УДК 519.612.4

МЕТОД ЗЕЙДЕЛЯ ДЛЯ РЕШЕНИЯ СЛАУ
SEIDEL'S METHOD FOR SOLVING SLAE

К.А. Марчук
Научный руководитель – А.А. Волков, старший преподаватель,
Е.М. Гецман, старший преподаватель
Белорусский национальный технический университет,
г. Минск, Республика Беларусь
volkau@bntu.by, hetsman@bntu.by
K. Marchuk
Supervisor – A. Volkau, Senior Lecturer
E. Hetsman, Senior Lecturer
Belarusian national technical university, Minsk, Belarus

Аннотация: *Решение систем линейных алгебраических уравнений (СЛАУ) методом Зейделя, и реализация алгоритма на языке программирования Python.*
Abstract: *Solution of systems of linear algebraic equations (SLAE) by the Seidel method and implementation of the algorithm in the Python programming language.*
Ключевые слова: *алгоритм, моделирование, метод Зейделя, решение СЛАУ.*
Keywords: *algorithm, modeling, Seidel's method, solution of SLAE.*

Введение

Моделирование – это один из способов исследование объекта, при котором объект исследования замещается моделью, которая находится в некотором соответствии с исходным объектом, способная замещать его в определенных отношениях и дающая при его исследовании информацию об самом объекте. Одним из востребованных видов моделирования является математическое моделирование, которое позволяет абстрагироваться от физической природы объектов, с помощью математических зависимостей, систем линейных и нелинейных уравнений – математической модели.

Математическая модель представляет собой математическое описание поведения реального объекта – элемента электроэнергетической системы

Основная часть

Часто инженер сталкивается с необходимостью решать громоздкие системы уравнений, например, полученные по законам Кирхгофа (СЛАУ). Если необходимо вычислять с заданной точностью, то лучше применить метод Зейделя.

$$\begin{aligned}
&a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\
&a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\
&a_{31}x_1 + a_{32}x_2 + \ldots + a_{3n}x_n = b_3 \\
&\vdots \\
&a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n
\end{aligned} \tag{1}$$

Метод Зейделя является итерационным методом решения, то есть находится не точное решение, а некоторое приближение к нему, которое задается пользователем. В свою очередь данный метод является лишь модификацией метода простой итерации, заключающейся в том, что при вычислении очередного приближения $x^{(k+1)}$, его уже полученные компоненты $x_1^{(k+1)},\ldots x_{i-1}^{(k+1)}$ сразу же используются для вычисления $x_i^{(k+1)}$.В координатной форме записи метод Зейделя имеет вид:

$$\begin{aligned} &x_1^{(k+1)} = b_1 + a_{12}x^k{}_2 + \ldots + a_{1n}x^k{}_n \\ &x_2^{(k+1)} = b_2 + a_{21}x_1^{(k+1)} + \ldots + a_{2n}x^k{}_n \\ &\vdots \\ &x_n^{(k+1)} = b_n + a_{n1}x_1^{(k+1)} + \ldots + a_{nn}x^k{}_n \quad (k = 0,1,2,\ldots). \end{aligned} \tag{2}$$

где $x^{(k)}$ – некоторое начальное приближение к решению (Рисунок 1).

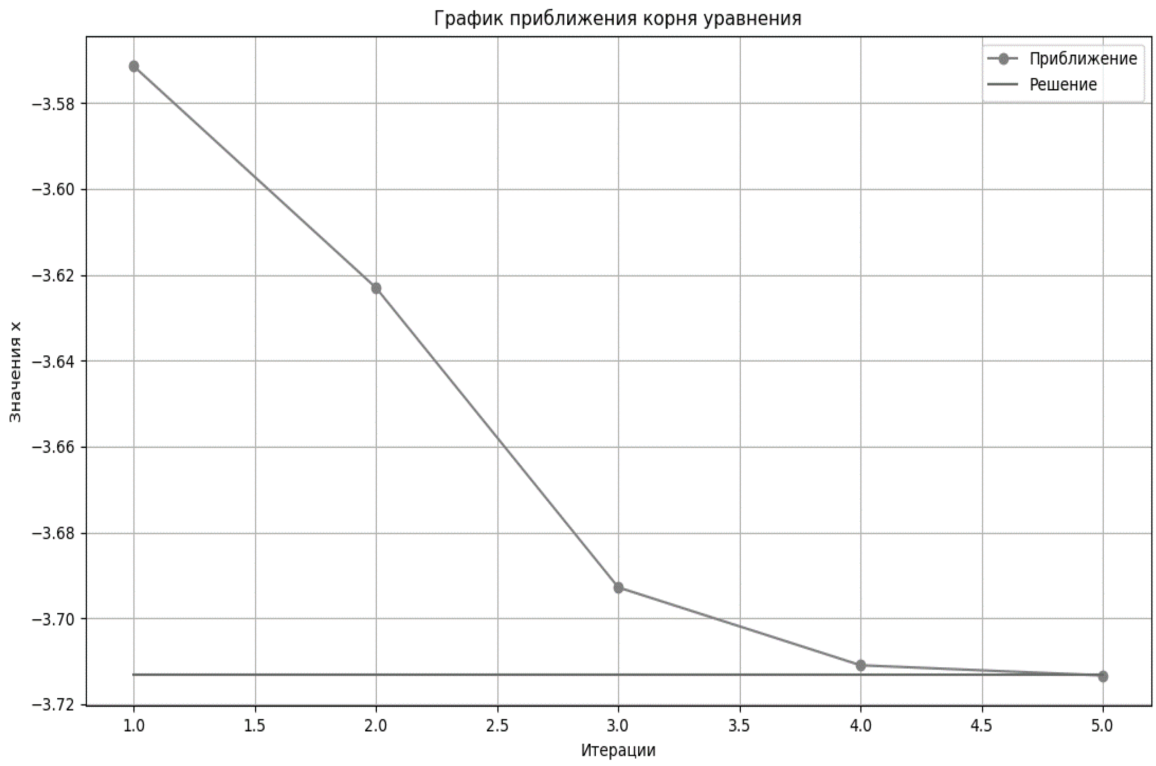

Рисунок 1 – График приближения корня уравнения

Условие окончания итерационного процесса по методу Зейделя:

