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

在解决实际问题的时候，常常会遇到求解线性方程组的问题，比如使用有限差分法解偏微分方程. 往往线性方程组的个数会达到上万个，因此研究高效的算法很有必要. 这次作业主要使用 Gauss 消元法、Doolittle 分解法、Gauss-Seidel 迭代法和超松弛迭代法来解线性方程组.

2 问题描述

问题 1. *分别使用 Gauss 消元法、Doolittle 分解法、超松弛 Gauss-Seidel 迭代法解以下方程组*

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

3 程序实现

为了方便，使用拓展矩阵来储存线性方程组的全部信息. 由于上面提到的所有方法都不希望对角元出现 0，因此在计算之前，先对矩阵做列主元变换. 代码见 Listing 1.

Listing 1: **列主元变换**

```
subroutine pivot_exchange(matrix, ndim1, ndim2)
    implicit none
    integer(8), intent(in) :: ndim1, ndim2
    real(8), dimension(ndim1, ndim2) :: matrix

    integer :: col, i
    real(8), dimension(ndim2) :: tmp_row

    do col = 1, ndim2
        do i = col, ndim1
            if (matrix(i, col) > matrix(col, col)) then
                tmp_row = matrix(col, :)
                matrix(col, :) = matrix(i, :)
                matrix(i, :) = tmp_row
            end if
        end do
```

```
    end do

    return
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，即在每次消某一列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.

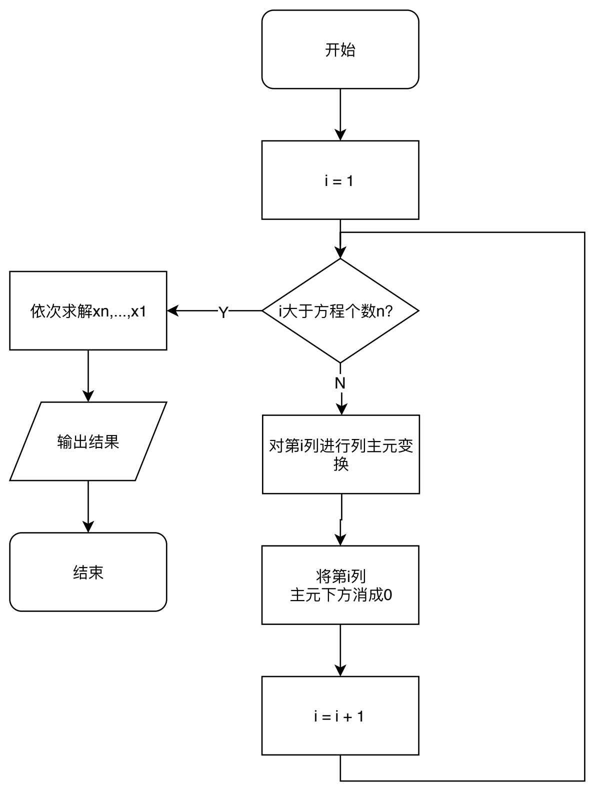

图 1: Gauss 消元法流程图

Listing 2: **Gauss 消元法**

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

```fortran
    type(result), dimension(ndim), intent(out) :: solution

    real(8), dimension(ndim, ndim+1) :: A
    real(8), dimension(ndim) :: tmp_row, X
    integer(8) :: t, i

    A = co_matrix
    do t = 1, ndim
        ! pivot exchange
        do i = t, ndim
            if (abs(A(i, t)) > abs(A(t, t))) then
                tmp_row = A(t, :)
                A(t, :) = A(i, :)
                A(i, :) = tmp_row
            end if
        end do
        ! Gauss elimination
        do i = t+1, ndim
            A(i, :) = A(i, :) - A(i, t) / A(t, t) * A(t, :)
        end do
    end do

    print "('The Gauss-eliminated extented coefficients matrix is')"
    print "(10f7.2)", (A(i, :), i=1,9)

    X = 0.0d0
    do t = ndim, 1, -1
        X(t) = (A(t, ndim+1) - dot_product(A(t, t+1:ndim), X(t+1:ndim))) / A(t, t)
    end do

    solution%value = X
    solution%error = 0.0d0

    return
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.

$$\boldsymbol{A} = \boldsymbol{L}\boldsymbol{U}$$

$\boldsymbol{L}$ 和 $\boldsymbol{U}$ 各分量的值可以由下式得到

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

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

方程最终的解分两步得到

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

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

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.

