — аргументы и значения аппроксимируемой функции,
    n — целое — степень аппроксимирующей функции
    Локальные переменные:
    a — список — расширенная матрица СЛАУ коэффициентов функции
    i,j,k — целое — переменные—счетчики для работы со списками
    '''
    a=[[0 for j in range (0,n+2)] for i in range (0,n+1)]
    for i in range (0,n+1):
        for j in range (0,n+1):
            for k in range(0,len(x)):
                a[i][j]+=approx_func(i,x[k])*approx_func(j,x[k])
    j=n+1
    for i in range (0,n+1):
        for k in range(0,len(x)):
            a[i][j]+=approx_func(i,x[k])*y[k]
    return a

def make_approx_poly(x,y,n,e):
    '''
    make_approx_poly — расчет коэффициентов аппроксимирующей функции на основе
    тригонометрического базиса.
    Параметры:
    x,y — список — аргументы и значения аппроксимируемой функции,
    n — целое — степень аппроксимирующей функции,
    e — веществ. — точность решения СЛАУ
    Локальные переменые:
    a — список — расширенная матрица СЛАУ коэффициентов функции
```

```
    '''
    a=make_matrix(x,y,n)
    return solvesys(a,e,False)[0]
```

Во второй части, представленной ниже, находятся функции решения СЛАУ.

```
#! /usr/bin/env python3
from sys import stdin,stdout
from math import sqrt
# Хелло
def solvesys(a,e,iter_check=True):
    '''
    Функция solvesys — функция решения СЛАУ.
    Возвращает — решение и статистику о решении.
    Если решение не найдено, возвращает None и
    поясняющее сообщение.

        solvesys(a,e,iter_check=True)

    Параметры:
    a — расширенная матрица СЛАУ
    e — точность вычислений
    iter_check — флаг необходимости проверки сходимости
        на каждом шаге итераций
    Локальные переменные:
    at — матрица системы, приведенная к трапец. виду
        (с полным переупорядовачением)
    r,rext — ранги основной и расширенной матриц СЛАУ
    '''
    at=a
    r,rext=rank(at)
    if r<len(a[0])—1:
        if r<rext:
            return (None,"Система не имеет решений!")
        else:
            return (None,"У системы бесконечно много решений!")
    else:
        if not is_positive(a):
            a=at_matr(a)

        return zeidel(a,e,iter_check)

def gauss_transform(a):
    '''
    Функция gauss_transform — функция
    преобразования расширенной матрицы СЛАУ
    с полным переупорядовачиванием.
    Возвращает преобразованную матрицу.

        gauss_transform(a)

    Параметры:
    a — расширенная матрица СЛАУ
    Локальные переменные:
    at — копия матрицы a для преобразования.
    i,j,k — переменные—счетчики для перебора элементов матрицы a.
    i2,j2 — координаты элемента для перестановки на диагональ
        матрицы
    factor — коэффициент домножения на слагаемую строку матрицы
    '''
    at=[[j for j in i] for i in a]
    def index_of_max():
        '''
        index_of_max — функция индексов нахождения максимального по модулю элемента
        в подматрице (i,i,n,n) в основной матрице СЛАУ A(n x n).
```

```
        Внутренняя функция функции gauss_transform.

            index_of_max()

        Переменные объемлющей функции:
        at — копия матрицы a для преобразования.
        i — переменная—счетчик; кордината левого верхнего угла подматрицы
        Локальные переменные:
        m — модуль максимального элемента
        j,k — переменные—счетчики для перебора элементов матрицы at.
        i2,j2 — индексы максимального элемента
        '''
        nonlocal at,i
        m=abs(at[i][i]);i2=i;j2=i;
        for j in range(i,len(at)):
            for k in range(i,len(at)):
                if abs(at[j][k])>m:
                    m=abs(at[j][k]);i2=j;j2=k
        return (i2,j2)

    for i in range(0,len(at)):
        i2,j2=index_of_max()

