```fortran
module  tdma
    contains
    !onetdma code for solving one line using TDMA
    subroutine onetdma(n,a,b,c,d,line)
        implicit none
        integer, intent (in) :: n
        double precision, dimension (1:n), intent (inout) ::  line
        double precision, dimension (1:n), intent (in) :: a,b,c,d
        double precision, dimension (1:n) :: p,q
        integer :: i,flag,rec
        !flag to identify fixed point handling
        flag = 0
        do i = 2,n-1
            if((a(i)==1).and.(b(i)==0).and.(c(i)==0)) then
                flag = 1
                rec = i
            end if
        end do
        if (flag==0) then
        !initialize boundaries
        p(1) = b(1)/a(1)
        q(1) = d(1)/a(1)
        !compute p and q
        do i = 2,n
            p(i) = b(i)/(a(i)-c(i)*p(i-1))
            q(i) = ((c(i)*q(i-1))+d(i))/(a(i)-(c(i)*p(i-1)))
        end do
        !calculate line values
        do i = n-1,2,-1
            line(i) = p(i)*line(i+1)+q(i)
        end do
        end if
        !if flag found, break a line into two and solve for individual lines
        if (flag==1) then
            call onetdmarec(rec,a(1:rec),b(1:rec),c(1:rec),d(1:rec),line(1:rec))
```

```fortran
                call onetdmarec(n-rec+1,a(rec:n),b(rec:n),c(rec:n),d(rec:n),line(rec:n))
            end if

    end subroutine onetdma
    !first recursion of onetdma required to handle fixed points
    subroutine onetdmarec(n,a,b,c,d,line)
        implicit none
        integer, intent (in) :: n
        double precision, dimension (1:n), intent (inout) ::  line
        double precision, dimension (1:n), intent (in) :: a,b,c,d
        double precision, dimension (1:n) :: p,q
        integer :: i
        !initialize boundaries
        p(1) = b(1)/a(1)
        q(1) = d(1)/a(1)
        !compute p and q
        do i = 2,n
            p(i) = b(i)/(a(i)-c(i)*p(i-1))
            q(i) = ((c(i)*q(i-1))+d(i))/(a(i)-(c(i)*p(i-1)))
        end do
        !calculate line values
        do i = n-1,2,-1
            line(i) = p(i)*line(i+1)+q(i)
        end do
    end subroutine onetdmarec
    !code for onedtdma, constructs coefficients itself and solves for the line
    subroutine onedtdma(n,mat,gmae,gmaw,Del_x,delxe,delxw,sc,sp)
        implicit none
        !delaratios
        integer, intent (in) :: n
        double precision, dimension (1:n), intent (in) :: gmae,gmaw,Del_x,delxe,delxw,sc,sp
        double precision, dimension (1:n), intent (inout) :: mat
        double precision, dimension (1:n) :: mat_dash
        double precision, dimension (1:n) :: lane, a,b,c,d
        double precision :: ae, aw, ap, delv,b_, eps
        integer :: i,c_
        c_ = 0
        do while (.true.)
            c_ = c_ + 1
            print*,c_
            eps = 0.0
            !backup matrix
            mat_dash = mat
            lane = mat_dash
            !setting boundaries
            a(1) = 1.0
            b(1) = 0.0
            c(1) = 0.0
            d(1) = mat_dash(1)
            a(n) = 1.0
            b(n) = 0.0
            c(n) = 0.0
            d(n) = mat_dash(n)
            !computing coefficients for inner points
            do i = 2,n-1
                ae = gmae(i)/delxe(i)
                aw = gmaw(i)/delxw(i)
                delv = Del_x(i)
                b_ = sc(i)*delv
                ap = ae+aw-(sp(i)*delv)
                a(i) = ap
                b(i) = ae
                c(i) = aw
                d(i) = b_
            end do
            call onetdma(n,a,b,c,d,lane)
            mat = lane
            !error using eucleadian distance
            eps = sqrt(sum((mat-mat_dash)**2))
            if (eps<1e-10) exit
        end do
    end subroutine onedtdma
    !code for extending onedtdma to two dimensions, builds all the coefficients itself, Dirichlet BC
    subroutine twodtdma(n,m,mat,gmae,gmaw,gman,gmas,Del_x,Del_y,delxe,delxw,delyn,delys,sc,sp,sweep)
        implicit none
        !declarations
        integer, intent (in) :: n,m
        character(2), intent (in) :: sweep
        double precision, dimension (1:n,1:m), intent (in) :: gmae,gmaw,gman,gmas,Del_x,Del_y,delxe,delxw,delyn,delys,sc,sp
        double precision, dimension (1:n,1:m), intent (inout) :: mat
        double precision, dimension (1:n,1:m) :: mat_dash
        double precision, dimension (1:m) :: lanex, ax,bx,cx,dx
        double precision, dimension (1:n) :: laney, ay,by,cy,dy
        double precision :: ae, aw, an, as, ap, delv,b, eps
        integer :: i,j,c
        c = 0
        !using if-statements for switching sweeps
        !discretize to find a,b,c,d
        if (sweep == '+x') then
            do while (.true.)