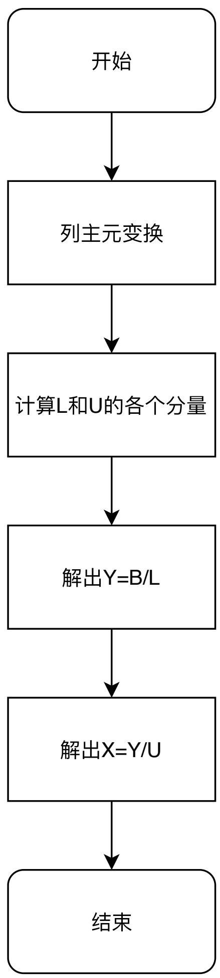

图 2: Doolittle 分解法流程图

Listing 3: **Doolittle 分解法**

```
subroutine doolittle(co_matrix, ndim, solution)
    implicit none
    integer(8), intent(in) :: ndim
    real(8), dimension(ndim, ndim+1), intent(in) :: co_matrix
    type(result), dimension(ndim), intent(out) :: solution

    real(8), dimension(ndim, ndim+1) :: A
    real(8), dimension(ndim, ndim) :: L, U
    real(8), dimension(ndim) :: X, Y
    integer(8) :: i, j, k

    A = co_matrix
    L = 0.0d0; U = 0.0d0;

    call pivot_exchange(A, ndim, ndim+1)

    do k = 1, ndim
        do j = k, ndim
            U(k, j) = A(k, j) - dot_product(L(k, :k-1), U(:k-1, j))
        end do
        L(k, k) = 1.0d0
        do i = k+1, ndim
            L(i, k) = (A(i, k) - dot_product(L(i, :k-1), U(:k-1, k))) / U(k, k)
        end do
    end do

    do i = 1, ndim
        Y(i) = A(i, ndim+1) - dot_product(L(i, :i-1), Y(:i-1))
    end do
    do i = ndim, 1, -1
        X(i) = (Y(i) - dot_product(U(i, i+1:), X(i+1:))) / U(i, i)
    end do

    solution%value = X
    solution%error = 0.0d0

    return
end subroutine
```

3.3 Gauss-Seidel 迭代法

Gauss-Seidel 迭代法的子程序其中的参数 omega 是可选参数. 如果不输入 omega，那么默认 omega 的值为 1，即 Gauss-Seidel 迭代；如果输入 omega，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.

