```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! DESCRIPTION: Program to compute the numerical solution of a non-linear
!              equation via efficient practical Quasi-Newton method using
!              BLAS and LAPACK
!
!              Option to save r_k if memory is sufficient
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
program quasi_newton

   implicit none
   integer :: m,n,kmax,i,info,k,yesno,save_rk,tstart
   real(kind=8), dimension(:,:), allocatable :: A, J,s,rs
   real(kind=8), dimension(:), allocatable:: r,u,ipiv
   real(kind=8) :: dnrm2,ddot ! BLAS functions
   real(kind=8) :: tau,t1,t2,t3,t4
   !indicator variable for cpu timer iteration loop called
   ! 0 if not called
   ! 1 if called
   tstart = 0
!  ask user for discretisation parameter m, tollerance tau and max iterations
   print*, 'Specify m, tolerance, kmax<101: '
   read*, m,tau,kmax
   ! check for illegal max iterations
   if (kmax>100)then
      print*, 'illegal kmax inputed, compute using default (kmax=100)'
      kmax = 100
   end if
   ! Ask user if memory available to save r_k and produce table for q4c
   print*, 'Memory avaliable to store computed r_k? (yes-1,no-0): '
   read*, save_rk
   !define n = (m-1)^2
   n=(m-1)**2
   allocate(A(m,n),J(m,n),r(n),u(n),ipiv(n),s(n,kmax),rs(n,kmax))
   u = 0.0_8
   s = 0.0_8
   if (save_rk==1) then
      rs = 0.0_8
   end if
   ! set k=1 for if else statements past loop to be valid
   k=1
   ! start timer for time taken to find U
   call cpu_time(t3)
   ! make m x n laplace matrix A
   call laplace(A,m)
   ! set r_0 = F(u_0)
   call func(A,u,m,r)
   ! save r_0 if memory
   if (save_rk ==1)then
      rs(:,1)=r
   end if
  !is U_0 a solution for given tolerance?
   if (dnrm2(n,r,1)<=tau) then
      print*, 'tolerance reached at iteration 0'
   else
      !approximate jacobian B_0 is J
      call bzero(A,u,m,J,ipiv)
   ! start main quasi-newton precedure
      ! solve B_0s_0 = -r_0
      r = (-1.0_8)*r
      call dpbtrs('U',n,m-1,1,J,m,r,n,info)
      ! store s_0 in matrix s
      s(:,1) = r
      ! start iterative loop to calculate U st F(U)=0
      ! call timer for calculating average run
      !indicator variable tstart indicates that the cpu timer is called for t1
      tstart = 1
```

```
      call cpu_time(t1)
      do k=1,kmax
        !update solution U
         u = u + s(:, k)
        !update r
         call func(A,u,m,r)
         if (save_rk ==1)then
            rs(:,k+1) = r
         end if
         !is updated U a solution given tolerance?
         if (dnrm2(n,r,1)<=tau) then
            print*, 'tolernace reached at iteration', k
            exit
         end if
         ! Stopping citerion for max iterations.
         if (kmax == k) then
            print*, 'maximum number of iteration reached'
            exit
         end if
         ! solve B_0\hat{s}=-r_{k+1} for \hat{s}. Note using storage r.
         r = (-1.0_8)*r
         call dpbtrs('U',n,m-1,1,J,m,r,n,info)
        ! if k>1, then compute sequence of rank 1 updates
         if (k>1) then
            do i=1,k-1
              ! computes constant times vector + vector using lapack
              ! Computes line 12 on pseudocode
               call daxpy(n,ddot(n,s(:,i)/dnrm2(n,s(:,i),1)**2,1,r,1), &
                    s(:,i+1),1,r,1)
            end do
         end if
         ! computes the new s_{k+1}
         s(:,k+1)= r/(1.0_8-ddot(n,s(:,k)/dnrm2(n,s(:,k),1)**2,1,r,1))
      end do
   end if
   ! computes the end time for both timers
   call cpu_time(t2)
   call cpu_time(t4)
   ! displays the solution of U if requested
   print*, 'display solution U? (1-yes, 0-no): '
   read*, yesno
   if (yesno == 1)then
      print*, 'this is u: '
      do i=1,n
         print*, u(i)
      end do
   end if
  ! prints half value if m is even
   if (mod(m,2) ==0)then
      print*, 'This is the value at u(1/2,1/2): ', u((n+1)/2)
   end if
   ! displays CPU times so user can replecate results from report
   if (tstart ==1)then
      !iteration loop was run
      print*, 'average cpu time for one run of iteration loop: ', (t2-t1)/(k-1)
   end if
   print*, 'total cpu time to find solution U', (t4-t3)
   ! if enough memory, displays table of saved r_k norms, as described in q4c.
   if (save_rk ==1)then
      write(*, '(A10," ",A20," ",A20," ",A20," ",A20)') &
           ' k','||r_k||','||r_{k+1}||/||r_k||2' &
           ,'||r_{k+1}||/||r_k|| '
      print*,'----------------------------------------------------------------------
--'
      if (k==1)then
         write(*, '(I10, " ",E16.8)',advance='no') k,dnrm2(n,rs(:,k),1)
         print*,' not computed...'
```

```
      else
         do i=1,k
            write(*, '(I10," ",E16.8," ",E16.8," ",E16.8," ",E16.8)') &
                 i, dnrm2(n,rs(:,i),1), dnrm2(n,rs(:,i+1),1)/(dnrm2(n,rs(:,i),1)**2), &
                 dnrm2(n,rs(:,i+1),1)/dnrm2(n,rs(:,i),1)
         end do
         write(*, '(I10, " ",E16.8)',advance='no') k+1,dnrm2(n,rs(:,k+1),1)
         print*, ' not computed... '
      end if
   end if
      !deallocate matrices and vectors
   deallocate(A,J,r,u,s,rs)
end program quasi_newton
```