Метод Гаусса — Зейделя решения системы линейных уравнений

Метод Гаусса — Зейделя (метод Зейделя, процесс Либмана, метод последовательных замещений) — является классическим итерационным методом решения системы линейных уравнений

Назван в честь Зейделя и Гаусса.

Содержание

Постановка задачи

Возьмём систему: $A\vec{x} = \vec{b}$, где $A = \begin{pmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \dots & a_{nn} \end{pmatrix}, \quad \vec{b} = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}$

Или $\begin{cases} a_{11}x_1 + \ldots + a_{1n}x_n = b_1 \\ \\ a_{n1}x_1 + \ldots + a_{nn}x_n = b_n \end{cases}$

И покажем, как её можно решить с использованием метода Гаусса-Зейделя.

Метод

Чтобы пояснить суть метода, перепишем задачу в виде

$$\begin{cases} a_{11}x_1 = -a_{12}x_2 - a_{13}x_3 - \ldots - a_{1n}x_n + b_1 \\ a_{21}x_1 + a_{22}x_2 = -a_{23}x_3 - \ldots - a_{2n}x_n + b_2 \\ \ldots \\ a_{(n-1)1}x_1 + a_{(n-1)2}x_2 + \ldots + a_{(n-1)(n-1)}x_{n-1} = -a_{(n-1)n}x_n + b_{n-1} \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{n(n-1)}x_{n-1} + a_{nn}x_n = b_n \end{cases}$$

Здесь в j-м уравнении мы перенесли в правую часть все члены, содержащие x_i, для $i > j$. Эта запись может быть представлена как

$$(\mathrm{L}+\mathrm{D})\vec{x} = -\mathrm{U}\,\vec{x} + \vec{b},$$

где в принятых обозначениях D означает матрицу, у которой на главной диагонали стоят соответствующие элементы матрицы A, а все остальные нули; тогда как матрицы U и L содержат верхнюю и нижнюю треугольные части A, на главной диагонали которых нули.

Итерационный процесс в методе Гаусса — Зейделя строится по формуле

$$(\mathrm{L}+\mathrm{D})\vec{x}^{(k+1)} = -\mathrm{U}\vec{x}^{(k)} + \vec{b}, \quad k = 0, 1, 2, \ldots$$

после выбора соответствующего начального приближения $\vec{x}^{(0)}$.

Метод Гаусса — Зейделя можно рассматривать как модификацию метода Якоби. Основная идея модификации состоит в том, что новые значения $\vec{x}^{(i)}$ используются здесь сразу же по мере получения, в то время как в методе Якоби они не используются до следующей итерации:

$$\begin{cases} x_1{}^{(k+1)} &= & c_{12}x_2^{(k)} &+ & c_{13}x_3^{(k)} &+ \ldots + & c_{1n}x_n{}^{(k)} &+ & d_1 \\ x_2{}^{(k+1)} &= & c_{21}x_1^{(k+1)} &+ & c_{23}x_3^{(k)} &+ \ldots + & c_{2n}x_n{}^{(k)} &+ & d_2 \\ \quad\ldots & & & & & & & & \\ x_n{}^{(k+1)} &= & c_{n1}x_1^{(k+1)} &+ & c_{n2}x_2^{(k+1)} &+ \ldots + & c_{n(n-1)}x_{n-1}{}^{(k+1)} &+ & d_n \end{cases},$$

где

$$c_{ij} = \begin{cases} -\frac{a_{ij}}{a_{ii}}, & j \neq i, \\ 0, & j = i. \end{cases} \quad d_i = \frac{b_i}{a_{ii}}, \quad i = 1, \ldots, n.$$

Таким образом, *i*-я компонента $(k+1)$-го приближения вычисляется по формуле

$$x_i^{(k+1)} = \sum_{j=1}^{i-1} c_{ij}x_j^{(k+1)} + \sum_{j=i}^{n} c_{ij}x_j^{(k)} + d_i, \quad i = 1, \ldots, n.$$

Например, при $n = 3$:

$x_1^{(k+1)} = \sum\limits_{j=1}^{1-1} c_{1j}x_j^{(k+1)} + \sum\limits_{j=1}^{3} c_{1j}x_j^{(k)} + d_1$, то есть $x_1^{(k+1)} = c_{11}x_1^{(k)} + c_{12}x_2^{(k)} + c_{13}x_3^{(k)} + d_1$,

$x_2^{(k+1)} = \sum\limits_{j=1}^{2-1} c_{2j}x_j^{(k+1)} + \sum\limits_{j=2}^{3} c_{2j}x_j^{(k)} + d_2$, то есть

$x_2^{(k+1)} = c_{21}x_1^{(k+1)} + c_{22}x_2^{(k)} + c_{23}x_3^{(k)} + d_2$,

$x_3^{(k+1)} = \sum\limits_{j=1}^{3-1} c_{3j}x_j^{(k+1)} + \sum\limits_{j=3}^{3} c_{3j}x_j^{(k)} + d_3$, то есть

$x_3^{(k+1)} = c_{31}x_1^{(k+1)} + c_{32}x_2^{(k+1)} + c_{33}x_3^{(k)} + d_3$.