        if at[i2][j2]==0:
            return at
        if i2!=i:
            for j in range(i,len(at[i])):
                at[i][j],at[i2][j]=at[i2][j],at[i][j]
        if j2!=i:
            for j in range(0,len(at)):
                at[j][i],at[j][j2]=at[j][j2],at[j][i]
        for j in range(i+1,len(at)):
            factor=at[j][i]/at[i][i];at[j][i]=0
            for k in range(i+1,len(at[i])):
                at[j][k]-=at[i][k]*factor;
                if abs(at[j][k])<1e-10:
                    at[j][k]=0

    return at

def is_positive(a):
    '''
    Функция is_positive — функция проверки матрицы на диагональное преобладание.
    Если матрица имееет диагональное преобладание, возвращает True,
    иначе возвращает False.
        is_positive(a)
    Параметры:
    a — расширенная матрица СЛАУ
    Локальные переменные:
    k — количество преобладающих диагональных элементов
    s — сумма модулей элементов строки без диегонального.
    i,j — переменные—счетчики для перебора элементов матрицы a.
    '''
    k=0
    for i in range(0,len(a)):
        s=0
        for j in range(0,len(a)):
            if i!=j:
                s+=abs(a[i][j])
        if a[i][i]>s:
            k+=1
        elif a[i][i]<s:
            return False

    if k>0:
```

```
        return True
    else:
        return False

def rank(a):
    '''
    rank — функция расчета рангов основной и расширенной матриц СЛАУ.

        rank(a)

    Параметры:
    a — расширенная матрица СЛАУ
    Локальные переменные:
    at — преобразованная в трапециедальную матрица a
    r,rext — r,rext — ранги основной и расширенной матриц СЛАУ
    i — переменная—счетчик для перебора элементов матрицы a.
    '''
    at=gauss_transform(a)
    r=0;rext=0;
    for i in range(0,len(a)):
        if at[i][i]!=0:
            r+=1
    row_end=len(a[i])—1
    for i in range(0,len(a)):
        if at[i][row_end]!=0:
            rext+=1
    return (r,rext)

def at_matr(a):
    '''
    at_matr — функция домножения расширенной матрицы СЛАУ
    на транспонированную основную.

        at_matr(a)

    Параметры:
    a — расширенная матрица СЛАУ
    Локальные переменные:
    an — преобразованная матрица СЛАУ
    i,j,k — переменные—счетчики для перебора элементов матрицы a.
    '''
    an=[]
    for i in range(0,len(a)):
        an.append([])
        for j in range(0,len(a[0])):
            an[i].append(0)
            for k in range(0,len(a)):

                an[i][j]+=(a[k][i]*a[k][j] )# an[i,j]:=an[
    return an

def norm(a):
    '''
    norm — функция расчета нормы матрицы alpha для методов
    простых итераций и Гаусса—Зейделя.

        norm(a)

    Параметры:
    a — расширенная матрица СЛАУ
    Локальные переменные:
    s — сумма квадратов строки матрицы alpha
    line — строка матрицы alpha
    a_ij — текущий элемент alpha
    i,j — переменные—счетчики.
```

```
    '''
    s=0;i=0
    for line in a:
        j=0
        for a_ij in line:
            if i!=j:
                s+=a_ij*a_ij
            j+=1
        s/=a[i][i]*a[i][i]
        i+=1
    return sqrt(s)

def zeidel(a,e,iter_check):
    '''
    zeidel - функция решения СЛАУ по методу Гаусса-Зейделя
    Возвращает - решение и статистику о решении.
    Если решение не найдено, возвращает None и
    поясняющее сообщение.

        zeidel(a,e,iter_check)

    Параметры:
    a - расширенная матрица СЛАУ
    e - точность вычислений
    iter_check - флаг необходимости проверки сходимости
        на каждом шаге итераций
    Локальные переменные:
    x - решение СЛАУ
    stats - статистика решения
    n - коэффициент оценки погрешности
    ek0,ek - точность на предыдущем и текущем шаге
    '''
    stats={'сложений':0,
           'вычитаний':0,
           'умножений':0,
           'делений':0,
           'итераций':0}
    def z_step():
        '''
        z_step - функция итерации метода Гаусса-Зейделя.
        Возвращает рассчитанное решение и точность этого решения.

