Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Курсовая работа по курсу « Численные методы»
Тема №2
«Решение систем линейных алгебраических уравнений с несимметричными разреженными матрицами большой размерности.
Метод бисопряженных градиентов»

Выполнил: Баранов А.Д.
Преподаватель: Пивоваров Д.Е.
Группа: М8О-408Б-20
Дата:
Оценка:
Подпись:

Москва, 2023

Теория

Метод бисопряженного градиента (BiConjugate Gradient, BiCG) является итерационным методом для решения систем линейных уравнений, особенно эффективным для несимметричных матриц. Он является вариантом метода сопряженных градиентов, адаптированным для несимметричных случаев.
Рассмотрим систему линейных уравнений Ax=b, где A - неквадратная, несимметричная матрица порядка n, x - вектор неизвестных, b - вектор правой части. Цель метода BiCG - найти приближенное решение этой системы с минимальным числом итераций.
Метод BiCG обеспечивает сходимость к решению для произвольных несимметричных матриц. Он широко применяется в численном моделировании, вычислительной математике, а также в задачах, связанных с линейной алгеброй и оптимизацией. Однако, как и многие итерационные методы, эффективность BiCG может зависеть от свойств конкретной матрицы системы.
Мотивировка предложенного Ван-дер-Ворстом алгоритма BiCGStab (стабилизированный метод бисопряженных градиентов) заключается в обеспечении более гладкой сходимости итераций, поскольку в методе BiCG зачастую наблюдается нерегулярный характер сходимости.

Алгоритм метода

Для решения СЛАУ вида $Ax = b$, где A – комплексная матрица, стабилизированным методом бисопряжённых градиентов может использоваться следующий алгоритм.

Подготовка перед итерационным процессом

1. Выберем начальное приближение x^0
2. $r^0 = b - Ax^0$
3. $\tilde{r} = r^0$
4. $\rho^0 = \alpha^0 = \omega^0 = 1$
5. $v^0 = p^0 = 0$

k-я итерация метода

1. $\rho^k = (\tilde{r}, r^{k-1})$
2. $\beta^k = \frac{p^k}{p^{k-1}} \frac{\alpha^{k-1}}{\omega^{k-1}}$
3. $p^k = r^{k-1} + \beta^k (p^{k-1} - \omega^{k-1} v^{k-1})$
4. $v^k = Ap^k$
5. $\alpha^k = \frac{\rho^k}{(\tilde{r}, v^k)}$
6. $s^k = r^{k-1} - \alpha^k v^k$
7. $t^k = As^k$
8. $\omega^k = \frac{[t^k, s^k]}{[t^k, t^k]}$
9. $x^k = x^{k-1} + \omega^k s^k + \alpha^k p^k$
10. $r^k = s^k - \omega^k t^k$

Критерий остановки итерационного процесса

$||r^k|| < \varepsilon$

Пример №1

Матрица коэффициентов 5x5 с плотностью 0.4.

$$\left(\begin{array}{ccccc|c} 0.341 & 0.0 & 0.0 & 0.704 & 0.0 & 36 \\ 0.0 & 0.0 & 0.542 & 0.0 & 0.578 & 20 \\ 0.0 & 0.0 & 0.305 & 0.416 & 0.0 & 48 \\ 0.0 & 0.0 & 0.215 & 0.0 & 0.0 & 44 \\ 0.182 & 0.961 & 0.0 & 0.0 & 0.435 & 32 \end{array}\right)$$

Подготовка перед итерационным процессом

1. Выберем начальное приближение

$$x^0 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

2.

$$r^0 = b - Ax^0 = \begin{pmatrix} 36 \\ 20 \\ 48 \\ 44 \\ 32 \end{pmatrix} - \begin{pmatrix} 0.341 & 0.0 & 0.0 & 0.704 & 0.0 \\ 0.0 & 0.0 & 0.542 & 0.0 & 0.578 \\ 0.0 & 0.0 & 0.305 & 0.416 & 0.0 \\ 0.0 & 0.0 & 0.215 & 0.0 & 0.0 \\ 0.182 & 0.961 & 0.0 & 0.0 & 0.435 \end{pmatrix} * \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 36 \\ 20 \\ 48 \\ 44 \\ 32 \end{pmatrix}$$

