

作业三：线性方程组求解

英才 1701 赵鹏威 U201710152

2019 年 10 月 3 日

目录

1	引言	2
2	问题描述	2
3	程序实现	2
3.1	Gauss 消元法	3
3.2	Doolittle 分解法	4
3.3	Gauss-Seidel 迭代法	7
4	运行时结果	8
4.1	Gauss 消元法	8
4.2	Doolittle 分解法	8
4.3	超松弛 Gauss-Seidel 迭代法	9

1 引言

在解决实际问题的時候，常常會遇到求解線性方程組的問題，比如使用有限差分法解偏微分方程。往往線性方程組的個數會達到上萬個，因此研究高效的算法很有必要。這次作業主要使用 Gauss 消元法、Doolittle 分解法、Gauss-Seidel 迭代法和超鬆弛迭代法來解線性方程組。

2 問題描述

問題 1. 分別使用 Gauss 消元法、Doolittle 分解法、超鬆弛 Gauss-Seidel 迭代法解以下方程組

$$Ax = \begin{bmatrix} -15 \\ 27 \\ -23 \\ 0 \\ -20 \\ 12 \\ -7 \\ 10 \end{bmatrix} \quad A = \begin{bmatrix} 31 & -13 & 0 & 0 & 0 & -10 & 0 & 0 & 0 \\ -13 & 35 & -9 & 0 & -11 & 0 & 0 & 0 & 0 \\ 0 & -9 & 31 & -10 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -10 & 79 & -30 & 0 & 0 & 0 & -9 \\ 0 & 0 & 0 & -30 & 57 & -7 & 0 & -5 & 0 \\ 0 & 0 & 0 & 0 & -7 & 47 & -30 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -30 & 41 & 0 & 0 \\ 0 & 0 & 0 & 0 & -5 & 0 & 0 & 27 & -2 \\ 0 & 0 & 0 & -9 & 0 & 0 & 0 & -2 & 29 \end{bmatrix}$$

3 程序實現

為了方便，使用擴展矩陣來儲存線性方程組的全部信息。由於上面提到的所有方法都不希望對角元出現 0，因此在計算之前，先對矩陣做列主元變換。代碼見 Listing 1。

Listing 1: 列主元變換

```
1  subroutine pivot_exchange(matrix, ndim1, ndim2)
2      implicit none
3      integer(8), intent(in) :: ndim1, ndim2
4      real(8), dimension(ndim1, ndim2) :: matrix
5
6      integer :: col, i
7      real(8), dimension(ndim2) :: tmp_row
8
9      do col = 1, ndim2
10         do i = col, ndim1
11             if (matrix(i, col) > matrix(col, col)) then
12                 tmp_row = matrix(col, :)
13                 matrix(col, :) = matrix(i, :)
14                 matrix(i, :) = tmp_row
15             end if
16         end do
```

```

17     end do
18
19     return
20 end subroutine

```

所有的求解方法都被封装在一个叫做 `linear_eqs_solver` 的 module 里. 其中还包含了一个常数 `EPSILON`, 用来控制求解的精度, 默认为 `1e-3`. 完整代码见附录.

3.1 Gauss 消元法

Gauss 消元法每次消元后矩阵大部分的值都改变了, 所以 Gauss 消元法时不用上面这个列主元变换子程序, 而是内置一个, 即在每次消某一行之前对这一列做列主元变换. 流程图如图1所示. 代码见 Listing 2.

