

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."

Студент: А. О. Дубинин  
Преподаватель: И. Э. Иванов  
Группа: М8О-306Б  
Дата:  
Оценка:  
Подпись:

Москва, 2020

## Теория.

Стабилизированный метод бисопряжённых градиентов (англ. Biconjugate gradient stabilized method, BiCGStab) — итерационный метод решения СЛАУ крыловского типа. Разработан Ван дэр Ворстом для решения систем с несимметричными матрицами. Сходится быстрее, чем обычный метод бисопряжённых градиентов, который является неустойчивым, и поэтому применяется чаще.

### 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.  $v^0 = p^0 = 0$

### к-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**

Кроме традиционных критериев остановки, как число итераций ( $k \leq k_{max}$ ) и заданная невязка ( $\frac{\|r^k\|}{\|b\|} < \varepsilon$ ), так же остановку метода можно производить, когда величина  $|\omega^k|$  стала меньше некоторого заранее заданного числа  $\varepsilon_\omega$ .

## Пример.

Возьмем небольшой пример для наглядности. 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.  $\rho^k = (\tilde{r}, r^{k-1}) = \left( \begin{pmatrix} 36 \\ 20 \\ 48 \\ 44 \\ 32 \end{pmatrix}, \begin{pmatrix} 36 \\ 20 \\ 48 \\ 44 \\ 32 \end{pmatrix} \right) = 6960$

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

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

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

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

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

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

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

$$\begin{aligned}
9. \quad x^k &= x^{k-1} + \omega^k s^k + \alpha^k p^k = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} + 0.3214390064696888 * \begin{pmatrix} -16.32780703 \\ -33.85219981 \\ 8.14317778 \\ 31.51449717 \\ -16.02079248 \end{pmatrix} + \\
&1.209835545802705 * \begin{pmatrix} 36 \\ 20 \\ 48 \\ 44 \\ 32 \end{pmatrix} = \begin{pmatrix} 38.30568558 \\ 13.31529344 \\ 60.68964117 \\ 63.36275267 \\ 33.56502985 \end{pmatrix} \\
10. \quad r^k &= s^k - \omega^k t^k = \begin{pmatrix} -16.32780703 \\ -33.85219981 \\ 8.14317778 \\ 31.51449717 \\ -16.02079248 \end{pmatrix} - 0.3214390064696888 * \begin{pmatrix} 16.61842381 \\ -4.8464157 \\ 15.59370004 \\ 1.75078322 \\ -42.47266963 \end{pmatrix} = \\
&\begin{pmatrix} -21.66961667 \\ -32.29437277 \\ 3.13075433 \\ 30.95172715 \\ -2.36841976 \end{pmatrix}
\end{aligned}$$

Продолжим выполнять итерации до выполнения критерия окончания и сравним ответ с ответом, выдаваемым библиотекой numru.

Критерий окончания был выполнен на 5 итерации за 0.01365 сек.

Ответ:

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

Ответ выдаваемый numru совпал с нашим, но выполнение произошло намного быстрее, за 0.00016 сек..

Сравним выполнение на случайном слау на 100 элементов с плотностью 0.4.

Наш алгоритм решил эту задачу за 0.85578 сек., в то время, как numru за 0.00342 сек., но хочется отметить, что так как мы задали точность  $\varepsilon = 0.01$ , то точность решения numru оказалась лучше, как мы можем судить по среднему значению вектора ответа