3.

$$\tilde{r} = r^0 = \begin{pmatrix} 36 \\ 20 \\ 48 \\ 44 \\ 32 \end{pmatrix}$$

4. $\rho^0 = \alpha^0 = \omega^0 = 1$

5. $v^0 = p^0 = 0$

1-я итерация метода

1. $\rho^k = (\tilde{r}, r^{k-1}) = (\begin{pmatrix} 36 \\ 20 \\ 48 \\ 44 \\ 32 \end{pmatrix}, \begin{pmatrix} 36 \\ 20 \\ 48 \\ 44 \\ 32 \end{pmatrix}) = 6960$

2. $\beta^k = \frac{p^k}{p^{k-1}} \frac{\alpha^{k-1}}{\omega^{k-1}} = \frac{(6960.0*1)}{(1*1)} = 6960$

3. $p^k = r^{k-1} + \beta^k(p^{k-1} - \omega^{k-1} v^{k-1}) = \begin{pmatrix} 36 \\ 20 \\ 48 \\ 44 \\ 32 \end{pmatrix} + 6960 * (\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} - 1 \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}) = \begin{pmatrix} 36 \\ 20 \\ 48 \\ 44 \\ 32 \end{pmatrix}$

4. $v^k = Ap^k = \begin{pmatrix} 43.252 \\ 44.512 \\ 32.944 \\ 10.32 \\ 39.692 \end{pmatrix}$

5. $\alpha^k = \frac{\rho^k}{(\tilde{r}, v^k)} = \frac{6960}{(\begin{pmatrix} 36 \\ 20 \\ 48 \\ 44 \\ 32 \end{pmatrix}, \begin{pmatrix} 43.252 \\ 44.512 \\ 32.944 \\ 10.32 \\ 39.692 \end{pmatrix})} = 1.209835545802705$

6. $s^k = r^{k-1} - \alpha^k v^k = \begin{pmatrix} 36 \\ 20 \\ 48 \\ 44 \\ 32 \end{pmatrix}$ - 1.209835545802705 * $\begin{pmatrix} 43.252 \\ 44.512 \\ 32.944 \\ 10.32 \\ 39.692 \end{pmatrix} = \begin{pmatrix} -16.32780703 \\ -33.85219981 \\ 8.14317778 \\ 31.51449717 \\ -16.02079248 \end{pmatrix}$

7. $t^k = As^k = \begin{pmatrix} 16.61842381 \\ -4.8464157 \\ 15.59370004 \\ 1.75078322 \\ -42.47266963 \end{pmatrix}$

8. $\omega^k = \frac{[t^k, s^k]}{[t^k, t^k]} = \frac{[\begin{pmatrix} 16.61842381 \\ -4.8464157 \\ 15.59370004 \\ 1.75078322 \\ -42.47266963 \end{pmatrix}, \begin{pmatrix} -16.32780703 \\ -33.85219981 \\ 8.14317778 \\ 31.51449717 \\ -16.02079248 \end{pmatrix}]}{[\begin{pmatrix} 16.61842381 \\ -4.8464157 \\ 15.59370004 \\ 1.75078322 \\ -42.47266963 \end{pmatrix}, \begin{pmatrix} 16.61842381 \\ -4.8464157 \\ 15.59370004 \\ 1.75078322 \\ -42.47266963 \end{pmatrix}]} = 0.3214390064696888$

9. $x^k = x^{k-1} + \omega^k s^k + \alpha^k p^k = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} + 0.3214390064696888 * \begin{pmatrix} -16.32780703 \\ -33.85219981 \\ 8.14317778 \\ 31.51449717 \\ -16.02079248 \end{pmatrix} +$
1.209835545802705 * $\begin{pmatrix} 36 \\ 20 \\ 48 \\ 44 \\ 32 \end{pmatrix} = \begin{pmatrix} 38.30568558 \\ 13.31529344 \\ 60.68964117 \\ 63.36275267 \\ 33.56502985 \end{pmatrix}$

