Homework Chapter 02

Linear_solving,刘玖阳,应用物理1301,U201310209

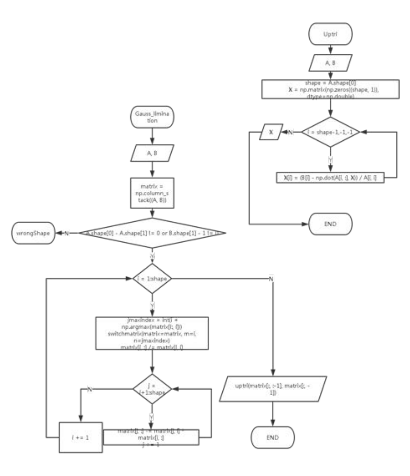
Question

Write a program to solve the following linear systems by *Gauss Elimination Method*, *Doolittle Decomposition Method*, and *Overrelaxation*.

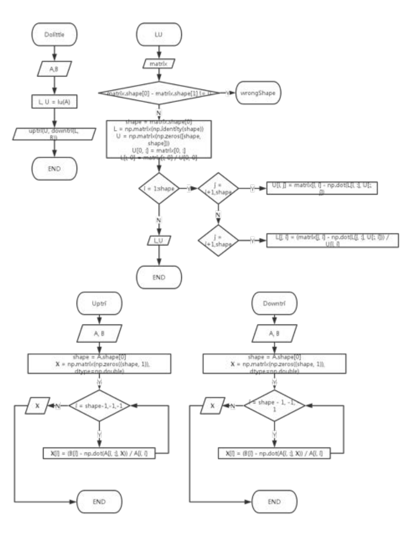
$$Ax = \begin{bmatrix} -15 \\ 27 \\ -23 \\ 0 \\ -20 \\ 12 \\ -7 \\ 7 \\ 10 \end{bmatrix} \quad A = \begin{bmatrix} 31 & -13 & 0 & 0 & 0 & -10 & 0 & 0 & 0 \\ -13 & 35 & -9 & 0 & -11 & 0 & 0 & 0 & 0 \\ 0 & -9 & 31 & -10 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -10 & 79 & -30 & 0 & 0 & 0 & -9 \\ 0 & 0 & 0 & -30 &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

Used function and algorithm

1. Gausse_limination
   i. Transfrom A to upper triangular matrix
   ii. Solve the triangluar matrix

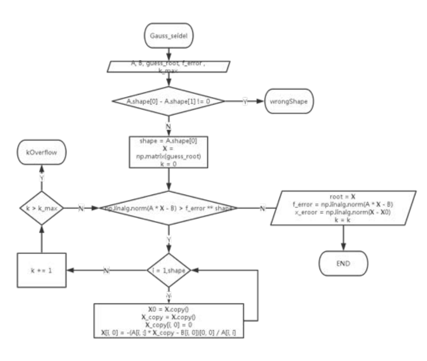


2. Doolittle
   i. Using LU decomposition to A
   ii. Using triangular solver to get X



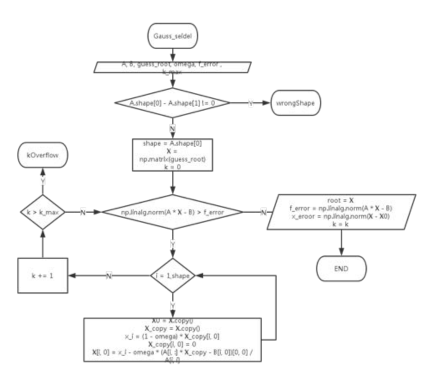
3. Gauss_seidel

$$x_i^{k+1} = \frac{-1}{a_{ii}} (\sum_{j=1}^{i-1} a_{ij} x_j^{k+1} + \sum_{j=i+1}^{n} a_{ij} x_j^k - b_i)$$



4. Successive_relaxation

$$x_i^{k+1} = (1-\omega) x_n^k + \omega \frac{-1}{a_{ii}} (\sum_{j=1}^{i-1} a_{ij} x_j^{k+1} + \sum_{j=i+1}^{n} a_{ij} x_j^k - b_i)$$