                c = c + 1
                print*,c
                eps = 0.0
                !backup matrix
                mat_dash = mat
                do j = 2,m-1
                    laney = mat_dash(:,j)
                    !setting boundaries
                    ay(1) = 1.0
                    by(1) = 0.0
                    cy(1) = 0.0
                    dy(1) = mat_dash(1,j)
                    ay(n) = 1.0
                    by(n) = 0.0
                    cy(n) = 0.0
                    dy(n) = mat_dash(n,j)
                    !computing coefficients for inner points
                    do i = 2,n-1
                        ae = gmae(i,j)*Del_y(i,j)/delxe(i,j)
                        aw = gmaw(i,j)*Del_y(i,j)/delxw(i,j)
                        an = gman(i,j)*Del_x(i,j)/delyn(i,j)
                        as = gmas(i,j)*Del_x(i,j)/delys(i,j)
                        delv = Del_x(i,j)*Del_y(i,j)
                        b = sc(i,j)*delv
                        ap = ae+aw+an+as-(sp(i,j)*delv)
                        ay(i) = ap
                        by(i) = an
                        cy(i) = as
                        dy(i) = ae*mat_dash(i,j+1)+aw*mat_dash(i,j-1)+b
                    end do
                    call onetdma(n,ay,by,cy,dy,laney)
                    mat(:,j) = laney
                end do
                !error using eucleadian distance
                eps = sqrt(sum((mat-mat_dash)**2))
                if (eps<1e-10) exit
            end do
        end if
        if (sweep == '-x') then
            do while (.true.)
                c = c + 1
                print*,c
                eps = 0.0
                !backup matrix
                mat_dash = mat
                do j = m-1,2,-1
                    laney = mat_dash(:,j)
                    !setting boundaries
                    ay(1) = 1.0
```

```fortran
                    by(1) = 0.0
                    cy(1) = 0.0
                    dy(1) = mat_dash(1,j)
                    ay(n) = 1.0
                    by(n) = 0.0
                    cy(n) = 0.0
                    dy(n) = mat_dash(n,j)
                    !computing coefficients for inner points
                    do i = 2,n-1
                        ae = gmae(i,j)*Del_y(i,j)/delxe(i,j)
                        aw = gmaw(i,j)*Del_y(i,j)/delxw(i,j)
                        an = gman(i,j)*Del_x(i,j)/delyn(i,j)
                        as = gmas(i,j)*Del_x(i,j)/delys(i,j)
                        delv = Del_x(i,j)*Del_y(i,j)
                        b = sc(i,j)*delv
                        ap = ae+aw+an+as-(sp(i,j)*delv)
                        ay(i) = ap
                        by(i) = an
                        cy(i) = as
                        dy(i) = ae*mat_dash(i,j+1)+aw*mat_dash(i,j-1)+b
                    end do
                    call onetdma(n,ay,by,cy,dy,laney)
                    mat(:,j) = laney
                end do
                !error using eucleadian distance
                eps = sqrt(sum((mat-mat_dash)**2))
                if (eps<1e-10) exit
            end do
        end if
        if (sweep == '+y') then
            do while (.true.)
                c = c + 1
                print*,c
                eps = 0.0
                !backup matrix
                mat_dash = mat
                do i = 2,n-1
                    lanex = mat_dash(i,:)
                    !setting boundaries
                    ax(1) = 1.0
                    bx(1) = 0.0
                    cx(1) = 0.0
                    dx(1) = mat_dash(i,1)
                    ax(m) = 1.0
                    bx(m) = 0.0
                    cx(m) = 0.0
                    dx(m) = mat_dash(i,m)
                    !computing coefficients for inner points
                    do j = 2,m-1
                        ae = gmae(i,j)*Del_y(i,j)/delxe(i,j)
                        aw = gmaw(i,j)*Del_y(i,j)/delxw(i,j)
                        an = gman(i,j)*Del_x(i,j)/delyn(i,j)
                        as = gmas(i,j)*Del_x(i,j)/delys(i,j)
                        delv = Del_x(i,j)*Del_y(i,j)
                        b = sc(i,j)*delv
                        ap = ae+aw+an+as-(sp(i,j)*delv)
                        ax(j) = ap
                        bx(j) = ae
                        cx(j) = aw
                        dx(j) = an*mat_dash(i+1,j)+as*mat_dash(i-1,j)+b
                    end do
                    call onetdma(m,ax,bx,cx,dx,lanex)
                    mat(i,:) = lanex
                end do
                !error using eucleadian distance
                eps = sqrt(sum((mat-mat_dash)**2))
                if (eps<1e-10) exit
            end do
        end if
        if (sweep == '-y') then
```

```fortran
            do while (.true.)