10. $r^k = s^k - \omega^k t^k = \begin{pmatrix} -16.32780703 \\ -33.85219981 \\ 8.14317778 \\ 31.51449717 \\ -16.02079248 \end{pmatrix}$ - 0.3214390064696888 * $\begin{pmatrix} 16.61842381 \\ -4.8464157 \\ 15.59370004 \\ 1.75078322 \\ -42.47266963 \end{pmatrix} =$
$\begin{pmatrix} -21.66961667 \\ -32.29437277 \\ 3.13075433 \\ 30.95172715 \\ -2.36841976 \end{pmatrix}$

Продолжим выполнять итерации до выполнения критерия окончания и сравним результат с результатом, выданным библиотекой Numpy (python). Критерий окончания был выполнен на пятой итерации за 0.01365 сек.

Ответ:

$$x = \begin{pmatrix} 177.1282 \\ 70.95663 \\ 204.65116 \\ -34.66011 \\ -157.30265 \end{pmatrix}$$

Совпал с результатом, выданным библиотекой Numpy.

Пример №2

Двумерная прямоугольная пластина ($0 \le x \le 1,\ 0 \le y \le 1$) подвергается однородным температурным граничным условиям (с верхней поверхностью, поддерживаемой при 100C и все остальные поверхности в 0C) показано на рисунке 1. То есть $T(0, y) = 0,\ T(1, y) = 0,\ T(x, 0) = 0,\ T(x, 1) = 100C$. Нужно найти значение температуры во внутренних точках. Эту задачу можно решить с помощью уравнения Лапласа.

$$\frac{\delta^2 T}{\delta x^2} + \frac{\delta^2 T}{\delta y^2} = 0$$

Дискретизируем это уравнение второго порядка путем замены частных производных их аппроксимациями по схеме крест.

Аппроксимация уравнения Лапласа для внутренних областей может быть выражена как:

$$4T_{i,j} - T_{i-1,j} - T_{i,j}-1 - T_{i+1,j} - T_{i,j+1} = 0$$

Предположим, нас интересуют только значения температуры в девяти внутренних узловых точках. (x_i, y_j), где $x_i = i\Delta x$ и $y_j = j\Delta y,\ i,j = 1,2,3$ с $\Delta x = \Delta y = 0.25$

Однако мы предполагаем симметрию для упрощения задачи. То есть мы предполагаем, что $T_{3,3} = T_{1,3},\ T_{3,2} = T_{1,2},\ T_{3,1} = T_{1,1}$. Таким образом, у нас есть только шесть неизвест- ных: $(T_{1,1}, T_{1,2}, T_{1,3})$ и $(T_{2,1}, T_{2,2}, T_{2,3})$

С помощью уравнения апроксимации Лапласа получим:

$$4T_{1,1} - 0 - T_{1,2} - T_{2,1} - 100 = 0$$
$$4T_{2,1} - T_{1,1} - T_{2,2} - T_{1,1} - 100 = 0$$
$$4T_{1,2} - 0 - T_{1,3} - T_{2,2} - T_{1,1} = 0$$
$$4T_{2,2} - T_{1,2} - T_{2,3} - T_{1,2} - T_{2,1} = 0$$

После подходящей перестановки эти уравнения можно записать в следующем виде:

$$\begin{pmatrix} 4 & -1 & -1 & 0 & 0 & 0 \\ -2 & 4 & 0 & -1 & 0 & 0 \\ -1 & 0 & 4 & -1 & -1 & 0 \\ 0 & -1 & -2 & 4 & 0 & -1 \\ 0 & 0 & -1 & 0 & 4 & -1 \\ 0 & 0 & 0 & -1 & -2 & 4 \end{pmatrix} \begin{pmatrix} T_{1,1} \\ T_{2,1} \\ T_{1,2} \\ T_{2,2} \\ T_{1,3} \\ T_{2,3} \end{pmatrix} = \begin{pmatrix} 100 \\ 100 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