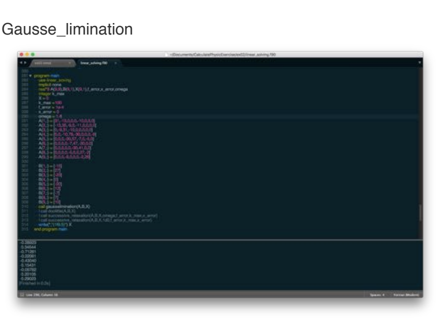


Source Code

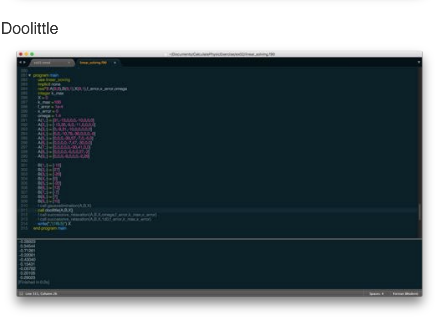
```
module linalg
!----------
!   author:
!       Sequencer
!   description:
!       计算物理线性代数辅助模块
!       MIT协议
!   contains:
!       norm(vector)
!       zeros(n,m)
!       identity(n)
!       diag(matrix)
!       column_stack(A,B)
!       row_stack(A,B)
!       switchmatrix(matrix,m,n)
!   exception:
!       wrongShape
!----------
contains

function norm(vector)
    !返回一个向量的长度
    implicit none
    real*8::vector(:,:)
    real*8::norm
    integer::i
    do while (i .le. size(vector))
        norm = norm + vector(i,1)**2
        i = i+1
    enddo
end function norm

function zeros(n,m)
    !返回一个n,m的零矩阵
    implicit none
    integer n,m
    real*8 zeros(n,m)
    zeros = 0d0
end function zeros

function identity(n)
    !返回一个n阶单位矩阵
    !返回一个n,m的零矩阵
    implicit none
    integer n,m,i
    real*8 identity(n,n)
    identity = 0d0
    forall(i = 1:n) identity(i,i) = 1d0
end function identity

function diag(matrix)
    implicit none
    real*8 :: matrix(:,:)
    real*8 :: diag(size(matrix,dim = 1),size(matrix,dim = 1))
    integer :: shape,i
    shape = size(matrix,dim = 1)
    !形状判断
    if (size(matrix,dim = 1)-size(matrix,dim = 2) .ne. 0) then
        write(*,*) 'wrongShape'
    endif
    diag = matrix
    forall(i = 1:shape) diag(i,i) = 0d0
    diag = matrix - diag
end function diag

function column_stack(A,B)
    implicit none
    real*8,intent(in):: A(:,:),B(:,:)
    real*8 :: column_stack(size(A,dim =1),size(A,dim=2)+size(B,dim =2))
    column_stack = 0
    column_stack(:,1:size(A,dim =2)) = A
    column_stack(:,size(A,dim =2)+1:) = B
end function column_stack

function row_stack(A,B)
    implicit none
    real*8,intent(in):: A(:,:),B(:,:)
    real*8 :: row_stack(size(A,dim=1)+size(B,dim =1),size(A,dim=2))
    row_stack = 0
    row_stack(1:size(A,dim = 1),:) = A
    row_stack(size(A,dim = 1)+1:,:) = B
end function row_stack


subroutine switchmatrix(matrix,m,n)
    !交换矩阵中的某两行
    real*8,intent(inout):: matrix(:,:)
    real*8::temp_mat(1,size(matrix,dim = 2))
    integer shape
    shape = size(matrix,dim = 1)
    temp_mat(1,:) = matrix(m,:)
    matrix(m,:) = matrix(n,:)
    matrix(n,:) = temp_mat(1,:)
end subroutine switchmatrix
end module linalg

module linear_soving
!----------
!   author:
!       Sequencer
!   description:
!       计算物理非线型方程寻根模块(HUST PHY 2013 第二次作业)
!       MIT协议
!   contains:
!       downtri(A,B,X)
!       uptri(A,B,X)
!       gausselimination(A,B,X)
!       lu(matrix,L,U)
!       doolittle(A,B,X)
!       jacobi(A,B,X,f_error,k_max,x_error)
!       successive_relaxation(A,B,X,omega,f_error,k_max,x_error)
!   exception:
!       kOverflow
!       fDivergence
!       wrongShape
!----------
use linalg

contains

subroutine downtri(A,B,X)
    implicit none
    real*8,intent(in) :: A(:,:),B(:,:)
    real*8,intent(out) :: X(size(A,dim = 1),1)
    integer :: shape
    integer :: i
    shape = size(A,dim = 1)
    X = zeros(shape,1)
    i = 1
    do while (i.le.shape)
        X(i,1) = (B(i,1) - dot_product(A(i,:),X(:,1)))/A(i,i)
        i = i+1
    enddo
end subroutine downtri

subroutine uptri(A,B,X)
    implicit none
    real*8,intent(in) :: A(:,:),B(:,:)
    real*8,intent(out) :: X(size(A,dim = 1),1)
    integer :: shape
    integer :: i
    shape = size(A,dim = 1)
    X = zeros(shape,1)
    i = shape
    do while (i.ge.1)
        X(i,1) = (B(i,1) - dot_product(A(i,:),X(:,1)))/A(i,i)
        i = i-1
    enddo
end subroutine uptri

subroutine gausselimination(A,B,X)
    implicit none
    real*8 :: A(:,:),B(:,:)
    real*8,intent(out) :: X(size(A,dim = 1),1)
    real*8 :: matrix(size(A,dim = 1),size(A,dim = 1)+1)
    integer :: i,j,shape,jmaxindex
    if (size(A,dim =1).ne.size(A,dim = 2)) then
        write(*,*) "wrongShape"
    endif
    matrix = column_stack(A,B)
    shape = size(A,dim = 1)
    X = 0
    i = 1
    jmaxindex = 0
    do while (i.le.shape)
        jmaxindex = i-1+ sum(maxloc(matrix(i:shape,i))) ! 用sum提取array中的唯一一个元素
        call switchmatrix(matrix,i,jmaxindex)
        matrix(i,:) = matrix(i,:)/matrix(i,i)