                c = c + 1
                print*,c
                eps = 0.0
                !backup matrix
                mat_dash = mat
                do i = n-1,2,-1
                    lanex = mat_dash(i,:)
                    !setting boundaries
                    ax(1) = 1.0
                    bx(1) = 0.0
                    cx(1) = 0.0
                    dx(1) = mat_dash(i,1)
                    ax(m) = 1.0
                    bx(m) = 0.0
                    cx(m) = 0.0
                    dx(m) = mat_dash(i,m)
                    !computing coefficients for inner points
                    do j = 2,m-1
                        ae = gmae(i,j)*Del_y(i,j)/delxe(i,j)
                        aw = gmaw(i,j)*Del_y(i,j)/delxw(i,j)
                        an = gman(i,j)*Del_x(i,j)/delyn(i,j)
                        as = gmas(i,j)*Del_x(i,j)/delys(i,j)
                        delv = Del_x(i,j)*Del_y(i,j)
                        b = sc(i,j)*delv
                        ap = ae+aw+an+as-(sp(i,j)*delv)
                        ax(j) = ap
                        bx(j) = ae
                        cx(j) = aw
                        dx(j) = an*mat_dash(i+1,j)+as*mat_dash(i-1,j)+b
                    end do
                    call onetdma(m,ax,bx,cx,dx,lanex)
                    mat(i,:) = lanex
                end do
                !error using eucleadian distance
                eps = sqrt(sum((mat-mat_dash)**2))
                if (eps<1e-10) exit
            end do
        end if
    end subroutine twodtdma
    !subroutine to solve two dimensional TDMA as done before but with neuman boundary conditions
    subroutine newtdma2d(n,m,mat,gmae,gmaw,gman,gmas,Del_x,Del_y,delxe,delxw,delyn,delys,sc,sp,p,q,phipq,quee,quew,quen,ques,omega,sweep)
        implicit none
        !declarations
        character(2), intent (in) :: sweep
        integer, intent (in) :: n,m,p,q
        double precision, dimension (1:n,1:m), intent (in) :: gmae,gmaw,gman,gmas,Del_x,Del_y,delxe,delxw,delyn,delys,sc,sp
        double precision, dimension (1:n,1:m), intent (inout) :: mat
        double precision, intent (in) :: phipq,quee,quew,quen,ques,omega
        double precision, dimension (1:n,1:m) :: mat_dash,mat_star
        double precision, dimension (1:m) :: lanex, ax,bx,cx,dx
        double precision, dimension (1:n) :: laney, ay,by,cy,dy
        double precision :: ae, aw, an, as, ap, delv,b, eps
        integer :: i,j,c
        c = 0
        !using if-statements for switching sweeps
        if(sweep == '+y') then
        do while (.true.)
            c = c + 1
            print*,c
            eps = 0.0
            mat(p,q) = phipq
            !backup matrix
            mat_dash = mat
            mat_star = mat
            do i = 2,n-1
                !fixing coefficients for boundary points and fixed point
                ax(1) = 1.0
                bx(1) = 0.0
                cx(1) = 0.0
                dx(1) = mat_dash(i,1)
                ax(m) = 1.0
                bx(m) = 0.0
                cx(m) = 0.0
                dx(m) = mat_dash(i,m)
                ax(q) = 1.0
                bx(q) = 0.0
                cx(q) = 0.0
                dx(q) = mat_dash(p,q)
                lanex = mat_dash(i,:)
                !computing coefficients for inner points
                do j = 2,m-1
                    if ((i==p).and.(j==q)) cycle
                    ae = gmae(i,j)*Del_y(i,j)/delxe(i,j)
                    aw = gmaw(i,j)*Del_y(i,j)/delxw(i,j)
                    an = gman(i,j)*Del_x(i,j)/delyn(i,j)
                    as = gmas(i,j)*Del_x(i,j)/delys(i,j)
                    delv = Del_x(i,j)*Del_y(i,j)
                    b = sc(i,j)*delv
                    ap = ae+aw+an+as-(sp(i,j)*delv)
                    ax(j) = ap
                    bx(j) = ae
                    cx(j) = aw
                    dx(j) = an*mat_dash(i+1,j)+as*mat_dash(i-1,j)+b
                end do
                call onetdma(m,ax,bx,cx,dx,lanex)
                mat_star(i,:) = lanex
            end do
            !boundaries
            do i = 1,m
                mat_star(1,i) = (ques+mat(2,i)*(gmas(2,i)/delys(2,i)))/(gmas(2,i)/delys(2,i))
                mat_star(n,i) = (quen+mat(n-1,i)*(gman(n-1,i)/delyn(n-1,i)))/(gman(n-1,i)/delyn(n-1,i))
            end do
            do i = 1,n
                mat_star(i,1) = (quew+mat(i,2)*(gmaw(i,2)/delxw(i,2)))/(gmaw(i,2)/delxw(i,2))
                mat_star(i,m) = (quee+mat(i,m-1)*(gmae(i,m-1)/delxe(i,m-1)))/(gmae(i,m-1)/delxe(i,m-1))
            end do
            !corner points
            mat_star(1,1) = (mat_star(1,2)+mat_star(2,1))*0.5
            mat_star(1,n) = (mat_star(1,n-1)+mat_star(2,n))*0.5
            mat_star(n,1) = (mat_star(1,2)+mat_star(n-1,1))*0.5
            mat_star(n,n) = (mat_star(n-1,n)+mat_star(n,n-1))*0.5
            !relaxation
            mat = omega*mat_star+(1-omega)*mat_dash
            !error using eucleadian distance
            eps = sqrt(sum((mat-mat_dash)**2))
            if (eps<1e-10) exit
        end do
        end if

        if(sweep == '-y') then
        do while (.true.)