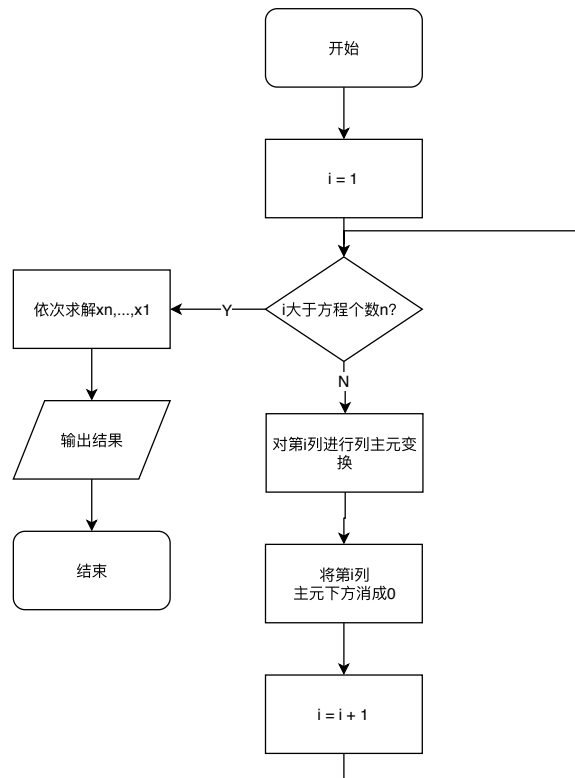


图 1: Gauss 消元法流程图

Listing 2: Gauss 消元法

```

1  subroutine gauss_elimination(co_matrix, ndim, solution)
2      implicit none
3      integer(8), intent(in) :: ndim
4      real(8), dimension(ndim, ndim+1), intent(in) :: co_matrix

```

```

5  type(result), dimension(ndim), intent(out) :: solution
6
7  real(8), dimension(ndim, ndim+1) :: A
8  real(8), dimension(ndim) :: tmp_row, X
9  integer(8) :: t, i
10
11  A = co_matrix
12  do t = 1, ndim
13      ! pivot exchange
14      do i = t, ndim
15          if (abs(A(i, t)) > abs(A(t, t))) then
16              tmp_row = A(t, :)
17              A(t, :) = A(i, :)
18              A(i, :) = tmp_row
19          end if
20      end do
21      ! Gauss elimination
22      do i = t+1, ndim
23          A(i, :) = A(i, :) - A(i, t) / A(t, t) * A(t, :)
24      end do
25  end do
26
27  print "('The Gauss-eliminated extented coefficients matrix is')
28  print "(10f7.2)", (A(i, :), i=1,9)
29
30  X = 0.0d0
31  do t = ndim, 1, -1
32      X(t) = (A(t, ndim+1) - dot_product(A(t, t+1:ndim), X(t+1:ndim))) / A(t, t)
33  end do
34
35  solution%value = X
36  solution%error = 0.0d0
37
38  return
39 end subroutine

```

3.2 Doolittle 分解法

Doolittle 分解法将系数矩阵分解为一个下三角矩阵和一个上三角矩阵的乘积.

$$A = LU$$

L 和 U 各分量的值可以由下式得到

$$u_{kj} = a_{kj} - \sum_{r=1}^{k-1} l_{kr} u_{rj} \quad j = k, \dots, n$$

$$l_{ik} = \frac{a_{ik} - \sum_{r=1}^{k-1} l_{ir} u_{rk}}{u_{kk}} \quad i = k+1, \dots, n$$

方程最终的解分两步得到

$$y_i = b_i - \sum_{j=1}^{i-1} l_{ij} y_j \quad i = 1, \dots, n$$

$$x_i = \frac{y_i - \sum_{j=i+1}^n u_{ij} x_j}{u_{ii}} \quad i = n, \dots, 1$$

由于 Gauss 消元法和 Doolittle 分解法都是精确的算法，因此最后输出的结果误差为 0（实际上不是严格的 0，但是误差会比双精度浮点数最小的那个数小）。流程图见图2. 代码见 Listing 3.