        j = i+1
        do while (j.le.shape)
            matrix(j,:) = matrix(j,:)-matrix(j,i)*matrix(i,:)
            j = j+1
        enddo
        i = i + 1
    enddo
    A(1:shape,1:shape)= matrix(:,1:shape)
    B(1:shape,1)= matrix(:, shape+1)
    call uptri(A,B,X)
end subroutine gausselimination

subroutine lu(matrix,L,U)
    implicit none
    real*8,intent(in) :: matrix(:,:)
    real*8,intent(inout) :: L(size(matrix,dim = 1),size(matrix,dim = 2)),U(size(matrix,dim = 1),size(matrix,d
    integer :: shape,i,j
    shape = size(matrix,dim = 1)
    if (size(matrix,dim =1).ne.size(matrix,dim = 2)) then
        write(*,*) "wrongShape"
    endif
    L = identity(shape)
    U = zeros(shape,shape)
    U(1,:) = matrix(1,:)
    L(:,1) =matrix(:,1) / U(1,1)
    i = 2
    do while (i .le. shape)
        j = i
        do while (j.le.shape)
            U(i,j) = matrix(i,i) - dot_product(L(i,:),U(:,j))
            j = j+1
        enddo
        j = i+1
        do while (j.le.shape)
            L(j,i) = (matrix(j,i)-dot_product(L(j,:),U(:,i)))/U(i,i)
            j =j+1
        enddo
        i = i + 1
    enddo
end subroutine lu

subroutine doolittle(A,B,X)
    implicit none
    real*8, intent(in) :: A(:,:),B(:,:)
    real*8, intent(out) :: X(size(B,dim = 1),1)
    real*8 ::L(size(A,dim = 1),size(A,dim = 2)),U(size(A,dim = 1),size(A,dim = 2)),Y(size(B,dim = 1),1)
    call lu(A,L,U)
    call downtri(L,B,Y)
    call uptri(U,Y,X)
end subroutine doolittle

subroutine jacobi(A,B,X,f_error,k_max,x_error)
    implicit none
    real*8, intent(in) :: A(:,:),B(:,:)
    real*8, intent(inout) :: X(:,:),x_error
    real*8 :: Di(size(A,dim = 1),size(A,dim = 2)),G(size(A,dim = 1),size(A,dim = 2)),g0(size(A,dim = 1),1)
    real*8 :: X0(size(X,dim =1),size(X,dim =2))
    real*8 :: f_error
    integer :: k_max,k,shape
    shape = size(A,dim = 1)
    if (size(A,dim =1).ne.size(A,dim = 2)) then
        write(*,*) "wrongShape"
    endif
    G = 1
    Di = diag(G/A)
    G = identity(shape) - matmul(Di,A)
    g0 = matmul(Di,B)
    X = matmul(G,X) + g0
    k = 1
    do while (norm(matmul(A,X)-B)>f_error)
        X0 = X
        X = matmul(G,X)+g0
        k = k+1
        if (k>k_max) then
            write(*,*) "kOverflow"
            call abort()
        endif
    enddo
end subroutine jacobi

subroutine successive_relaxation(A,B,X,omega,f_error,k_max,x_error)
    implicit none
    real*8, intent(in) :: A(:,:),B(:,:),omega,f_error,x_error
    real*8, intent(inout) :: X(:,:)
    real*8 :: X0(size(X,dim =1),size(X,dim =2)),X_copy(size(X,dim =1),size(X,dim =2)),X_i
    integer k_max,shape,k,i

    shape = size(A,dim = 1)
    if (size(A,dim =1).ne.size(A,dim = 2)) then
        write(*,*) "wrongShape"
    endif
    k = 0
    do while (norm(matmul(A,X)-B).gt.f_error)
        i = 1
        do while (i.le.shape)
            X0 = X
            X_copy = X
            x_i = (1-omega)*X_copy(i,1)
            X_copy(i,1) = 0
            X(i,1) = x_i - omega*(sum(matmul(A(i,:),X_copy)-B(i,1))) / A(i,i)
            i = i+1
        enddo
        k = k+1
        if (k>k_max) then
            write(*,*) "kOverflow"
            call abort()
        endif
    enddo
end subroutine

end module linear_soving

program main
    use linear_soving
    implicit none
    real*8 A(9,9),B(9,1),X(9,1),f_error,x_error,omega
    integer k_max
    X = 0
    k_max =100
    f_error = 1e-4
    x_error = 0
    omega = 1.4
    A(1,:) = [31,-13,0,0,0,-10,0,0,0]
    A(2,:) = [-13,35,-9,0,-11,0,0,0,0]
    A(3,:) = [0,-9,31,-10,0,0,0,0,0]
    A(4,:) = [0,0,-10,79,-30,0,0,0,-9]
    A(5,:) = [0,0,0,-30,57,-7,0,-5,0]
    A(6,:) = [0,0,0,0,-7,47,-30,0,0]
    A(7,:) = [0,0,0,0,0,-30,41,0,0]
    A(8,:) = [0,0,0,0,-5,0,0,27,-2]
    A(9,:) = [0,0,0,-9,0,0,0,-2,29]

    B(1,:) = [-15]
    B(2,:) = [27]
    B(3,:) = [-23]
    B(4,:) = [0]
    B(5,:) = [-20]
    B(6,:) = [12]
    B(7,:) = [-7]
    B(8,:) = [7]
    B(9,:) = [10]
    ! call gausselimination(A,B,X)
    ! call doolittle(A,B,X)
    ! call successive_relaxation(A,B,X,omega,f_error,k_max,x_error)
    write(*,"(1f9.5)") X
end program main
```