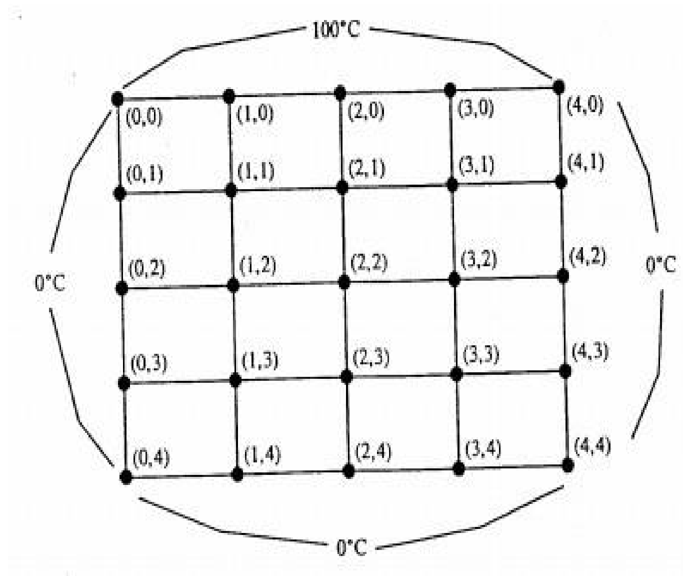

Рис. 1

Решим эту систему с помощью BiCGStab:

$$x = \begin{pmatrix} 42.85714 \\ 52.67857 \\ 18.75 \\ 25 \\ 7.14286 \\ 9.82143 \end{pmatrix}$$

Кол-во итераций: 5

Среднее значение:26.041666666666668

Решение найдено за: 0.02828 сек

Исходный код:

```
import numpy as np
from numpy.linalg import norm
from scipy.sparse import diags, csc_matrix
from time import time



def getMatrix(filename, isDiag):
    with open(filename) as f:
        shape = int(f.readline())
        matrix = [[float(num) for num in line.split()]
                  for _, line in zip(range(shape), f)]
        if isDiag:
            matrix[0].insert(0, 0)
            matrix[-1].append(0) a,
            b, c = zip(*matrix)
            matrix = diags([a[1:], b, c[:-1]], [-1, 0, 1])
            matrix = csc_matrix(matrix)
        else:
            matrix = csc_matrix(matrix)
        b = np.array([float(num) for num in f.readline().split()])
        return matrix, b



def biCGStabSolve(matrix, b, eps, shape, x0, k):
    r0 = b - matrix @ x0
    x0 = x0
    r2 = r0
    rho0 = 1
    alpha0 = 1
    omega0 = 1
    v0 = np.array([0] * shape)
    p0 = np.array([0] * shape)
    while True:
        rho = r2 @ r0
        beta = (rho * alpha0) / (rho0 * omega0)
        p = r0 + beta * (p0 - omega0 * v0)
        v = matrix @ p
        alpha = rho / (r2 @ v)
        s = r0 - alpha * v
        t = matrix @ s
        omega = (t @ s) / (t @ t)
        x = x0 + omega * s + alpha * p
        r = s - omega * t

        k += 1
        if norm(r) < eps:
            break
        r0 = r
        rho0 = rho
        alpha0 = alpha
        omega0 = omega
        v0 = v
        p0 = p
        x0 = x
    return x
```

```
def print_solution(matrix, b):
    eps = 1e-5
    shape  =  matrix.shape[0]
    x0 = np.array([0] * shape)
    k = 0

    start = time()
    x = biCGStabSolve(matrix, b, eps, shape, x0, k)
    end = time()
    start2 = time()
    x2 = np.linalg.solve(matrix.toarray(), b)
    end2 = time()
    print('My solve:\n')
    print(f'{x.round(5)}\n')
    print(f'EPS = {eps}\n')
    print(f'Shape = {shape}\n')
    print(f'Count of iterations = {k}\n')
    print(f'Mean = {np.mean(x)}\n')
    print(f'Time = {round(end - start, 5)} sec\n')
    print('\nNumPy solve:\n')
    print(f'{x2.round(5)}\n')
    print(f'Mean = {np.mean(x2)}\n')
    print(f'Time = {round(end2 - start2, 5)} sec\n')


matrix, b = getMatrix('test10', False)
printSolution(matrix, b)

matrix, b = getMatrix('test20', False)
printSolution(matrix, b)

matrix, b = getMatrix('test30', False)
printSolution(matrix, b)
```

Вывод программы

Входные данные:

Выходные данные: кол-во итераций, среднее значение и время за которое выполнился поиск решения.