            c = c + 1
            print*,c
            eps = 0.0
            mat(p,q) = phipq
            !backup matrix
            mat_dash = mat
            mat_star = mat
            do i = n-1,2,-1
                !fixing coefficients for boundary points and fixed point
                ax(1) = 1.0
                bx(1) = 0.0
                cx(1) = 0.0
                dx(1) = mat_dash(i,1)
                ax(m) = 1.0
                bx(m) = 0.0
```

```fortran
                cx(m) = 0.0
                dx(m) = mat_dash(i,m)
                ax(q) = 1.0
                bx(q) = 0.0
                cx(q) = 0.0
                dx(q) = mat_dash(p,q)
                lanex = mat_dash(i,:)
                !computing coefficients for inner points
                do j = 2,m-1
                    if ((i==p).and.(j==q)) cycle
                    ae = gmae(i,j)*Del_y(i,j)/delxe(i,j)
                    aw = gmaw(i,j)*Del_y(i,j)/delxw(i,j)
                    an = gman(i,j)*Del_x(i,j)/delyn(i,j)
                    as = gmas(i,j)*Del_x(i,j)/delys(i,j)
                    delv = Del_x(i,j)*Del_y(i,j)
                    b = sc(i,j)*delv
                    ap = ae+aw+an+as-(sp(i,j)*delv)
                    ax(j) = ap
                    bx(j) = ae
                    cx(j) = aw
                    dx(j) = an*mat_dash(i+1,j)+as*mat_dash(i-1,j)+b
                end do
                call onetdma(m,ax,bx,cx,dx,lanex)
                mat_star(i,:) = lanex
            end do
            !boundaries
            do i = 1,m
                mat_star(1,i) = (ques+mat(2,i)*(gmas(2,i)/delys(2,i)))/(gmas(2,i)/delys(2,i))
                mat_star(n,i) = (quen+mat(n-1,i)*(gman(n-1,i)/delyn(n-1,i)))/(gman(n-1,i)/delyn(n-1,i))
            end do
            do i = 1,n
                mat_star(i,1) = (quew+mat(i,2)*(gmaw(i,2)/delxw(i,2)))/(gmaw(i,2)/delxw(i,2))
                mat_star(i,m) = (quee+mat(i,m-1)*(gmae(i,m-1)/delxe(i,m-1)))/(gmae(i,m-1)/delxe(i,m-1))
            end do
            !corner points
            mat_star(1,1) = (mat_star(1,2)+mat_star(2,1))*0.5
            mat_star(1,n) = (mat_star(1,n-1)+mat_star(2,n))*0.5
            mat_star(n,1) = (mat_star(1,2)+mat_star(n-1,1))*0.5
            mat_star(n,n) = (mat_star(n-1,n)+mat_star(n,n-1))*0.5
            !relaxation
            mat = omega*mat_star+(1-omega)*mat_dash
            !error using eucleadian distance
            eps = sqrt(sum((mat-mat_dash)**2))
            if (eps<1e-10) exit
        end do
        end if

        if(sweep == '+x') then
        do while (.true.)