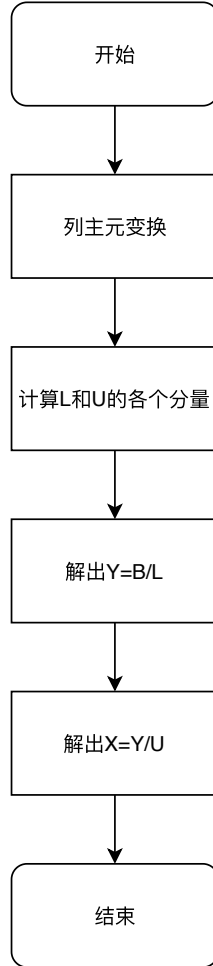


图 2: Doolittle 分解法流程图

Listing 3: Doolittle 分解法

```

1  subroutine doolittle(co_matrix, ndim, solution)
2      implicit none
3      integer(8), intent(in) :: ndim
4      real(8), dimension(ndim, ndim+1), intent(in) :: co_matrix
5      type(result), dimension(ndim), intent(out) :: solution
6
7      real(8), dimension(ndim, ndim+1) :: A
8      real(8), dimension(ndim, ndim) :: L, U
9      real(8), dimension(ndim) :: X, Y
10     integer(8) :: i, j, k
11
12     A = co_matrix
13     L = 0.0d0; U = 0.0d0;
14
15     call pivot_exchange(A, ndim, ndim+1)
16
17     do k = 1, ndim
18         do j = k, ndim
19             U(k, j) = A(k, j) - dot_product(L(k, :k-1), U(:,k-1, j))
20         end do
21         L(k, k) = 1.0d0
22         do i = k+1, ndim
23             L(i, k) = (A(i, k) - dot_product(L(i, :k-1), U(:,k-1, k))) / U(k, k)
24         end do
25     end do
26
27     do i = 1, ndim
28         Y(i) = A(i, ndim+1) - dot_product(L(i, :i-1), Y(:,i-1))
29     end do
30     do i = ndim, 1, -1
31         X(i) = (Y(i) - dot_product(U(i, i+1:), X(i+1:))) / U(i, i)
32     end do
33
34     solution%value = X
35     solution%error = 0.0d0
36
37     return
38 end subroutine

```

3.3 Gauss-Seidel 迭代法

Gauss-Seidel 迭代法的子程序其中的参数 ω 是可选参数. 如果不输入 ω , 那么默认 ω 的值为 1, 即 Gauss-Seidel 迭代; 如果输入 ω , 若大于 1 则是 overrelaxation 迭代, 若小于 1 则是 underrelaxation 迭代. 迭代初值设置为零向量. 迭代结果的误差大小取的是最后两次迭代的差值. 流程图见图3. 代码见 Listing 4.