            z_step()

        Переменные объемлющей функции:
        a - расширенная матрица СЛАУ
        x - решение СЛАУ
        stats - статистика решения
        Локальные переменные:
        ek - точность на текущем шаге
        i,j - переменные-счетчики для перебора элементов матрицы a.
        '''
        nonlocal a,x,stats
        ek=0;
        for i in range(0,len(a)):
            p=x[i];x[i]=0;
            for j in range(0,len(a)):
                if i!=j:
                    x[i]+=a[i][j]*x[j]
                    stats['умножений']+=1;stats['сложений']+=1
            x[i]=(a[i][len(a[i])-1]-x[i])/a[i][i]
            stats['делений']+=1;stats['вычитаний']+=1
            ek+=abs(x[i]-p)

            stats['итераций']+=1
```

```
        return  x,ek
    if norm(a)!=1:
        n=abs(norm(a)/(1-norm(a)))
    else:
        n=1
    x=[0 for i in range(0,len(a))]
    x,ek=z_step();ek0=ek+1
    while (not iter_check or (ek0>ek))and(n*ek>=e):
        ek0=ek;
        x,ek=z_step()
    if ek>ek0:
        return (None,
            'Сходимость нарушается на шаге {}!'.format(stats['итераций']))
    return x,stats
```

Третья часть - файл inter.py - является графическим интерфейсом для вычислительного ядра первых двух модулей. Далее представлен текст этого модуля.

```
#!/usr/bin/env python3
from tkinter import *
from math import floor,cos,sin,sqrt
from mat3 import approx_func,func,make_approx_poly


class approx_poly:
    def __init__(self,coef):
        self.coef=coef
    def __call__(self,x):
        res=0;
        for i in range(0,len(self.coef)):
            res+=approx_func(i,x)*self.coef[i]
        return res
class plotlet:
    def __init__(self,f,color,width):
        self.f=f
        self.color=color
        self.width=width
class plotter:
    def __init__(self,canv=None,dx=1,zoom=1,x0=0,y0=0,delta=1,plotlets=None):
        self.canv=canv;
        self.dx=dx
        self.x0=x0
        self.y0=y0
        self.delta=delta
        self.zoom=zoom
        if plotlets:
            self.plotlets=plotlets
        else:
            self.plotlets=[]
        if canv:
            self.canv.bind('<Button-1>',self.__lock_mouse)
            self.canv.bind('<B1-Motion>',self.__move_handler)
            if plotlets:
                self.plot()
    def draw_orts(self):
        self.canv.create_line(3,self.canv["height"],3,1,width=2,arrow=LAST)
        self.canv.create_line(0,int(self.canv["height"]),
                              self.canv["width"],
                              int(self.canv["height"]),
                              width=2,arrow=LAST)
        min_=floor(-self.x0/self.zoom/self.delta)+1
        max_=floor((int(self.canv["width"])-self.x0)/self.zoom/self.delta)
        while min_<=max_:
            self.canv.create_line(self.x0+min_*self.delta*self.zoom,
```