BiCGStab: Среднее значение ответа = -1.8889739760582946

Numru: Среднее значение ответа = -1.8889739760583342

# 1 Исходный код

```
1 import argparse
2 import numpy as np
3 from numpy.linalg import norm
4 from scipy.sparse import diags, csc_matrix
5 from scipy.sparse.linalg import bicgstab, spilu
6 from time import time
7
8
9 def get_matrix(filename, is_diag):
10     with open(filename) as f:
11         shape = int(f.readline())
12         matrix = [[float(num) for num in line.split()]
13                   for _, line in zip(range(shape), f)]
14         if is_diag:
15             matrix[0].insert(0, 0)
16             matrix[-1].append(0)
17             a, b, c = zip(*matrix)
18             matrix = diags([a[1:], b, c[:-1]], [-1, 0, 1])
19             matrix = csc_matrix(matrix)
20         else:
21             # matrix = np.array([np.array(xi) for xi in matrix])
22             matrix = csc_matrix(matrix)
23         b = np.array([float(num) for num in f.readline().split()])
24         return matrix, b
25
26
27 class BiCGStab:
28     def __init__(self, matrix, b, output_file, log_file,
29                 x0=None, eps=1e-5):
30         self.output = 'res_default' if output_file is None else output_file
31         self.log = 'log_default' if log_file is None else log_file
32         self.matrix = matrix
33         self.b = b
34         self.eps = eps
35         self.shape = matrix.shape[0]
36         self.x0 = np.array([0] * self.shape) if x0 is None else x0
37         self.k = 0
38
39     def solve(self):
40         f = open(self.log, 'w')
41         r0 = self.b - self.matrix self.x0
42         x0 = self.x0
43         r2 = r0
44         rho0 = 1
45         alpha0 = 1
46         omega0 = 1
47         v0 = np.array([0] * self.shape)
```

```

48 p0 = np.array([0] * self.shape)
49 f.write(f'r^0 = {self.b} - A {self.x0} = {r0}\n')
50 f.write(f'\tilde{{r^0}} = {r0} \n')
51 f.write(f'\rho^0 = \alpha^0 = \omega^0 = 1 \n')
52 f.write(f'\v^0 = p^0 = 0 \n')
53 while True:
54     rho = r2 r0
55     beta = (rho * alpha0) / (rho0 * omega0)
56     p = r0 + beta * (p0 - omega0 * v0)
57     v = self.matrix p
58     alpha = rho / (r2 v)
59     s = r0 - alpha * v
60     t = self.matrix s
61     omega = (t s) / (t t)
62     x = x0 + omega * s + alpha * p
63     r = s - omega * t
64
65     # Logging
66     f.write(f'Iter: {self.k}\n')
67     f.write(f'\rho^k = ({r2}, {r0}) = {rho}\n')
68     f.write(f'\beta^k = ({rho} * {alpha0}) / ({rho0} * {omega0}) = {beta}\n')
69     f.write(f'p^k = {r0} + {beta} * ({p0} - {omega0} * {v0}) = {p}\n')
70     f.write(f'v^k = {v}\n')
71     f.write(f'\alpha^k = {rho} / ({r2}, {v}) = {alpha}\n')
72     f.write(f's^k = {r0} - {alpha} * {v} = {s}\n')
73     f.write(f't^k = {t}\n')
74     f.write(f'\omega^k = ({t} , {s}) / ({t} , {t}) = {omega}\n')
75     f.write(f'x^k = {x0} + {omega} * {s} + {alpha} * {p} = {x}\n')
76     f.write(f'r^k = {s} - {omega} * {t} = {r}\n')
77     f.write('=' * 10 + '\n')
78
79     self.k += 1
80     if norm(r) < self.eps:
81         break
82     r0 = r
83     rho0 = rho
84     alpha0 = alpha
85     omega0 = omega
86     v0 = v
87     p0 = p
88     x0 = x
89 f.close()
90 return x
91
92 def precondition_solve(self):
93     f = open(self.log, 'a')
94     k_mtx = spilu(self.matrix)
95     r0 = self.b - self.matrix self.x0
96     x0 = self.x0

```



```

97     r2 = r0 #r_tilda
98     p = r0
99     while True:
100         if r0 r2 == 0:
101             f.write(f'Error\nIters = self.k')
102             exit()
103         pp = k_mtrx.solve(p)
104         f.write('Kp* = p solved\n')
105         tmp = self.matrix pp
106         alpha = (r0 r2) / (tmp r2)
107         s = r0 - alpha * tmp
108         if norm(s) < self.eps:
109             self.k += 1
110             x = x0 + alpha * pp
111             break
112         z = k_mtrx.solve(s)
113         f.write('Kz = s solved\n')
114         tmp2 = self.matrix z
115         omega = (tmp2 s) / (tmp2 tmp2)
116         x = x0 + alpha * pp + omega * z
117         r = s - omega * tmp2
118         beta = (r r2) / (r0 r2) * (alpha / omega)
119         p = r + beta * (p - omega * tmp)
120         self.k += 1
121         if norm(r) < self.eps:
122             break
123         x0 = x
124         r0 = r
125
126         # Logging
127         f.write(f'Iter: {self.k}\nx0 = {x0}\np* = {pp}\nbeta = {beta}\n')
128         f.write(f'z = {z}\nomega = {omega}\nr2 = {r2}\nr0 = {r0}')
129         f.write('\n' + '=' * 10 + '\n')
130
131     f.close()
132     return x
133
134
135 def print_solution(self):
136     start = time()
137     # x = self.solve()
138     x = self.precond_solve()
139     end = time()
140     start2 = time()
141     # x2 = bicgstab(self.matrix, self.b, tol=self.eps, x0=self.x0)[0]
142     x2 = np.linalg.solve(self.matrix.toarray(), self.b)
143     end2 = time()
144     with open(self.output, 'w') as f:
145         f.write('My solve:\n')