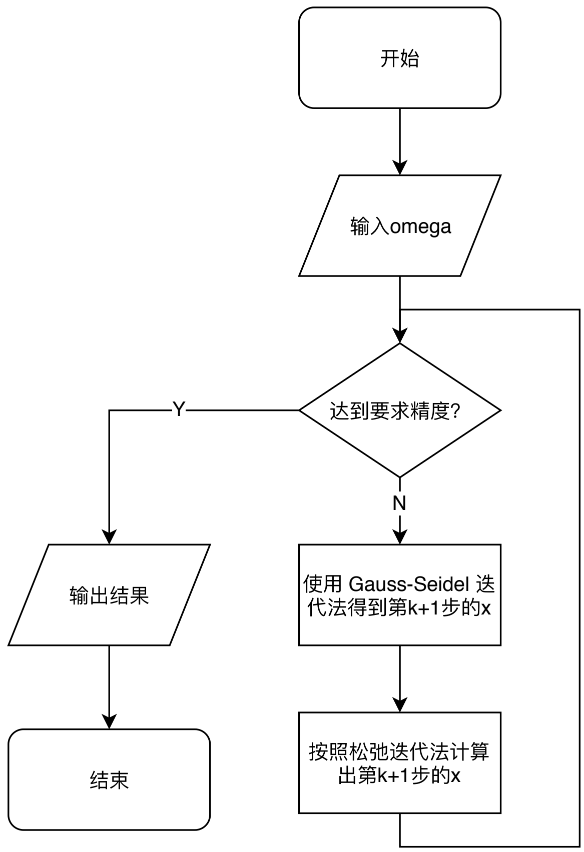

图 3: 使用松弛迭代的 Gauss-Seidel 迭代法流程图

Listing 4: **Gauss-Seidel 迭代法**

```
subroutine gauss_seidel_iteration(co_matrix, ndim, solution, omega)
   implicit none
   integer(8), intent(in) :: ndim
   real(8), optional, intent(in) :: omega
   real(8), dimension(ndim, ndim+1), intent(in) :: co_matrix
   type(result), dimension(ndim), intent(out) :: solution

   real(8), dimension(ndim, ndim+1) :: A
   real(8) :: lambda = 1.0d0
   real(8), dimension(ndim) :: X, X_tmp, Error
```

```
    integer(8) :: iter, i

    A = co_matrix
    call pivot_exchange(A, ndim, ndim+1)

    if (present(omega)) then
        lambda = omega
    end if

    X = 0.0d0
    do iter = 1, 10000
        X_tmp = X
        do i = 1, ndim
            X(i) = -(dot_product(A(i, :ndim), X) - A(i, i) * X(i) - A(i, ndim+1)) / A(i, i)
        end do
        X = (1 - lambda) * X_tmp + lambda * X
        Error = abs(X - X_tmp)

        if (maxval(Error) < EPSILON) then
            exit
        end if

    end do
    print "('Iter: ',i4)", iter
    solution%value = X
    solution%error = Error

    return