```
                                self.canv["height"],
                                self.x0+min_*self.delta*self.zoom,
                                int(self.canv["height"])-5,width=2)
            canv.create_text(self.x0+min_*self.zoom*self.delta,
                             int(self.canv["height"])-8,
                             anchor='s',text=str(min_*self.delta),
                             font="Verdana 7",justify=CENTER,fill='black')
            min_+=1
        min_=floor(-self.y0/self.zoom/self.delta)
        max_=floor((int(self.canv["height"])-self.y0)/self.zoom/self.delta)+1
        while max_>=min_:
            self.canv.create_line(3,
                                int(self.canv["height"])-self.y0-
                                      max_*self.delta*self.zoom,
                                8,
                                int(self.canv["height"])-self.y0-
                                      max_*self.zoom*self.delta,
                                width=2)
            canv.create_text(11,int(self.canv["height"])-self.y0-
                                      max_*self.delta*self.zoom,
                             anchor='w',text=str(max_*self.delta),
                             font="Verdana 7",justify=CENTER,fill='black')
            max_-=1
    def clear(self):
        self.canv.delete('all')
        self.plot()
        self.draw_orts()
    def plot(self):
        self.canv.delete('all')
        maxx,maxy=int(self.canv["width"]),int(self.canv["height"])
        for plt in self.plotlets:
            if plt.f:
                x=-self.x0/self.zoom;y=plt.f(x)
                while x<maxx/self.zoom:
                    x2=x+self.dx;y2=plt.f(x2)
                    canv.create_line(self.x0+self.zoom*x, #x begin
                                     maxy-self.zoom*y-self.y0, #y begin
                                     self.x0+self.zoom*x2, #x end
                                     maxy-self.zoom*y2-self.y0, #y end
                                     width=plt.width,fill=plt.color)
                    x,y=x2,y2
        self.draw_orts()
    def __lock_mouse(self,event):
        self._lock_x,self._lock_y=event.x-self.x0,
                             int(self.canv["height"])-event.y-self.y0
    def __move_handler(self,event):
        self.x0=event.x-self._lock_x
        self.y0=int(self.canv["height"])-event.y-self._lock_y

        self.plot()

def add_click(event):
    try:
        x=float(addl.get())
    except:
        pass
    else:
        lb.insert(END,x)
        y=addly.get()
        try:
            if y=='':
                y=func(float(addl.get()))
            else:
                y=float(addly.get())
        except:
```

```
            pass
        else:
            lbY.insert(END,y)

def del_click(event):
    x=lb.curselection()
    while x:
        lb.delete(x[0])
        lbY.delete(x[0])
        x=lb.curselection()

    x=lbY.curselection()
    while x:
        lb.delete(x[0])
        lbY.delete(x[0])
        x=lbY.curselection()
def print_approx_func(n):
    if n%2:
        return 'sin({}x)'.format(round(n//2+1,4))
    else:
        return 'cos({}x)'.format(round(n//2,4))
def print_poly(poly):
    s=''
    if len(poly)>1:
        for i in range(len(poly)−1,0,−1):
            if poly[i]>0:
                s+='+{}{}'.format(round(poly[i],4),print_approx_func(i))
            elif poly[i]<0:
                s+='{}{}'.format(round(poly[i],4),print_approx_func(i))
    if poly[0]>0:
        s+='+{}'.format(round(poly[i],4))
    elif poly[0]<0:
        s+='{}'.format(round(poly[i],4))
    return s

def run_click(event):
    global poly
    x=[float(i) for i in lb.get(0, END)]
    y=[float(i) for i in lbY.get(0,END)]
    try:
        e=float(enEps.get())
        n=int(enN.get())
        if n<0:
            raise ValueError
    except:
        pass
    else:
        coeff=make_approx_poly(x,y,n,e)
        lbcoef.config(text=print_poly(coeff))
        poly=approx_poly(coeff)
        pl.plotlets=[pl.plotlets[0]]
        pl.plotlets.append(plotlet(poly,"red",3))
        pl.plot()

def zoom_move(event):
    pl.zoom=sclzoom.get()
    pl.plot()


def det_move(event):
    try:
        pl.dx=1/scldet.get()
    except:
        pass
    else:
```

```
        pl.plot()

def delta_enter(event):
    try:
        pl.delta=float(ec.get())
    except:
        pass
    else:
        pl.plot()

def check(event):
    try:
        x=float(e1.get())
    except:
        exit()
    f1.set(str(func(x)));
    try:
        f2.set(str(poly(x)))
    except:
        pass

def yview(*args):
    lb.yview(*args)
    lbY.yview(*args)

def scr_set(*args):
    scr.set(*args)
    lb.yview('moveto',args[0])
    lbY.yview('moveto',args[0])
root=Tk()
root.title("Вычислительный практикум №3")