```

```

146         f.write(f'{x.round(5)}\n')
147         f.write(f'EPS = {self.eps}\n')
148         f.write(f'Shape = {self.shape}\n')
149         f.write(f'Count of iterations = {self.k}\n')
150         f.write(f'Mean = {np.mean(x)}\n')
151         f.write(f'Time = {round(end - start, 5)} sec\n')
152         f.write(f'\nNumPy solve:\n')
153         f.write(f'{x2.round(5)}\n')
154         f.write(f'Mean = {np.mean(x2)}\n')
155         f.write(f'Time = {round(end2 - start2, 5)} sec\n')
156
157
158 def main():
159     parser = argparse.ArgumentParser()
160     parser.add_argument('--input', required=True, help='Input file')
161     parser.add_argument('--output', required=True, help='Output file')
162     parser.add_argument('--log', help='Log file')
163     parser.add_argument('--eps', type=float, help='Epsilon', default=1e-2)
164     parser.add_argument('--diag', help='If matrix is diag', \
165                         action='store_true')
166     args = parser.parse_args()
167
168     matrix, b = get_matrix(args.input, args.diag)
169
170     solver = BiCGStab(matrix, b, output_file=args.output,
171                      log_file=args.log, eps=args.eps)
172     solver.print_solution()
173
174 if __name__ == "__main__":
175     main()

```

## 2 Вывод программы

Входные данные: Имя входной файл с данными слау и имя выходного файла для записи ответа.

Выходные данные: В выходном файле ответ, кол-во итераций, среднее значение и время за которое выполнялся поиск решения.

В данном случае на вход подается случайное слау с 30 уравнениями и плотностью коэффициентов 0.4.

```
art@mars:~/study/NM/cp/code python3 bicgstab.py --input tests/rand_mtrx30 --output  
output --log log
```

```
art@mars:~/study/NM/cp/code cat output
```

My solve:

```
[-9.4680000e-02  5.9079670e+01 -5.5810300e+00 -8.8898560e+01  
-6.2004470e+01  1.2041147e+02  2.1709040e+01  1.3667503e+02  
-1.6318042e+02  3.9331000e+00  2.3225710e+01 -2.1930390e+01  
-9.8933780e+01 -1.2530712e+02  7.5075550e+01  3.0164300e+00  
1.3420510e+02 -2.8925500e+00  1.6446969e+02 -1.0715606e+02  
1.6687371e+02 -6.8702610e+01  1.5355362e+02  1.5282011e+02  
-2.9798310e+01 -1.2562731e+02  3.8566960e+01 -2.2770390e+02  
3.5455800e+01 -6.0865160e+01]
```

EPS = 0.01

Shape = 30

Count of iterations = 108

Mean = 3.346488050611977

Time = 0.07234 sec

NumPy solve:

```
[-9.5120000e-02  5.9079290e+01 -5.5813200e+00 -8.8897210e+01  
-6.2004170e+01  1.2041065e+02  2.1708750e+01  1.3667395e+02  
-1.6317770e+02  3.9336300e+00  2.3226360e+01 -2.1929640e+01  
-9.8932600e+01 -1.2530600e+02  7.5074460e+01  3.0164200e+00  
1.3420279e+02 -2.8922400e+00  1.6446647e+02 -1.0715443e+02  
1.6687129e+02 -6.8700980e+01  1.5355195e+02  1.5281823e+02  
-2.9798240e+01 -1.2562546e+02  3.8566770e+01 -2.2770101e+02  
3.5455340e+01 -6.0864470e+01]
```

Mean = 3.3465244909640166

Time = 0.00027 sec

### 3 Выводы

Благодаря этой лабораторной работе, я узнал ещё один новый способ, которым можно решать слау с разреженной матрицей коэффициентов, метод хорошо работает, если уметь быстро вычислять шаги, которые применяются в этом методе.

## Список литературы

- [1] *Wiki. Стабилизированный метод бисопряжённых градиентов*  
*[https://ru.wikipedia.org/wiki/Стабилизированный\\_метод\\_бисопряжённых\\_градиентов](https://ru.wikipedia.org/wiki/Стабилизированный_метод_бисопряжённых_градиентов)*