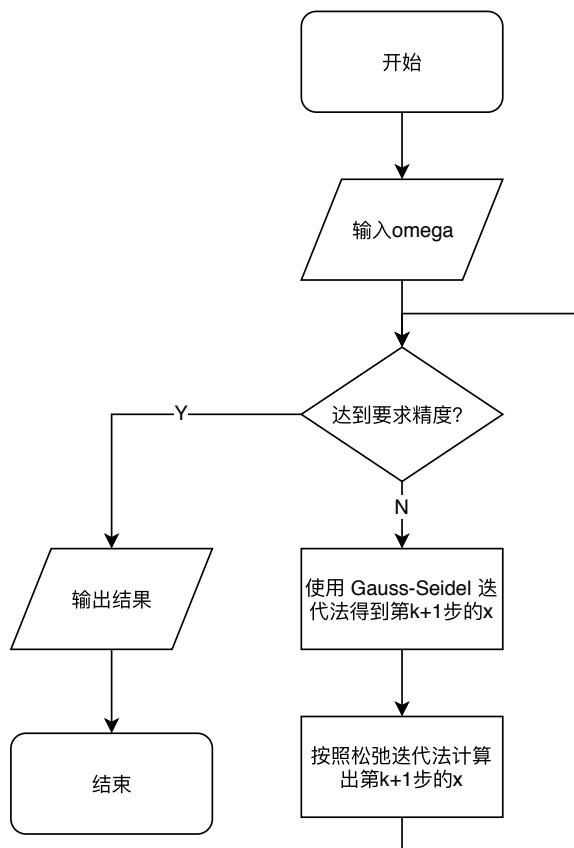


图 3: 使用松弛迭代的 Gauss-Seidel 迭代法流程图

Listing 4: Gauss-Seidel 迭代法

```
1  subroutine gauss_seidel_iteration(co_matrix, ndim, solution, omega)
2      implicit none
3      integer(8), intent(in) :: ndim
4      real(8), optional, intent(in) :: omega
5      real(8), dimension(ndim, ndim+1), intent(in) :: co_matrix
6      type(result), dimension(ndim), intent(out) :: solution
7
8      real(8), dimension(ndim, ndim+1) :: A
9      real(8) :: lambda = 1.0d0
10     real(8), dimension(ndim) :: X, X_tmp, Error
```

```

11     integer(8) :: iter, i
12
13     A = co_matrix
14     call pivot_exchange(A, ndim, ndim+1)
15
16     if (present(omega)) then
17         lambda = omega
18     end if
19
20     X = 0.0d0
21     do iter = 1, 10000
22         X_tmp = X
23         do i = 1, ndim
24             X(i) = -(dot_product(A(i, :ndim), X) - A(i, i) * X(i) - A(i, ndim+1)) / A(i, i)
25         end do
26         X = (1 - lambda) * X_tmp + lambda * X
27         Error = abs(X - X_tmp)
28
29         if (maxval(Error) < EPSILON) then
30             exit
31         end if
32
33     end do
34     print "('Iter: ',i4)", iter
35     solution%value = X
36     solution%error = Error
37
38     return
39 end subroutine

```

4 运行时结果

4.1 Gauss 消元法

Gauss 消元法的运行时结果如图4所示. 得到的结果为

$$x = \begin{bmatrix} -0.2892 & 0.3454 & -0.7128 & -0.2206 & -0.4304 & 0.1543 & -0.0578 & 0.2011 & 0.2982 \end{bmatrix}^T$$

命令行中 Checking results... 的下一行是将数值结果代回原方程得到的值, 可见完全满足方程组 (也就是和拓展矩阵的最后一列相等)

4.2 Doolittle 分解法

Doolittle 分解法的运行时结果如图5, 结果与 Gauss 消元法完全一致.


```

zipwin@WorldGate: ~/WorkPlace/fortran/computational_physics/assignment3/task0
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
+ task0 gfortran linear_eqs_solver.f90 -o linear_eqs_solver
+ task0 ./linear_eqs_solver
The extended coefficients matrix is
 31.00 -13.00  0.00  0.00  0.00 -10.00  0.00  0.00  0.00 -15.00
-13.00 35.00 -9.00  0.00 -11.00  0.00  0.00  0.00  0.00 27.00
  0.00 -9.00 31.00 -10.00  0.00  0.00  0.00  0.00  0.00 -23.00
  0.00  0.00 -10.00 79.00 -30.00  0.00  0.00  0.00 -9.00  0.00
  0.00  0.00  0.00 -30.00 57.00 -7.00  0.00 -5.00  0.00 -20.00
  0.00  0.00  0.00  0.00 -7.00 47.00 -30.00  0.00  0.00 12.00
  0.00  0.00  0.00  0.00  0.00 -30.00 41.00  0.00  0.00 -7.00
  0.00  0.00  0.00  0.00 -5.00  0.00  0.00 27.00 -2.00  7.00
  0.00  0.00  0.00 -9.00  0.00  0.00  0.00 -2.00 29.00 10.00
The Gauss-eliminated extended coefficients matrix is
 31.00 -13.00  0.00  0.00  0.00 -10.00  0.00  0.00  0.00 -15.00
  0.00 29.55 -9.00  0.00 -11.00 -4.19  0.00  0.00  0.00 20.71
  0.00  0.00 28.26 -10.00 -3.35 -1.28  0.00  0.00  0.00 -16.69
  0.00  0.00  0.00 75.46 -31.19 -0.45  0.00  0.00 -9.00 -5.91
  0.00  0.00  0.00  0.00 44.60 -7.18  0.00 -5.00 -3.58 -22.35
  0.00  0.00  0.00  0.00 -0.00 45.87 -30.00 -0.78 -0.56  8.49
  0.00  0.00  0.00  0.00 -0.00 -0.00 21.38 -0.51 -0.37 -1.45
  0.00  0.00  0.00  0.00 -0.00 -0.00  0.00 26.41 -2.42  4.61
  0.00  0.00  0.00  0.00 -0.00 -0.00  0.00  0.00 27.39  7.95
The solution vector is
-0.2892  0.3454 -0.7128 -0.2206 -0.4304  0.1543 -0.0578  0.2011  0.2902
The errors are
 0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
Checking result...
-15.00 27.00 -23.00 -0.00 -20.00 12.00 -7.00  7.00 10.00
+ task0