$$\left\| x^{(k+1)} - x^{(k)} \right\| \le \varepsilon. \tag{3}$$

где ε – точность, которая задается пользователем.

Следует обратить внимание на особенность метода Зейделя, которая состоит в том, что полученное в первом уравнении значение x_1 сразу же используется во втором уравнении, а значения x_1, x_2 – в третьем уравнении и так далее. То есть, все найденные значения x_1 подставляются в уравнения для нахождения x_{i+1} [1]. Пример реализации метода Зейделя с использованием Python:

```
from math import * #import бибилотеки math
import numpy as np #импорт библиотеки numpy
import time as t #импорт библиотеки time
import matplotlib.pyplot as plt # импорт библиотеки matplotlib.pyplot
z=t.time()
c=[]
A = np.array([[35, 10.5, 4],# матрица коэффициентов
           [12, 19.5, 4],
          [33, 10.5, 49]])
b = np.array([[-125],#столбец свободных коэффициентов
           [-32],
           [-161.5]])
m = len(A)# длинна матрицы А
x = [0. for i in range(m)] # начальное приближение
count= 0
pogr = 0.
while True:
    x_new = np.copy(x)
    for i in range(m):
        s1 = sum(A[i][j] * x_new[j] for j in range(i))
        s2 = sum(A[i][j] * x[j] for j in range(i + 1, m))
        x_new[i] = (b[i] - s1 - s2) / A[i][i]#получаем начальные значения для x1,x2,x3...xn
        c.append(list((b[i] - s1 - s2) / A[i][i]))# создаем массив начальных приближений
    pogr = sum(abs(x_new[i] - x[i])  for i in range(m))#вычисляем погрешность
    if  pogr < 1e-2: #проверка условия
        break
    count+= 1
    x=x_new
print('Количество итераций :', count+1)
print('Решение системы уравнений :', x)
print('Погрешность :', pogr)
print('Производительность кода',t.time()-z, 'секунд')
       Количество итераций : 5
       Решение системы уравнений : [-3.710972    0.84559159 -0.97789053]
       Погрешность : 0.005052519440214342 5
       Производительность кода 0.014000892639160156 секунд
```

Рисунок 2 – Пример реализации метода Зейделя на Python

Осуществим проверку программы. Решим ту же систему СЛАУ методом обратной матрицы (Рисунок 3). Метод обратной матрицы можно отнести к точным методам, использующийся в том случае, если число неизвестных совпадает с числом уравнений [2]. Система СЛАУ в матричной форме примет вид:

$$A \cdot X = B \tag{4}$$

$$X = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}, \quad A = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ & & & \\ a_{m,1} & a_{m,1} & \cdots & a_{m,n} \end{pmatrix}, \quad B = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}, \tag{5}$$

где A – матрица, составленная из коэффициентов $a_{i,j}$ при неизвестных x;

B – матрица свободных членов;

X – матрица неизвестных.

Из полученного матричного уравнения необходимо выразить X. Для этого умножим обе части матричного уравнения на A^{-1}, получим:

$$X = A^{-1} \cdot B, \tag{6}$$

```
import numpy as np #импорт библиотеки numpy

A = np.array([[35, 10.5, 4],# матрица коэффициентов
             [12, 19.5, 4],
            [33, 10.5, 49]])
b = np.array([[-125],#столбец свободных коэффициентов
             [-32],
             [-161.5]])
if np.linalg.det(A)!=0: #проверка на вырожденность матрицы A
    r = np.linalg.solve(A,b)# функция решающая СЛАУ
    print('Решение системы уравнений :', r[0],r[1],r[2])
else:
    print("Матрица системы вырожденная")

Решение системы уравнений : [-3.71313385] [0.8442135] [-0.97613928]
```

Рисунок 3 – Пример решения СЛАУ с помощью обратной матрицы на Python

Значения немного отличаются, но это связано с точность решения, которая в нашем случае равна 0,01.

Заключение

Основными преимуществами метода Зейделя являются:

– метод Зейделя являются абсолютно сходящимся, так нет необходимости вводить условия сходимости;

– удобным при программировании, так как позволяет накапливать сумму произведений без записи промежуточных результатов.

К недостатку можно отнести необходимость контролировать ведущие элементы, чтобы какой-нибудь из них не стал равным нулю, что в свою очередь приведет к ошибке, так как деление на ноль неосуществимо.

Литература

1. В.И. Копнина Численные методы линейной и нелинейной алгебры / Копнина В.И., Вельмисова А.И. – Саратов: СГУ имени Н.Г. Чернышевского, 2016. – 48 с.

2. В.В. Конев Линейная алгебра/ В.В. Конев – Томск: ТПУ, 2008. – 65 с.