            c = c + 1
            print*,c
            eps = 0.0
            mat(p,q) = phipq
            !backup matrix
            mat_dash = mat
            mat_star = mat
            do j = 2,m-1
                !fixing coefficients for boundary points and fixed point
                ay(1) = 1.0
                by(1) = 0.0
                cy(1) = 0.0
                dy(1) = mat_dash(1,j)
                ay(n) = 1.0
                by(n) = 0.0
                cy(n) = 0.0
                dy(n) = mat_dash(n,j)
                ay(p) = 1.0
                by(p) = 0.0
                cy(p) = 0.0
                dy(p) = mat_dash(p,q)
                laney = mat_dash(:,j)
                !computing coefficients for inner points
                do i = 2,n-1
                    if ((i==p).and.(j==q)) cycle
                    ae = gmae(i,j)*Del_y(i,j)/delxe(i,j)
                    aw = gmaw(i,j)*Del_y(i,j)/delxw(i,j)
                    an = gman(i,j)*Del_x(i,j)/delyn(i,j)
                    as = gmas(i,j)*Del_x(i,j)/delys(i,j)
                    delv = Del_x(i,j)*Del_y(i,j)
                    b = sc(i,j)*delv
                    ap = ae+aw+an+as-(sp(i,j)*delv)
                    ay(i) = ap
                    by(i) = an
                    cy(i) = as
                    dy(i) = ae*mat_dash(i,j+1)+aw*mat_dash(i,j-1)+b
                end do
                call onetdma(n,ay,by,cy,dy,laney)
                mat_star(:,j) = laney
            end do
            !boundaries
            do i = 1,m
                mat_star(1,i) = (ques+mat(2,i)*(gmas(2,i)/delys(2,i)))/(gmas(2,i)/delys(2,i))
                mat_star(n,i) = (quen+mat(n-1,i)*(gman(n-1,i)/delyn(n-1,i)))/(gman(n-1,i)/delyn(n-1,i))
            end do
            do i = 1,n
                mat_star(i,1) = (quew+mat(i,2)*(gmaw(i,2)/delxw(i,2)))/(gmaw(i,2)/delxw(i,2))
                mat_star(i,m) = (quee+mat(i,m-1)*(gmae(i,m-1)/delxe(i,m-1)))/(gmae(i,m-1)/delxe(i,m-1))
            end do
            !corner points
            mat_star(1,1) = (mat_star(1,2)+mat_star(2,1))*0.5
            mat_star(1,n) = (mat_star(1,n-1)+mat_star(2,n))*0.5
            mat_star(n,1) = (mat_star(1,2)+mat_star(n-1,1))*0.5
            mat_star(n,n) = (mat_star(n-1,n)+mat_star(n,n-1))*0.5
            !relaxation
            mat = omega*mat_star+(1-omega)*mat_dash
            !error using eucleadian distance
            eps = sqrt(sum((mat-mat_dash)**2))
            if (eps<1e-10) exit
        end do
        end if

        if(sweep == '-x') then
        do while (.true.)
            c = c + 1
            print*,c
            eps = 0.0
            mat(p,q) = phipq
            !backup matrix
            mat_dash = mat
            mat_star = mat
            do j = m-1,2,-1
                !fixing coefficients for boundary points and fixed point
                ay(1) = 1.0
                by(1) = 0.0
                cy(1) = 0.0
                dy(1) = mat_dash(1,j)
                ay(n) = 1.0
                by(n) = 0.0
                cy(n) = 0.0
                dy(n) = mat_dash(n,j)
                ay(p) = 1.0
                by(p) = 0.0
                cy(p) = 0.0
                dy(p) = mat_dash(p,q)
                laney = mat_dash(:,j)
                !computing coefficients for inner points
                do i = 2,n-1
                    if ((i==p).and.(j==q)) cycle
                    ae = gmae(i,j)*Del_y(i,j)/delxe(i,j)
                    aw = gmaw(i,j)*Del_y(i,j)/delxw(i,j)
```

```fortran
                    an = gman(i,j)*Del_x(i,j)/delyn(i,j)
                    as = gmas(i,j)*Del_x(i,j)/delys(i,j)
                    delv = Del_x(i,j)*Del_y(i,j)
                    b = sc(i,j)*delv
                    ap = ae+aw+an+as-(sp(i,j)*delv)
                    ay(i) = ap
                    by(i) = an
                    cy(i) = as
                    dy(i) = ae*mat_dash(i,j+1)+aw*mat_dash(i,j-1)+b
                end do
                call onetdma(n,ay,by,cy,dy,laney)
                mat_star(:,j) = laney
            end do
            do i = 1,m
                mat_star(1,i) = (ques+mat(2,i)*(gmas(2,i)/delys(2,i)))/(gmas(2,i)/delys(2,i))
                mat_star(n,i) = (quen+mat(n-1,i)*(gman(n-1,i)/delyn(n-1,i)))/(gman(n-1,i)/delyn(n-1,i))
            end do
            do i = 1,n
                mat_star(i,1) = (quew+mat(i,2)*(gmaw(i,2)/delxw(i,2)))/(gmaw(i,2)/delxw(i,2))
                mat_star(i,m) = (quee+mat(i,m-1)*(gmae(i,m-1)/delxe(i,m-1)))/(gmae(i,m-1)/delxe(i,m-1))
            end do
            !corner points
            mat_star(1,1) = (mat_star(1,2)+mat_star(2,1))*0.5
            mat_star(1,n) = (mat_star(1,n-1)+mat_star(2,n))*0.5
            mat_star(n,1) = (mat_star(1,2)+mat_star(n-1,1))*0.5
            mat_star(n,n) = (mat_star(n-1,n)+mat_star(n,n-1))*0.5
            !relaxation
            mat = omega*mat_star+(1-omega)*mat_dash
            !error using eucleadian distance
            eps = sqrt(sum((mat-mat_dash)**2))
            if (eps<1e-10) exit
        end do
        end if
    end subroutine newtdma2d