```

图 4: Gauss 消元法

```

zipwin@WorldGate: ~/WorkPlace/fortran/computational_physics/assignment3/task0
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
+ task0 gfortran linear_eqs_solver.f90 -o linear_eqs_solver
+ task0 ./linear_eqs_solver
The extended coefficients matrix is
 31.00 -13.00  0.00  0.00  0.00 -10.00  0.00  0.00  0.00 -15.00
-13.00 35.00 -9.00  0.00 -11.00  0.00  0.00  0.00  0.00 27.00
  0.00 -9.00 31.00 -10.00  0.00  0.00  0.00  0.00  0.00 -23.00
  0.00  0.00 -10.00 79.00 -30.00  0.00  0.00  0.00 -9.00  0.00
  0.00  0.00  0.00 -30.00 57.00 -7.00  0.00 -5.00  0.00 -20.00
  0.00  0.00  0.00  0.00 -7.00 47.00 -30.00  0.00  0.00 12.00
  0.00  0.00  0.00  0.00  0.00 -30.00 41.00  0.00  0.00 -7.00
  0.00  0.00  0.00  0.00 -5.00  0.00  0.00 27.00 -2.00  7.00
  0.00  0.00  0.00 -9.00  0.00  0.00  0.00 -2.00 29.00 10.00
The solution vector is
-0.2892  0.3454 -0.7128 -0.2206 -0.4304  0.1543 -0.0578  0.2011  0.2902
The errors are
 0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
Checking result...
-15.00 27.00 -23.00 -0.00 -20.00 12.00 -7.00  7.00 10.00
+ task0

```

图 5: Doolittle 分解法

4.3 超松弛 Gauss-Seidel 迭代法

设置 ω 为 1.5, 使用 Gauss-Seidel 迭代法的运行时结果如图6所示. 得到的结果为

$$x = \begin{bmatrix} -0.2895 & 0.3455 & -0.7128 & -0.2206 & -0.4304 & 0.1543 & -0.0578 & 0.2011 & 0.2982 \end{bmatrix}^T$$

除第一个数有 0.0007 的误差外, 其他的都没有误差. 达到需求的精度迭代了 19 次. 代回方程组检验发现, 第一个方程有 0.01 的偏差.

```

zipwin@WorldGate: ~/WorkPlace/fortran/computational_physics/assignment3/task0
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
+ task0 gfortran linear_eqs_solver.f90 -o linear_eqs_solver
+ task0 ./linear_eqs_solver
The extended coefficients matrix is
 31.00 -13.00  0.00  0.00  0.00 -10.00  0.00  0.00  0.00 -15.00
-13.00 35.00 -9.00  0.00 -11.00  0.00  0.00  0.00  0.00 27.00
 0.00 -9.00 31.00 -10.00  0.00  0.00  0.00  0.00  0.00 -23.00
 0.00  0.00 -10.00 79.00 -30.00  0.00  0.00  0.00 -9.00  0.00
 0.00  0.00  0.00 -30.00 57.00 -7.00  0.00 -5.00  0.00 -20.00
 0.00  0.00  0.00  0.00 -7.00 47.00 -30.00  0.00  0.00 12.00
 0.00  0.00  0.00  0.00  0.00 -30.00 41.00  0.00  0.00 -7.00
 0.00  0.00  0.00  0.00 -5.00  0.00  0.00 27.00 -2.00  7.00
 0.00  0.00  0.00 -9.00  0.00  0.00  0.00 -2.00 29.00 10.00
Iter: 19
The solution vector is
-0.2895  0.3455 -0.7128 -0.2206 -0.4304  0.1543 -0.0578  0.2011  0.2902
The errors are
 0.0007  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
Checking result...
-15.01 27.00 -23.00 -0.00 -20.00 12.00 -7.00 7.00 10.00
+ task0

```

图 6: 超松弛 Gauss-Seidel 迭代法

附录

代码可在https://github.com/ZipWin/computational_physics/tree/master/assignments/assignment3找到.

