```
!
!  Description: Newton's method (parallel version)
!  function to compute the inexact newton method using CG for the non
!  linear thermal conduction equation
! ----------------------------------------------------------------------

subroutine newton(A,u,lambda,beta,tau,kmax,its,ierr)

  use header
  implicit none
  include 'mpif.h'
  type(matrix), intent(in) :: A
  type(matrix) :: J
  type(vector), intent(inout) :: u
  type(vector) :: r,s
  integer, intent(in) :: kmax
  integer, intent(inout) :: ierr
  integer ::  maxits,itscg,itscgsave,myid
  integer :: its,n_loc,nrows,nprocs,uflag
  real(kind = 8) :: eps,eps_bar,rtr,gamma,rtr_prev,t1,t2,t3,tn1,tn2
  real(kind = 8) :: Vec_Dot
  real(kind = 8), intent(in) :: beta,lambda,tau

  ! returns the processor identification number in 'myid'

  call mpi_comm_rank(mpi_comm_world,myid,ierr)

  ! calculate the number of rows in distributed A and number of local indicies in distrib
uted vectors

  n_loc = u%iend - u%ibeg + 1
  nrows = A%iend - A%ibeg + 1

  ! allocate memory to processors for vectors and matrix s, r, and J respectively

  allocate(J%aa(5*nrows))
  allocate(J%jj(5*nrows))
  allocate(J%ii(A%n+1))

  J%n = A%n
  J%ibeg = A%ibeg
  J%iend = A%iend


  allocate(r%xx(u%n))
  allocate(s%xx(u%n))

  r%n = u%n
  r%ibeg = u%ibeg
  r%iend = u%iend

  s%n = u%n
  s%ibeg = u%ibeg
  s%iend = u%iend

  ! initialise vectors s and r

  call dcopy(n_loc,u%xx(u%ibeg),1,s%xx(u%ibeg),1)
  call dcopy(n_loc,u%xx(u%ibeg),1,r%xx(u%ibeg),1)

  ! Initialise max iterations for CG, eps_0, CG iteration count, \bar{eps}, gamma and CG
total time count.

  itscgsave = 0
  gamma = 0.9_8
  eps = 0.1_8
  eps_bar = 0.1_8
```

```
  t3 = 0.0_8
  maxits = 9999

  ! calculate r_0

  call func(A,u,r,beta,lambda,n_loc)

  ! calculate ||r_0||_2

  rtr = Vec_Dot(r,r)
  rtr_prev = rtr

  ! start clock for inexact newton loop

  tn1 = mpi_wtime()

  ! start iterative loop

  do its=1,kmax

     ! is ||r_k||_2<= tau?

     if (sqrt(rtr) .LE. tau) then
        ierr = 0
        exit
     end if

     ! compute epsilon for conugate gradient input

     if (its>1) then

        ! compute epsilon using previous value of ||F(U)||^2

        eps = min(eps_bar, gamma*(rtr/rtr_prev))

        ! save current ||F(U)||^2 for next iteration

        rtr_prev = rtr
     end if

     ! scale r by -1

     call dscal(n_loc,-1.0_8,r%xx(r%ibeg),1)

     ! calculate the jacobian F'(U) at current solution point

     call jacobian(A,u,J,beta,lambda)

     ! calculate time spent in cg

     t1 = mpi_wtime()

     ! call cg to calculate s at iteration k

     call cg(J,s,r,eps,maxits,itscg)

     t2 = mpi_wtime()
     t3 = t3 + t2-t1

     !update solution U (compute U = U + s)

     call daxpy(n_loc,1.0_8,s%xx(s%ibeg),1,u%xx(u%ibeg),1)

     ! is the updated U solution negative at any point?

     call negu(u,uflag)
     if (uflag>0) then
```

```
        if (myid ==0) then
           print*, 'Solution U becomes negative at iteration ', its,', terminate newton'
        end if
        ! return unique positive error for user
        ierr = 2
        exit
     end if

     ! Residual update

     call func(A,u,r,beta,lambda,n_loc)

     ! compute norm of residual in preperation for next iteration

     rtr = Vec_Dot(r,r)

     ! running count of number of total cg iterations
     itscgsave = itscgsave + itscg
  end do

  tn2 = mpi_wtime()

  ! returns number of processors

  call mpi_comm_size(mpi_comm_world,nprocs,ierr)

  if (nprocs > 1) then
     if (myid ==0) then
        print*, 'Total number of all CG iterations: ', itscgsave
        print*, 'Time per one CG iteration: ', t3/itscgsave
        print*, 'Time per 1 iteration of inexact newton: ', (tn2-tn1)/its
     end if
  else
     print*, 'Total number of all CG iterations: ',itscgsave
     print*, 'Time per one CG iteration: ', (t3)/itscgsave
     print*, 'Time per newton iteration: ', (tn2-tn1)/its
  end if

  if (kmax .LE. its) then
     ! did not converge
     ierr = 1
  end if
  ! deallocate memory
  deallocate(r%xx)
  deallocate(s%xx)
  deallocate(J%aa,J%ii,J%jj)
end subroutine newton
```