    !code for solving a line using matrix inversion, uses LU inversion method to inverse the matrix
    subroutine inmatsolve(n,a,b,c,d,line)
        double precision, dimension(n,n) :: matA,matAinv
        integer, intent(in) :: n
        double precision, dimension(n), intent(in) :: a,b,c,d
        double precision, dimension(n), intent(inout) :: line
        integer :: i,flag,rec
        flag = 0
        do i = 2,n-1
            if((a(i)==1).and.(b(i)==0).and.(c(i)==0)) then
                flag = 1
                rec = i
            end if
        end do
        if(flag==0) then
        !build a matrix
        matA = 0.0
        matA(1,1) = a(1)
        do i = 2,n-1
            matA(i,i-1) = -1.0*c(i)
            matA(i,i) = a(i)
            matA(i,i+1) = -1.0*b(i)
        end do
        matA(n,n) = a(n)
        call inverse(matA,matAinv,n)
        line = matmul(matAinv,d)
        end if
        if(flag==1) then
            call inmatsolverec(rec,a(1:rec),b(1:rec),c(1:rec),d(1:rec),line(1:rec))
            call inmatsolverec(n-rec+1,a(rec:n),b(rec:n),c(rec:n),d(rec:n),line(rec:n))
        end if
    end subroutine inmatsolve
    !recursive image of inmatsolve
    subroutine inmatsolverec(n,a,b,c,d,line)
        double precision, dimension(n,n) :: matA,matAinv
        integer, intent(in) :: n
```

```fortran
      double precision, dimension(n), intent(in) :: a,b,c,d
      double precision, dimension(n), intent(inout) :: line
      integer :: i
      !build a matrix
      matA = 0.0
      matA(1,1) = a(1)
      do i = 2,N-1
        matA(i,i-1) = -1.0*c(i)
        matA(i,i) = a(i)
        matA(i,i+1) = -1.0*b(i)
      end do
      matA(n,n) = a(n)
      call inverse(matA,matAinv,n)
      line = matmul(matAinv,d)
    end subroutine inmatsolverec
    !extending matrix inversion solver to two dimensions, neuman BC
    subroutine newtdma2dinmat(n,m,mat,gmae,gmaw,gman,gmas,Del_x,Del_y,delxe,delxw,delyn,delys,sc,sp,p,q,phipq,quee,quew,quen,ques,omega,sweep)
      implicit none
      !declarations
      character(2), intent (in) :: sweep
      integer, intent (in) :: n,m,p,q
      double precision, dimension (1:n,1:m), intent (in) :: gmae,gmaw,gman,gmas,Del_x,Del_y,delxe,delxw,delyn,delys,sc,sp
      double precision, dimension (1:n,1:m), intent (inout) :: mat
      double precision, intent (in) :: phipq,quee,quew,quen,ques,omega
      double precision, dimension (1:n,1:m) :: mat_dash,mat_star
      double precision, dimension (1:m) :: lanex, ax,bx,cx,dx
      double precision, dimension (1:n) :: laney, ay,by,cy,dy
      double precision :: ae, aw, an, as, ap, delv,b, eps
      integer :: i,j,c
      c = 0
      !using if-statements for switching sweeps
      if(sweep == '+y') then
```

```fortran
        do while (.true.)
            c = c + 1
            print*,c
            eps = 0.0
            mat(p,q) = phipq
            !backup matrix
            mat_dash = mat
            mat_star = mat
            do i = 2,n-1
                !fixing coefficients for boundary points and a fixed point
                ax(1) = 1.0
                bx(1) = 0.0
                cx(1) = 0.0
                dx(1) = mat_dash(i,1)
                ax(m) = 1.0
                bx(m) = 0.0
                cx(m) = 0.0
                dx(m) = mat_dash(i,m)
                ax(q) = 1.0
                bx(q) = 0.0
                cx(q) = 0.0
                dx(q) = mat_dash(p,q)
                lanex = mat_dash(i,:)
                !computing coefficients for inner points
                do j = 2,m-1
                    if ((i==p).and.(j==q)) cycle
                    ae = gmae(i,j)*Del_y(i,j)/delxe(i,j)
                    aw = gmaw(i,j)*Del_y(i,j)/delxw(i,j)
                    an = gman(i,j)*Del_x(i,j)/delyn(i,j)
                    as = gmas(i,j)*Del_x(i,j)/delys(i,j)
                    delv = Del_x(i,j)*Del_y(i,j)
                    b = sc(i,j)*delv
                    ap = ae+aw+an+as-(sp(i,j)*delv)
                    ax(j) = ap
                    bx(j) = ae
```

```
          cx(j) = aw
          dx(j) = an*mat_dash(i+1,j)+as*mat_dash(i-1,j)+b
        end do
        call inmatsolve(m,ax,bx,cx,dx,lanex)
        mat_star(i,:) = lanex
      end do
      !boundaries
      do i = 1,m
        mat_star(1,i) = (ques+mat(2,i)*(gmas(2,i)/delys(2,i)))/(gmas(2,i)/delys(2,i))
        mat_star(n,i) = (quen+mat(n-1,i)*(gman(n-1,i)/delyn(n-1,i)))/(gman(n-1,i)/delyn(n-1,i))
      end do
      do i = 1,n
        mat_star(i,1) = (quew+mat(i,2)*(gmaw(i,2)/delxw(i,2)))/(gmaw(i,2)/delxw(i,2))
        mat_star(i,m) = (quee+mat(i,m-1)*(gmae(i,m-1)/delxe(i,m-1)))/(gmae(i,m-1)/delxe(i,m-1))
      end do
      !corner points
      mat_star(1,1) = (mat_star(1,2)+mat_star(2,1))*0.5
      mat_star(1,n) = (mat_star(1,n-1)+mat_star(2,n))*0.5
      mat_star(n,1) = (mat_star(1,2)+mat_star(n-1,1))*0.5
      mat_star(n,n) = (mat_star(n-1,n)+mat_star(n,n-1))*0.5
      !relaxation
      mat = omega*mat_star+(1-omega)*mat_dash
      !error using eucleadian distance
      eps = sqrt(sum((mat-mat_dash)**2))
      if (eps<1e-10) exit
    end do
    end if
    if(sweep == '-y') then
    do while (.true.)