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.

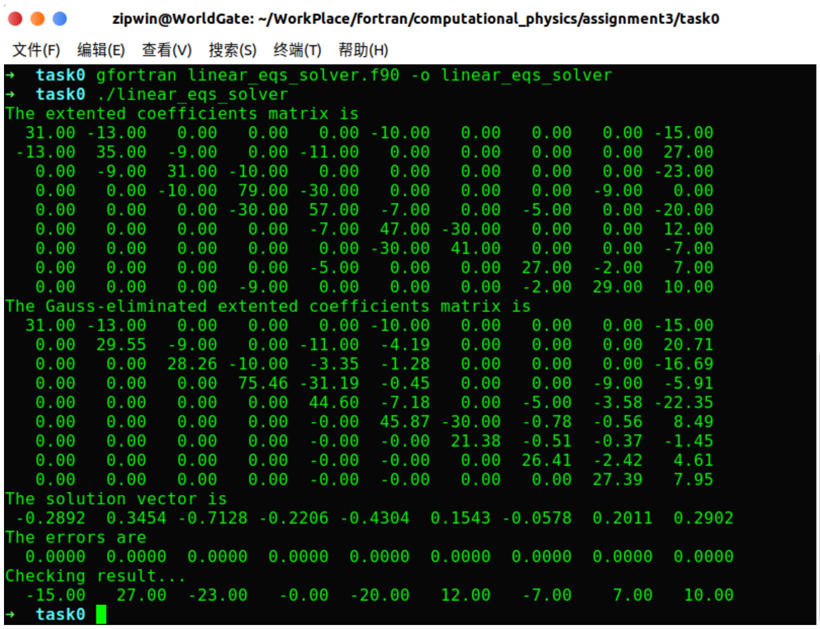

图 4: Gauss 消元法

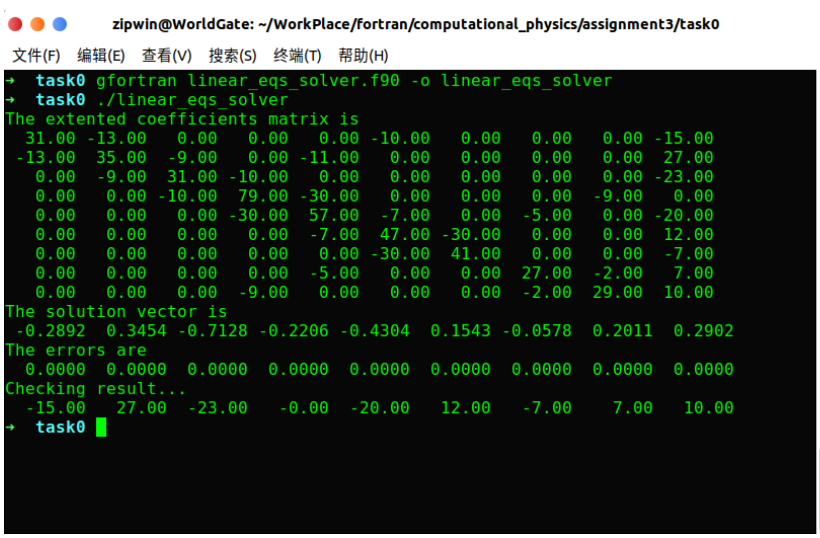

图 5: Doolittle 分解法

4.3 超松弛 Gauss-Seidel 迭代法

设置 omega 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

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

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.

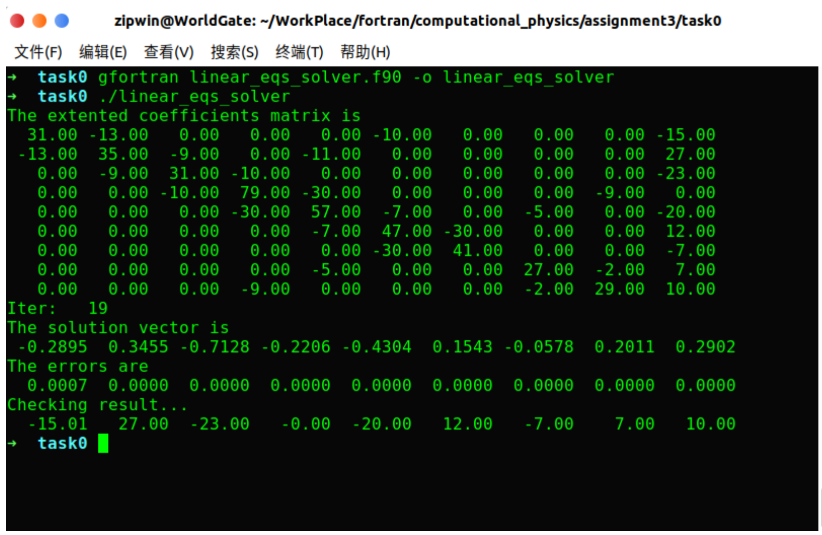

图 6: 超松弛 Gauss-Seidel 迭代法

附录

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

Listing 5: **linear__eqs__solver.f90**

```
module utils
    implicit none
    type result
        real(8) :: value
        real(8) :: error
    end type

    contains
    subroutine pivot_exchange(matrix, ndim1, ndim2)
        implicit none
        integer(8), intent(in) :: ndim1, ndim2
        real(8), dimension(ndim1, ndim2) :: matrix

        integer :: col, i
        real(8), dimension(ndim2) :: tmp_row

        do col = 1, ndim2
            do i = col, ndim1
                if (matrix(i, col) > matrix(col, col)) then
                    tmp_row = matrix(col, :)
                    matrix(col, :) = matrix(i, :)
```