frm1=Frame(root,height=350)
delb=Button(frm1,text="Удалить")
delb.bind("<Button-1>",del_click)
lb = Listbox(frm1,selectmode=EXTENDED,width=5)
lbY= Listbox(frm1,selectmode=EXTENDED,width=5)
scr = Scrollbar(frm1,command=yview)
lb.configure(yscrollcommand=scr_set)
lbY.configure(yscrollcommand=scr_set)

addl=Entry(frm1,width=5)
addly=Entry(frm1,width=5)

lbEps=Label(frm1,text="Точность\nрешения\nСЛАУ:")
enEps=Entry(frm1,width=8)
lbN=Label(frm1,text="Степень\nапрокс.\nфункции:")
enN=Entry(frm1,width=8)

addb=Button(frm1,text='+')
addb.bind("<Button-1>",add_click)
addl.bind("<Return>",add_click)
addly.bind("<Return>",add_click)
runb=Button(frm1,text="Посчитать")
runb.bind('<Button-1>',run_click)

frm1.grid(row=0,column=0)

delb.grid(row=0,column=0,columnspan=2)
lb.grid(row=1,column=0)
lbY.grid(row=1,column=1)
scr.grid(row=1,column=2,sticky=N+S+W)
addl.grid(row=3,column=0)
addly.grid(row=3,column=1)
```

```
addb.grid(row=3,column=2)
lbEps.grid(row=4,column=0,columnspan=2)
enEps.grid(row=5,column=0,columnspan=2)
lbN.grid(row=6,column=0,columnspan=2)
enN.grid(row=7,column=0,columnspan=2)
runb.grid(row=8,column=0,columnspan=2)

canv=Canvas(root,width=500,height=500,bg="white")
canv.grid(row=0,column=1)

frm2=Frame(root,height=350)
lz=Label(frm2,text="Масштаб:")
sclzoom=Scale(frm2,from_=1,to=500,orient=HORIZONTAL)
sclzoom.set(30)
ld=Label(frm2,text="Детализация:")
scldet=Scale(frm2,from_=1,to=100,orient=HORIZONTAL)
scldet.set(10)
lbcoef=Label(frm2)
lc=Label(frm2,text="Цена деления:")
ec=Entry(frm2)
l1=Label(frm2,text="Проверка точки:")
e1=Entry(frm2,width=20)
f1,f2=StringVar(),StringVar()
l2=Label(frm2,text="Значение аппроксимируемой функции:")
e2=Entry(frm2,width=20,textvariable=f1)
l3=Label(frm2,text="Значение аппроксимирующей функции:")
e3=Entry(frm2,width=20,textvariable=f2)
b2=Button(frm2,text="Посчитать")

sclzoom.bind("<B1-Motion>",zoom_move)
scldet.bind("<B1-Motion>",det_move)
ec.bind("<Return>",delta_enter)
b2.bind("<Button-1>",check)
lbcoef.grid(row=0,column=2,rowspan=2)
lz.grid(row=0,column=0)
sclzoom.grid(row=0,column=1)
ld.grid(row=1,column=0)
scldet.grid(row=1,column=1)
lc.grid(row=2,column=0)
ec.grid(row=2,column=1)
l1.grid(row=4,column=0)
e1.grid(row=5,column=0)
b2.grid(row=3,column=1,rowspan=4)
l2.grid(row=3,column=2)
e2.grid(row=4,column=2)
l3.grid(row=5,column=2)
e3.grid(row=6,column=2)
frm2.grid(row=1,column=0,columnspan=2)