      c = c + 1
      print*,c
      eps = 0.0
      mat(p,q) = phipq
      !backup matrix
      mat_dash = mat
```

```fortran
            mat_star = mat
            do i = n-1,2,-1
                !fixing coefficients for boundary points and a fixed point
                ax(1) = 1.0
                bx(1) = 0.0
                cx(1) = 0.0
                dx(1) = mat_dash(i,1)
                ax(m) = 1.0
                bx(m) = 0.0
                cx(m) = 0.0
                dx(m) = mat_dash(i,m)
                ax(q) = 1.0
                bx(q) = 0.0
                cx(q) = 0.0
                dx(q) = mat_dash(p,q)
                lanex = mat_dash(i,:)
                !computing coefficients for inner points
                do j = 2,m-1
                    if ((i==p).and.(j==q)) cycle
                    ae = gmae(i,j)*Del_y(i,j)/delxe(i,j)
                    aw = gmaw(i,j)*Del_y(i,j)/delxw(i,j)
                    an = gman(i,j)*Del_x(i,j)/delyn(i,j)
                    as = gmas(i,j)*Del_x(i,j)/delys(i,j)
                    delv = Del_x(i,j)*Del_y(i,j)
                    b = sc(i,j)*delv
                    ap = ae+aw+an+as-(sp(i,j)*delv)
                    ax(j) = ap
                    bx(j) = ae
                    cx(j) = aw
                    dx(j) = an*mat_dash(i+1,j)+as*mat_dash(i-1,j)+b
                end do
                call inmatsolve(m,ax,bx,cx,dx,lanex)
                mat_star(i,:) = lanex
            end do
            !boundaries
            do i = 1,m
                mat_star(1,i) = (ques+mat(2,i)*(gmas(2,i)/delys(2,i)))/(gmas(2,i)/delys(2,i))
                mat_star(n,i) = (quen+mat(n-1,i)*(gman(n-1,i)/delyn(n-1,i)))/(gman(n-1,i)/delyn(n-1,i))
            end do
            do i = 1,n
                mat_star(i,1) = (quew+mat(i,2)*(gmaw(i,2)/delxw(i,2)))/(gmaw(i,2)/delxw(i,2))
                mat_star(i,m) = (quee+mat(i,m-1)*(gmae(i,m-1)/delxe(i,m-1)))/(gmae(i,m-1)/delxe(i,m-1))
            end do
            !corner points
            mat_star(1,1) = (mat_star(1,2)+mat_star(2,1))*0.5
            mat_star(1,n) = (mat_star(1,n-1)+mat_star(2,n))*0.5
            mat_star(n,1) = (mat_star(1,2)+mat_star(n-1,1))*0.5
            mat_star(n,n) = (mat_star(n-1,n)+mat_star(n,n-1))*0.5
            !relaxation
            mat = omega*mat_star+(1-omega)*mat_dash
            !error using eucleadian distance
            eps = sqrt(sum((mat-mat_dash)**2))
            if (eps<1e-10) exit
        end do
        end if

        if(sweep == '+x') then
        do while (.true.)