```
                matrix(i, :) = tmp_row
            end if
        end do
    end do

    return
end subroutine

end module


module linear_eqs_solver
  use utils
  implicit none
  real(8), parameter :: EPSILON = 1e-3

  contains
  subroutine gauss_elimination(co_matrix, ndim, solution)
    implicit none
    integer(8), intent(in) :: ndim
    real(8), dimension(ndim, ndim+1), intent(in) :: co_matrix
    type(result), dimension(ndim), intent(out) :: solution

    real(8), dimension(ndim, ndim+1) :: A
    real(8), dimension(ndim) :: tmp_row, X
    integer(8) :: t, i

    A = co_matrix
    do t = 1, ndim
      ! pivot exchange
      do i = t, ndim
        if (abs(A(i, t)) > abs(A(t, t))) then
          tmp_row = A(t, :)
          A(t, :) = A(i, :)
          A(i, :) = tmp_row
        end if
      end do
      ! Gauss elimination
      do i = t+1, ndim
        A(i, :) = A(i, :) - A(i, t) / A(t, t) * A(t, :)
      end do
    end do

    print "('The Gauss-eliminated extented coefficients matrix is')"
```

```fortran
    print "(10f7.2)", (A(i, :), i=1,9)

    X = 0.0d0
    do t = ndim, 1, -1
        X(t) = (A(t, ndim+1) - dot_product(A(t, t+1:ndim), X(t+1:ndim))) / A(t, t)
    end do

    solution%value = X
    solution%error = 0.0d0

    return
end subroutine

subroutine doolittle(co_matrix, ndim, solution)
    implicit none
    integer(8), intent(in) :: ndim
    real(8), dimension(ndim, ndim+1), intent(in) :: co_matrix
    type(result), dimension(ndim), intent(out) :: solution

    real(8), dimension(ndim, ndim+1) :: A
    real(8), dimension(ndim, ndim) :: L, U
    real(8), dimension(ndim) :: X, Y
    integer(8) :: i, j, k

    A = co_matrix
    L = 0.0d0; U = 0.0d0;

    call pivot_exchange(A, ndim, ndim+1)

    do k = 1, ndim
        do j = k, ndim
            U(k, j) = A(k, j) - dot_product(L(k, :k-1), U(:k-1, j))
        end do
        L(k, k) = 1.0d0
        do i = k+1, ndim
            L(i, k) = (A(i, k) - dot_product(L(i, :k-1), U(:k-1, k))) / U(k, k)
        end do
    end do

    do i = 1, ndim
        Y(i) = A(i, ndim+1) - dot_product(L(i, :i-1), Y(:i-1))
    end do
    do i = ndim, 1, -1
        X(i) = (Y(i) - dot_product(U(i, i+1:), X(i+1:))) / U(i, i)
```

```
        end do

        solution%value = X
        solution%error = 0.0d0

        return
    end subroutine


    subroutine jacobi_iteration(co_matrix, ndim, solution)
        implicit none
        integer(8), intent(in) :: ndim
        real(8), dimension(ndim, ndim+1), intent(in) :: co_matrix
        type(result), dimension(ndim), intent(out) :: solution

        real(8), dimension(ndim, ndim+1) :: A
        real(8), dimension(ndim) :: X, X_tmp, Error
        integer(8) :: iter, i

        A = co_matrix
        call pivot_exchange(A, ndim, ndim+1)