Listing 5: linear_eqs_solver.f90

```

1 module utils
2   implicit none
3   type result
4     real(8) :: value
5     real(8) :: error
6   end type
7
8   contains
9   subroutine pivot_exchange(matrix, ndim1, ndim2)
10    implicit none
11    integer(8), intent(in) :: ndim1, ndim2
12    real(8), dimension(ndim1, ndim2) :: matrix
13
14    integer :: col, i
15    real(8), dimension(ndim2) :: tmp_row
16
17    do col = 1, ndim2
18      do i = col, ndim1
19        if (matrix(i, col) > matrix(col, col)) then
20          tmp_row = matrix(col, :)
21          matrix(col, :) = matrix(i, :)

```

```

22         matrix(i, :) = tmp_row
23     end if
24 end do
25 end do
26
27     return
28 end subroutine
29
30 end module
31
32
33 module linear_eqs_solver
34     use utils
35     implicit none
36     real(8), parameter :: EPSILON = 1e-3
37
38     contains
39     subroutine gauss_elimination(co_matrix, ndim, solution)
40         implicit none
41         integer(8), intent(in) :: ndim
42         real(8), dimension(ndim, ndim+1), intent(in) :: co_matrix
43         type(result), dimension(ndim), intent(out) :: solution
44
45         real(8), dimension(ndim, ndim+1) :: A
46         real(8), dimension(ndim) :: tmp_row, X
47         integer(8) :: t, i
48
49         A = co_matrix
50         do t = 1, ndim
51             ! pivot exchange
52             do i = t, ndim
53                 if (abs(A(i, t)) > abs(A(t, t))) then
54                     tmp_row = A(t, :)
55                     A(t, :) = A(i, :)
56                     A(i, :) = tmp_row
57                 end if
58             end do
59             ! Gauss elimination
60             do i = t+1, ndim
61                 A(i, :) = A(i, :) - A(i, t) / A(t, t) * A(t, :)
62             end do
63         end do
64
65         print "('The Gauss-eliminated extended coefficients matrix is')"
```

```

66     print "(10f7.2)", (A(i, :), i=1,9)
67
68     X = 0.0d0
69     do t = ndim, 1, -1
70         X(t) = (A(t, ndim+1) - dot_product(A(t, t+1:ndim), X(t+1:ndim))) / A(t, t)
71     end do
72
73     solution%value = X
74     solution%error = 0.0d0
75
76     return
77 end subroutine
78
79 subroutine doolittle(co_matrix, ndim, solution)
80     implicit none
81     integer(8), intent(in) :: ndim
82     real(8), dimension(ndim, ndim+1), intent(in) :: co_matrix
83     type(result), dimension(ndim), intent(out) :: solution
84
85     real(8), dimension(ndim, ndim+1) :: A
86     real(8), dimension(ndim, ndim) :: L, U
87     real(8), dimension(ndim) :: X, Y
88     integer(8) :: i, j, k
89
90     A = co_matrix
91     L = 0.0d0; U = 0.0d0;
92
93     call pivot_exchange(A, ndim, ndim+1)
94
95     do k = 1, ndim
96         do j = k, ndim
97             U(k, j) = A(k, j) - dot_product(L(k, :k-1), U(:k-1, j))
98         end do
99         L(k, k) = 1.0d0
100        do i = k+1, ndim
101            L(i, k) = (A(i, k) - dot_product(L(i, :k-1), U(:k-1, k))) / U(k, k)
102        end do
103    end do
104
105    do i = 1, ndim
106        Y(i) = A(i, ndim+1) - dot_product(L(i, :i-1), Y(:i-1))
107    end do
108    do i = ndim, 1, -1
109        X(i) = (Y(i) - dot_product(U(i, i+1:), X(i+1:))) / U(i, i)

```

```

110         end do
111
112         solution%value = X
113         solution%error = 0.0d0
114
115         return
116     end subroutine
117
118
119     subroutine jacobi_iteration(co_matrix, ndim, solution)
120         implicit none
121         integer(8), intent(in) :: ndim
122         real(8), dimension(ndim, ndim+1), intent(in) :: co_matrix
123         type(result), dimension(ndim), intent(out) :: solution
124
125         real(8), dimension(ndim, ndim+1) :: A
126         real(8), dimension(ndim) :: X, X_tmp, Error
127         integer(8) :: iter, i
128
129         A = co_matrix
130         call pivot_exchange(A, ndim, ndim+1)
131
132         X = 0.0d0
133         do iter = 1, 10000
134             do i = 1, ndim
135                 X_tmp(i) = -(dot_product(A(i, :ndim), X) - A(i, i) * X(i) - A(i, ndim+1)) / A(i
                    , i)
136             end do
137             Error = abs(X - X_tmp)
138             X = X_tmp
139
140             if (maxval(Error) < EPSILON) then
141                 exit
142             end if
143
144         end do
145         print *, iter
146         solution%value = X
147         solution%error = Error
148
149         return
150     end subroutine
151
152