В данном случае на вход подается случайное СЛАУ с 10 уравнениями:

```
My solve:

[-195.34591   26.53793  309.59768  -34.31888   47.3909  -126.5331
  214.97703  -20.73636  249.00413 -260.32933]

EPS = 1e-05

Shape = 10

Count of iterations = 0

Mean = 21.024408467227477

Time = 0.00067 sec


NumPy solve:

[-195.34591   26.53793  309.59768  -34.31888   47.3909  -126.5331
  214.97703  -20.73636  249.00413 -260.32933]

Mean = 21.024408466560754

Time = 5e-05
```

В данном случае на вход подается случайное СЛАУ с 20 уравнениями:

```
My solve:

[-18.23957 -14.64113 -10.45047  33.47596  23.60765 -44.7088   45.67713
   8.19735  33.83973   8.87599 -25.66239  -9.31423 -19.90542 -23.84705
  34.99552  29.66388  22.22547 -12.89704  54.27331  46.53818]

EPS = 1e-05

Shape = 20

Count of iterations = 0

Mean = 8.08520327117203

Time = 0.00474 sec
```

```
NumPy solve:

[-18.23957 -14.64113 -10.45047  33.47596  23.60765 -44.7088   45.67713
   8.19735  33.83973   8.87599 -25.66239  -9.31423 -19.90542 -23.84705
  34.99552  29.66388  22.22547 -12.89705  54.27331  46.53818]

Mean = 8.085203088065231

Time = 9e-05 sec
```

В данном случае на вход подается случайное СЛАУ с 30 уравнениями:

```
My solve:

[ -53.64412  -35.47177 -118.89011  104.64771 -128.42506  249.20309
   70.78105   57.49387 -158.63259  -76.66191    6.34065  311.24475
   96.8618    29.04643  -64.97601  -95.87315   74.15824  234.61787
   80.88134   72.80343  -74.44886   58.00882 -148.87989  -63.78908
   99.56411 -122.365   -133.83062   86.21966  -89.91691   25.67745]

EPS = 1e-05

Shape = 30

Count of iterations = 0

Mean = 9.724839647431981

Time = 0.01143 sec


NumPy solve:

[ -53.64412  -35.47177 -118.89009  104.64771 -128.42506  249.20307
   70.78104   57.49385 -158.6326   -76.66189    6.34065  311.24474
   96.8618    29.04643  -64.976    -95.87315   74.15823  234.61787
   80.88134   72.80342  -74.44885   58.00882 -148.87987  -63.78908
   99.56411 -122.36499 -133.83061   86.21965  -89.9169    25.67744]

Mean = 9.724839707418928

Time = 0.00016 sec
```

Вывод

В данной курсовой работе была рассмотрена актуальная проблема решения систем линейных алгебраических уравнений, особенно сфокусированная на случае несимметричных разреженных матриц большой размерности. Одним из ключевых инструментов для решения таких задач является метод бисопряженных градиентов.

Метод бисопряженных градиентов (Conjugate Gradient method) представляет собой итерационный алгоритм, эффективно решающий системы линейных уравнений с симметричными и положительно определенными матрицами. В данной работе был проанализирован и адаптирован этот метод для случая несимметричных матриц, что расширяет его применимость в различных областях, таких как численное моделирование, машинное обучение и другие.

Основной упор был сделан на изучение метода бисопряженных градиентов как эффективного итерационного средства для решения систем линейных уравнений с несимметричными разреженными матрицами больших размерностей. Были представлены теоретические основы метода, описаны основные шаги его работы.

В результате работы можно заключить, что метод бисопряженных градиентов представляет собой мощный инструмент для решения систем линейных уравнений с несимметричными разреженными матрицами. Его преимущества в эффективности и относительной простоте реализации делают его значимым элементом в численных методах для решения сложных задач в науке и инженерии. Однако, для успешного применения метода в конкретных прикладных задачах, важно учитывать особенности структуры матрицы, тщательно выбирать параметры и контролировать процесс сходимости.

Список литературы

[1] *Методы бисопряженных градиентов в подпространствах Крылова, В.П.Ильин*

[2] *Wikipedia, Стабилизированный метод бисопряженных градиентов.*