        X = 0.0d0
        do iter = 1, 10000
            do i = 1, ndim
                X_tmp(i) = -(dot_product(A(i, :ndim), X) - A(i, i) * X(i) - A(i, ndim+1)) / A(i
                    , i)
            end do
            Error = abs(X - X_tmp)
            X = X_tmp

            if (maxval(Error) < EPSILON) then
                exit
            end if

        end do
        print *, iter
        solution%value = X
        solution%error = Error

        return
    end subroutine


```

```
    subroutine gauss_seidel_iteration(co_matrix, ndim, solution, omega)
        implicit none
        integer(8), intent(in) :: ndim
        real(8), optional, intent(in) :: omega
        real(8), dimension(ndim, ndim+1), intent(in) :: co_matrix
        type(result), dimension(ndim), intent(out) :: solution

        real(8), dimension(ndim, ndim+1) :: A
        real(8) :: lambda = 1.0d0
        real(8), dimension(ndim) :: X, X_tmp, Error
        integer(8) :: iter, i

        A = co_matrix
        call pivot_exchange(A, ndim, ndim+1)

        if (present(omega)) then
            lambda = omega
        end if

        X = 0.0d0
        do iter = 1, 10000
            X_tmp = X
            do i = 1, ndim
                X(i) = -(dot_product(A(i, :ndim), X) - A(i, i) * X(i) - A(i, ndim+1)) / A(i, i)
            end do
            X = (1 - lambda) * X_tmp + lambda * X
            Error = abs(X - X_tmp)

            if (maxval(Error) < EPSILON) then
                exit
            end if

        end do
        print "('Iter: ',i4)", iter
        solution%value = X
        solution%error = Error

        return
    end subroutine

end module


program main
```

```
    use utils
    use linear_eqs_solver
    implicit none
    integer(8), parameter :: ndim = 9
    real(8), dimension(ndim, ndim+1) :: A
    type(result), dimension(ndim) :: solution
    integer(8) :: i

    A(1, :9) = (/31.0d0, -13.0d0, 0.0d0, 0.0d0, 0.0d0, -10.0d0, 0.0d0, 0.0d0, 0.0d0/)
    A(2, :9) = (/-13.0d0, 35.0d0, -9.0d0, 0.0d0, -11.0d0, 0.0d0, 0.0d0, 0.0d0, 0.0d0/)
    A(3, :9) = (/0.0d0, -9.0d0, 31.0d0, -10.0d0, 0.0d0, 0.0d0, 0.0d0, 0.0d0, 0.0d0/)
    A(4, :9) = (/0.0d0, 0.0d0, -10.0d0, 79.0d0, -30.0d0, 0.0d0, 0.0d0, 0.0d0, -9.0d0/)
    A(5, :9) = (/0.0d0, 0.0d0, 0.0d0, -30.0d0, 57.0d0, -7.0d0, 0.0d0, -5.0d0, 0.0d0/)
    A(6, :9) = (/0.0d0, 0.0d0, 0.0d0, 0.0d0, -7.0d0, 47.0d0, -30.0d0, 0.0d0, 0.0d0/)
    A(7, :9) = (/0.0d0, 0.0d0, 0.0d0, 0.0d0, 0.0d0, -30.0d0, 41.0d0, 0.0d0, 0.0d0/)
    A(8, :9) = (/0.0d0, 0.0d0, 0.0d0, 0.0d0, -5.0d0, 0.0d0, 0.0d0, 27.0d0, -2.0d0/)
    A(9, :9) = (/0.0d0, 0.0d0, 0.0d0, -9.0d0, 0.0d0, 0.0d0, 0.0d0, -2.0d0, 29.0d0/)
    A(:, 10) = (/-15.0d0, 27.0d0, -23.0d0, 0.0d0, -20.0d0, 12.0d0, -7.0d0, 7.0d0, 10.0d0/)

    print "('The extented coefficients matrix is')"
    print "(10f7.2)", (A(i, :), i=1,9)

    call gauss_seidel_iteration(A, ndim, solution)

    print "('The solution vector is')"
    print "(9f8.4)", (solution(i)%value, i=1,9)
    print "('The errors are')"
    print "(9f8.4)", (solution(i)%error, i=1,9)

    print "('Checking result...')"
    print "(9f8.2)", (dot_product(A(i, :9), solution%value), i=1,9)

end program
```