            c = c + 1
            print*,c
            eps = 0.0
            mat(p,q) = phipq
            !backup matrix
            mat_dash = mat
            mat_star = mat
            do j = 2,m-1
                !fixing coefficients for boundary points and a fixed point
                ay(1) = 1.0
                by(1) = 0.0
                cy(1) = 0.0
                dy(1) = mat_dash(1,j)
                ay(n) = 1.0
                by(n) = 0.0
                cy(n) = 0.0
                dy(n) = mat_dash(n,j)
                ay(p) = 1.0
                by(p) = 0.0
                cy(p) = 0.0
                dy(p) = mat_dash(p,q)
                laney = mat_dash(:,j)
                !computing coefficients for inner points
                do i = 2,n-1
                    if ((i==p).and.(j==q)) cycle
                    ae = gmae(i,j)*Del_y(i,j)/delxe(i,j)
                    aw = gmaw(i,j)*Del_y(i,j)/delxw(i,j)
                    an = gman(i,j)*Del_x(i,j)/delyn(i,j)
                    as = gmas(i,j)*Del_x(i,j)/delys(i,j)
                    delv = Del_x(i,j)*Del_y(i,j)
                    b = sc(i,j)*delv
                    ap = ae+aw+an+as-(sp(i,j)*delv)
                    ay(i) = ap
                    by(i) = an
                    cy(i) = as
                    dy(i) = ae*mat_dash(i,j+1)+aw*mat_dash(i,j-1)+b
                end do
                call inmatsolve(n,ay,by,cy,dy,laney)
                mat_star(:,j) = laney
            end do
            !boundaries
            do i = 1,m
                mat_star(1,i) = (ques+mat(2,i)*(gmas(2,i)/delys(2,i)))/(gmas(2,i)/delys(2,i))
                mat_star(n,i) = (quen+mat(n-1,i)*(gman(n-1,i)/delyn(n-1,i)))/(gman(n-1,i)/delyn(n-1,i))
            end do
            do i = 1,n
                mat_star(i,1) = (quew+mat(i,2)*(gmaw(i,2)/delxw(i,2)))/(gmaw(i,2)/delxw(i,2))
                mat_star(i,m) = (quee+mat(i,m-1)*(gmae(i,m-1)/delxe(i,m-1)))/(gmae(i,m-1)/delxe(i,m-1))
            end do
            !corner points
            mat_star(1,1) = (mat_star(1,2)+mat_star(2,1))*0.5
            mat_star(1,n) = (mat_star(1,n-1)+mat_star(2,n))*0.5
            mat_star(n,1) = (mat_star(1,2)+mat_star(n-1,1))*0.5
            mat_star(n,n) = (mat_star(n-1,n)+mat_star(n,n-1))*0.5
            !relaxation
            mat = omega*mat_star+(1-omega)*mat_dash
            !error using eucleadian distance
            eps = sqrt(sum((mat-mat_dash)**2))
            if (eps<1e-10) exit
        end do
        end if

        if(sweep == '-x') then
        do while (.true.)
            c = c + 1
            print*,c
            eps = 0.0
            mat(p,q) = phipq
            !backup matrix
            mat_dash = mat
            mat_star = mat
            do j = m-1,2,-1
                !fixing coefficients for boundary points and a fixed point
                ay(1) = 1.0
                by(1) = 0.0
                cy(1) = 0.0
                dy(1) = mat_dash(1,j)
                ay(n) = 1.0
                by(n) = 0.0
                cy(n) = 0.0
                dy(n) = mat_dash(n,j)
                ay(p) = 1.0
                by(p) = 0.0
                cy(p) = 0.0
                dy(p) = mat_dash(p,q)
                laney = mat_dash(:,j)
                !computing coefficients for inner points
                do i = 2,n-1
                    if ((i==p).and.(j==q)) cycle
                    ae = gmae(i,j)*Del_y(i,j)/delxe(i,j)
                    aw = gmaw(i,j)*Del_y(i,j)/delxw(i,j)
                    an = gman(i,j)*Del_x(i,j)/delyn(i,j)
                    as = gmas(i,j)*Del_x(i,j)/delys(i,j)
                    delv = Del_x(i,j)*Del_y(i,j)
                    b = sc(i,j)*delv
                    ap = ae+aw+an+as-(sp(i,j)*delv)
                    ay(i) = ap
                    by(i) = an
                    cy(i) = as
                    dy(i) = ae*mat_dash(i,j+1)+aw*mat_dash(i,j-1)+b
                end do
                if ((i==2).and.(c==1)) then
                    !print*,ax
                    !print*,bx
                    !print*,cx
                    !print*,dx
                end if
                call inmatsolve(n,ay,by,cy,dy,laney)
                mat_star(:,j) = laney
            end do
            do i = 1,m
                mat_star(1,i) = (ques+mat(2,i)*(gmas(2,i)/delys(2,i)))/(gmas(2,i)/delys(2,i))
                mat_star(n,i) = (quen+mat(n-1,i)*(gman(n-1,i)/delyn(n-1,i)))/(gman(n-1,i)/delyn(n-1,i))
            end do
            do i = 1,n
                mat_star(i,1) = (quew+mat(i,2)*(gmaw(i,2)/delxw(i,2)))/(gmaw(i,2)/delxw(i,2))
                mat_star(i,m) = (quee+mat(i,m-1)*(gmae(i,m-1)/delxe(i,m-1)))/(gmae(i,m-1)/delxe(i,m-1))
```

```fortran
            end do
            !corner points
            mat_star(1,1) = (mat_star(1,2)+mat_star(2,1))*0.5
            mat_star(1,n) = (mat_star(1,n-1)+mat_star(2,n))*0.5
            mat_star(n,1) = (mat_star(1,2)+mat_star(n-1,1))*0.5
            mat_star(n,n) = (mat_star(n-1,n)+mat_star(n,n-1))*0.5
            !relaxation
            mat = omega*mat_star+(1-omega)*mat_dash
            !error using eucleadian distance
            eps = sqrt(sum((mat-mat_dash)**2))
            if (eps<1e-10) exit
        end do
        end if
    end subroutine newtdma2dinmat

end module
```