```
!
!  Description: Numerical solution of a nonlinear reaction-diffusion
!               equation via Newton's method using BLAS and LAPACK
!
! ----------------------------------------------------------------------
program newton
   implicit none
   integer :: m,n,kmax,i,info,yesno,k,save_rk,tstart
   real(kind=8), dimension(:,:),allocatable :: A, J,rs
   real(kind=8), dimension(:),allocatable :: r, u,ipiv
   real(kind=8) :: dnrm2  ! BLAS function
   real(kind=8) :: tau, t1,t2,t3,t4
   ! indicator to see whether the iteration loop runs
   tstart = 1
   ! Ask user for discretisation parameter m, tolerance tau
   ! and maximun number of iterations kmax.
   print*,'Specify m,tolerance: '
   read*, m,tau
   ! if there is memory avaliable, then produce the table for question 2.
   print*,'Memory avaliable to store computed r_k? (1-yes,0-no): '
   read*, save_rk
   ! Define n = (m-1)^2
   n=(m-1)**2
   ! set max iteration stopping criterion
   kmax = 100
   ! Allocate vectors and matrices
   allocate(rs(n,kmax),A(m,n),J(m,n),r(n),u(n),ipiv(n))
   ! Implement Newton's method
   ! Initialise U = 0
   u = 0.0_8
   ! initialise rs if memory avaliable
   if (save_rk == 1) then
      rs = 0.0_8
   end if
   ! create m x n laplace matrix A
   !start cpu timer for complete run
   call cpu_time(t3)
   call laplace(A,m)
   ! compute r_0 using subroutine func.f90
   call func(A,u,m,r)
   !store r if asked by user
   if (save_rk == 1) then
      rs(:,1) = r
   end if
   !start do loop for newton iteration
   !start cpu timer for average run
   call cpu_time(t1)
   do k=1,kmax
      if (dnrm2(n,r,1)<=tau) then
         print*, 'tolerance reached at iteration: ', k
         tstart = 0
         exit
      end if
      ! Compute the jacobian matrix for U
      call jacobian(A,u,m,J)
      ! solve for the newton step
      r=(-1.0_8)*r
      call dpbsv('U',n,m-1,1,J,m,r,n,info)
      ! update solution U
      u = u + r
      !compute r_{k+1}
      call func(A,u,m,r)
      ! store r_{k+1} if asked by user
      if (save_rk ==1)then
         rs(:,k+1)=r
      end if
   end do
```

```fortran
   !end respective cpu timers
   call cpu_time(t2)
   call cpu_time(t4)
   ! if max iterationn stopping criterion reached, tell user
   if (kmax==k)then
      print*, 'maximum number of iterations reached'
   end if
   ! ask user if they want to see computed solution U
   print*, 'Print value of U? (1-yes, 0-no): '
   read*, yesno
   ! print U if requested by user
   if (yesno ==1) then
      print*,'This is u: '
      do i=1,n
         print*, u(i)
      end do
   end if
   !since h=1/m by x1,x2 (note =1/2) = ih,jh => i,j = (1/2)*m
   !sub into uij = u((j-1)*(m-1)+i) => u((1/2)m^2-m+1)
   !(1/2)m^2-m+1 = ((m-1)^2 + 1)/2 = (n+1)/2
   if (mod(m,2) == 0)then
      print*, 'This is the value at u(1/2,1/2): ', u((n+1)/2)
   end if
   !These are left here so result times in report can be easily replecated by user
   if (tstart ==1) then
      print*, 'average time per run of iteration loop: ',(t2-t1)/(k-1)
   end if
   print*, 'total CPU time to compute solution U: ', t4-t3
   ! display table of residuals if there is enough memory
   if (save_rk == 1) then
      write(*,'(A10," ",A20," ",A20," ",A20)') &
           'k','||r_k||','||r_{k+1}||/||r_k||2'
      print*,'----------------------------------------------'
      if (k>1) then
         do i=1,k-1
            write(*,'(I10," ",E16.8," ",E16.8," ",E16.8)')&
                 i, dnrm2(n,rs(:,i),1), &
                 dnrm2(n,rs(:,i+1),1)/(dnrm2(n,rs(:,i),1)**2)
         end do
      end if
      write(*,'(I10," ",E16.8)',advance='no') k,dnrm2(n,rs(:,k),1)
      print*,'       -not computed-'
   end if
   ! deallocate vectors and matrices
   deallocate(ipiv,A,J,r,u,rs)
end program newton
```