mainfunc=plotlet(func,"blue",3)
pl=plotter(canv,0.1,30,plotlets=[mainfunc])
pl.plot()
root.mainloop()
```

Тестовый пример

Ниже на рисунке 12 представлен пример работы программы при построении аппроксимирующей функции тригонометрического базиса второй степени для функции $f(x) = x^4 + 3x - 1$.

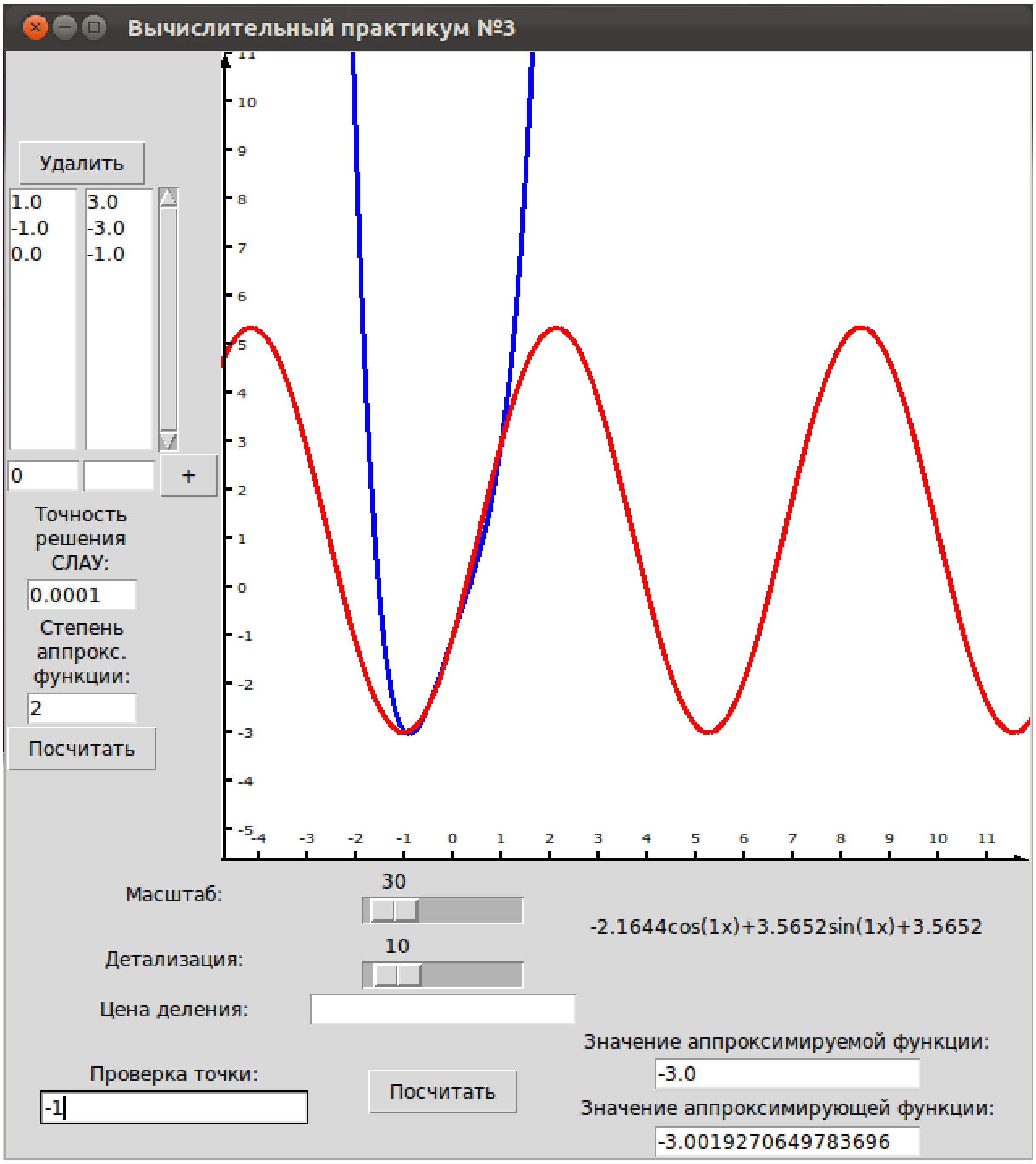

Рисунок 12 - Пример работы программы

Вывод

В этой лабораторной работе я изучил методы среднеквадратического приближения функций. Аппроксимация применяется, когда число узлов для построения превышает число известных параметров. Метод среднеквадратического приближения дает хорошие результаты - функции, построенные с помощью такого метода, достаточно приближены к исходной. Аппроксимация часто используется во многих областях науки и техники.