Screenshot

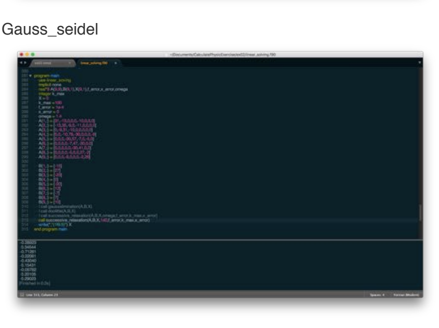
1. Gausse_limination



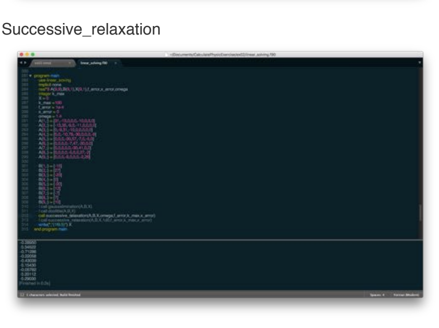
2. Doolittle



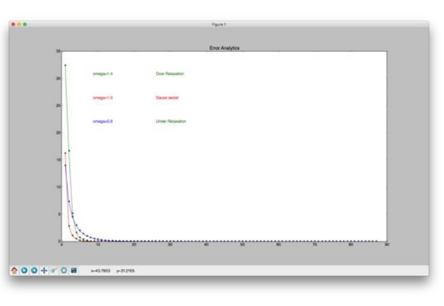
3. Gauss_seidel



4. Successive_relaxation



Error analytics



Using `norm(A*X-B)` as the error function, set error to 1×10^{-9}.
As we can see, different ω has a different speed. When $\omega = 1$ is Gauss Seidel Method.

> The origin data has upload to Github