Условие сходимости

Приведём достаточное условие сходимости метода.

Теорема.
Пусть $\|\mathbf{A_2}\| < 1$, где $\mathbf{A_2} = -(\mathbf{L}+\mathbf{D})^{-1}\,\mathbf{U}$, $(\mathbf{L}+\mathbf{D})^{-1}$ – матрица, обратная к $(\mathbf{L}+\mathbf{D})$.
Тогда при любом выборе начального приближения $\vec{x}^{(0)}$:

1. метод Гаусса-Зейделя сходится;
2. скорость сходимости метода равна скорости сходимости геометрической прогрессии со знаменателем $q = \|\mathbf{A_2}\|$;
3. верна оценка погрешности: $\|\vec{x}^{(k)} - \vec{x}\| = q^k \, \|\vec{x}^{(0)} - \vec{x}\|$.

Условие окончания

Условие окончания итерационного процесса Зейделя при достижении точности ε в упрощённой форме имеет вид:

$$\|x^{(k+1)} - x^{(k)}\| \leq \varepsilon$$

Более точное условие окончания итерационного процесса имеет вид

$$\|Ax^{(k)} - b\| \leq \varepsilon$$

и требует больше вычислений. Хорошо подходит для разреженных матриц.

Пример реализации на C++

```
// Условие окончания
bool converge(double *xk, double *xkp)
{
    double norm = 0;
    for (int i = 0; i < n; i++)
        norm += (xk[i] - xkp[i])*(xk[i] - xkp[i]);
    return (sqrt(norm)<eps);
}

/*
    Ход метода, где:
    a[n][n] - Матрица коэффициентов
    x[n], p[n] - Текущее и предыдущее решения
    b[n] - Столбец правых частей
    Все перечисленные массивы вещественные и
    должны быть определены в основной программе,
    также в массив x[n] следует поместить начальное
    приближение столбца решений (например, все нули)
*/

do
{
    for (int i = 0; i < n; i++)
        p[i] = x[i];
    for (int i = 0; i < n; i++)
    {
        double var = 0;
        for (int j = 0; j < i; j++)
            var += (a[i][j] * x[j]);
        for (int j = i + 1; j < n; j++)
            var += (a[i][j] * p[j]);
        x[i] = (b[i] - var) / a[i][i];
    }
}
while (!converge(x, p));
```

Пример реализации на Pascal

```
type ar2d = array [1..50, 1..50] of double;
     ar1d = array [1..50] of double;

procedure seidel(n: byte; e: extended; a: ar2d; b: ar1d; x: ar1d);
var i, j: longint;
    v, m: double;
begin
  // Проверка на совместность
  for i := 1 to n do
    begin
      for j := 1 to n do
        if j <> i then
        begin
          if abs(a[i, j]) >= abs(a[i, i]) then
          begin
            writeln('SLAE is inconsistent' );
            exit;
          end;
        end;

    end;

  // Сам алгоритм
  repeat
    m := 0;
    for i := 1 to n do
      begin
        s := 0;
        for j := 1 to n do
          if i <> j then s := s + a[i, j] * x[j];
        v := x[i];
        x[i] := (b[i] - s) / a[i, i];
        m:=abs(x[i]-v);
      end;
  until m > e;

  // Вывод корней
  writeln('roots: ');
  for i := 1 to n do
    writeln('x[', i, ']= ', x[i]:0:4);
end;
```

Пример реализации на Python

```
from math import sqrt
import numpy as np

def seidel(A, b, eps):
    n = len(A)
    x = [.0 for i in range(n)]

    converge = False
    while not converge:
        x_new = np.copy(x)
        for i in range(n):
            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, n))
            x_new[i] = (b[i] - s1 - s2) / A[i][i]

        converge = sqrt(sum((x_new[i] - x[i]) ** 2 for i in range(n))) <= eps
        x = x_new

    return x
```

См. также

- Геометрическая прогрессия
- Метод простой итерации
- Метод Якоби
- Теорема Банаха
- Метод итерации

Примечания