```

```

153  subroutine gauss_seidel_iteration(co_matrix, ndim, solution, omega)
154      implicit none
155      integer(8), intent(in) :: ndim
156      real(8), optional, intent(in) :: omega
157      real(8), dimension(ndim, ndim+1), intent(in) :: co_matrix
158      type(result), dimension(ndim), intent(out) :: solution
159
160      real(8), dimension(ndim, ndim+1) :: A
161      real(8) :: lambda = 1.0d0
162      real(8), dimension(ndim) :: X, X_tmp, Error
163      integer(8) :: iter, i
164
165      A = co_matrix
166      call pivot_exchange(A, ndim, ndim+1)
167
168      if (present(omega)) then
169          lambda = omega
170      end if
171
172      X = 0.0d0
173      do iter = 1, 10000
174          X_tmp = X
175          do i = 1, ndim
176              X(i) = -(dot_product(A(i, :ndim), X) - A(i, i) * X(i) - A(i, ndim+1)) / A(i, i)
177          end do
178          X = (1 - lambda) * X_tmp + lambda * X
179          Error = abs(X - X_tmp)
180
181          if (maxval(Error) < EPSILON) then
182              exit
183          end if
184
185      end do
186      print "('Iter: ',i4)", iter
187      solution%value = X
188      solution%error = Error
189
190      return
191  end subroutine
192
193  end module
194
195
196  program main

```

```

197     use utils
198     use linear_eqs_solver
199     implicit none
200     integer(8), parameter :: ndim = 9
201     real(8), dimension(ndim, ndim+1) :: A
202     type(result), dimension(ndim) :: solution
203     integer(8) :: i
204
205     A(1, :9) = (/31.0d0, -13.0d0, 0.0d0, 0.0d0, 0.0d0, -10.0d0, 0.0d0, 0.0d0, 0.0d0/)
206     A(2, :9) = (/ -13.0d0, 35.0d0, -9.0d0, 0.0d0, -11.0d0, 0.0d0, 0.0d0, 0.0d0, 0.0d0/)
207     A(3, :9) = (/0.0d0, -9.0d0, 31.0d0, -10.0d0, 0.0d0, 0.0d0, 0.0d0, 0.0d0, 0.0d0/)
208     A(4, :9) = (/0.0d0, 0.0d0, -10.0d0, 79.0d0, -30.0d0, 0.0d0, 0.0d0, 0.0d0, -9.0d0/)
209     A(5, :9) = (/0.0d0, 0.0d0, 0.0d0, -30.0d0, 57.0d0, -7.0d0, 0.0d0, -5.0d0, 0.0d0/)
210     A(6, :9) = (/0.0d0, 0.0d0, 0.0d0, 0.0d0, -7.0d0, 47.0d0, -30.0d0, 0.0d0, 0.0d0/)
211     A(7, :9) = (/0.0d0, 0.0d0, 0.0d0, 0.0d0, 0.0d0, -30.0d0, 41.0d0, 0.0d0, 0.0d0/)
212     A(8, :9) = (/0.0d0, 0.0d0, 0.0d0, 0.0d0, -5.0d0, 0.0d0, 0.0d0, 27.0d0, -2.0d0/)
213     A(9, :9) = (/0.0d0, 0.0d0, 0.0d0, -9.0d0, 0.0d0, 0.0d0, 0.0d0, -2.0d0, 29.0d0/)
214     A(:, 10) = (/ -15.0d0, 27.0d0, -23.0d0, 0.0d0, -20.0d0, 12.0d0, -7.0d0, 7.0d0, 10.0d0/)
215
216     print "('The extented coefficients matrix is')"

```