```fortran
! file src/main.f90
!
! Copyright 2009-2017 Dalton Harvie (daltonh@unimelb.edu.au)
!
! This file is part of arb finite volume solver, referred to as `arb'.
!
! arb is a software package designed to solve arbitrary partial
! differential equations on unstructured meshes using the finite volume
! method.  Primarily it consists of fortran source code, perl source
! code and shell scripts.  arb replies on certain third party software
! to run, most notably the computer algebra system maxima
! <http://maxima.sourceforge.net/> which is released under the GNU GPL.
!
! The original copyright of arb is held by Dalton Harvie, however the
! project is now under collaborative development.
!
! arb is released under the GNU GPL.  arb is free software: you can
! redistribute it and/or modify it under the terms of the GNU General
! Public License (version 3) as published by the Free Software Foundation.
! You should have received a copy of the GNU General Public Licence
! along with arb (see file licence/gpl.txt after unpacking).  If not,
! see <http://www.gnu.org/licences/>.
!
! arb is distributed in the hope that it will be useful, but WITHOUT
! ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
! FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public Licence
! for more details.
!
! For full details of arb's licence see the licence directory.
!
! The current homepage for the arb finite volume solver project is
! <http://people.eng.unimelb.edu.au/daltonh/downloads/arb>.
!
!-------------------------------------------------------------------------
program arb

! written in f90 with object oriented approach, hopefully
! daltonh, 070608

use general_module
use setup_module
use equation_module
use solver_module
use output_module
!$ use omp_lib

implicit none
character(len=1000) :: formatline
integer :: ierror = 0
logical :: newtconverged
logical, parameter :: debug = .true.

!----------------------------------------------------

formatline = '(a,f4.2,a)'
write(*,fmt=formatline) 'program arb, version ',version,' ('//trim(versionname)//'),
written by dalton harvie'

! find number of threads if openmp is in use
!$omp parallel
!$ nthreads = omp_get_num_threads()
!$omp end parallel
if (nthreads > 1) then
  formatline = '(a,'//trim(dindexformat(nthreads))//',a)'
  write(*,fmt=formatline) 'INFO: openmp version running, with ',nthreads,' threads in use
else if (nthreads == 1) then
  write(*,fmt=formatline) 'INFO: openmp version running, with 1 thread in use'
else
  write(*,'(a,i2,a)') 'INFO: serial version running'
end if
! now make nthreads = 1 for serial version
nthreads = max(nthreads,1)

call initialise_random_number ! initialise the random seed used to evaluate the arb
variable <random>

call time_process
call setup ! sets up variable metadata, reads in values, allocates arrays, creates mesh,
initialises fields etc
call time_process(description='setup')

! increment timestep if timestepaddional is specified
if (transient_simulation.and.timestepadditional > 0) timestepmin =
max(timestepmin,timestep+timestepadditional)
```

```fortran
! output initial conditions if transient and this is the first timestep
if (transient_simulation.and.timestep == 0) then
  call time_process
  call output
  if (trim(output_step_file) == "timestep") call
output_step(action="write",do_update_outputs=.false.)
  call time_process(description='initial transient output')
end if

if (.not.transient_simulation) then
  backline = newtline
  newtline = timeline
end if

!---------------------------------------
time_loop: do while ( &
  (transient_simulation.and..not.check_stopfile("stoptime").and.
((.not.check_condition("stop").and.timestep < timestepmax).or. &
  timestep < timestepmin)).or..not.transient_simulation)

  newtres = huge(1.d0)

  if (transient_simulation) then
    timestep = timestep + 1
    formatline = "(a,"//trim(dindexformat(timestep))//",a)"
    write(*,fmt=formatline) repeat('+',timeline)//' timestep ',timestep,' starting
'//repeat('+',totalline-timeline)
    if (convergence_details_file) then
      write(fconverge,fmt=formatline) repeat('+',timeline)//' timestep ',timestep,'
starting '//repeat('+',totalline-timeline)
      call flush(fconverge)
    end if
    call time_process
    call update_and_check_transients(ierror=ierror)
    call time_process(description='start of timestep update and check transients')
    if (ierror /= 0) then
      write(*,'(a)') 'ERROR: problem completing update_and_check_transients'
      exit time_loop
    end if
    newtstep = 0  ! only reset this for transient simulations, as may be required to
carry-on from old newtstep for steady-state simulations - now resetting is delayed to
allow saving in a transient
    if (newtient_simulation) then
      call time_process
      call update_and_check_initial_newtients(ierror=ierror)
      call time_process(description='start of timestep update and check initial
newtients')
      if (ierror /= 0) then
        write(*,'(a)') 'ERROR: problem completing update_and_check_initial_newtients'
        exit time_loop
      end if
    end if
    call time_process
    call update_and_check_derived_and_equations(ierror=ierror)
    call time_process(description='start of timestep update and check derived and
equations')
    if (ierror /= 0) then
      write(*,'(a)') 'ERROR: problem completing update_and_check_derived_and_equations'
      exit time_loop
    end if
  end if

  if (trim(output_step_file) == "newtstep") call output_step(action="write")

! dump solution starting point if newtstepout is set to 1 or dumpnewt is found
  if (check_dumpfile("dumpnewt").or.newtstepout /= 0) then
    write(*,'(a)') 'INFO: user has requested output via a dump file or newtstepout
specification'
    call time_process
    call output(intermediate=.true.)
    call output_step(action="write",do_update_outputs=.false.)
    call time_process(description='output')
  end if

!---------------------
! newton loop

  newtconverged = .false.
  if (newtres <= newtrestol) newtconverged = .true.
  if (.not.newtconverged) then
    if (check_condition("convergence")) newtconverged = .true.
  end if

  newt_loop: do while (((.not.newtconverged.and.newtstep < newtstepmax).or. &
      newtstep < newtstepmin).and.ierror == 0)
    newtstep = newtstep + 1

    formatline = "(a,"//trim(dindexformat(newtstep))//",a)"
    write(*,fmt=formatline) repeat('+',newtline)//' newtstep ',newtstep,' starting
'//repeat('+',totalline-newtline)
    if (convergence_details_file) then
      write(fconverge,fmt=formatline) repeat('+',newtline)//' newtstep ',newtstep,'
starting '//repeat('+',totalline-newtline)
      call flush(fconverge)
    end if

! calculate and check on the equation magnitudes
    call time_process
    call update_magnitudes(ierror)
    call time_process(description='start of newtstep calculating variable magnitudes')
    if (ierror /= 0) then
      write(*,'(a)') 'ERROR: problem completing update_magnitudes'
      exit newt_loop
    end if

! calculate the latest residual, based on the new variable magnitudes
    call time_process
    call residual(ierror=ierror)
    call time_process(description='start of newtstep calculating residual')
    if (ierror /= 0) then
      write(*,'(a)') 'ERROR: problem completing residual calculation'
      exit newt_loop
    end if
    write(*,'(a,g10.3,a)') "INFO: initial newton loop newtres = ",newtres," after updatin
variable magnitudes"
    if (convergence_details_file) write(fconverge,'(a,g16.9,a)') &
      "INFO: initial newton loop newtres = ",newtres," after updating variable magnitudes

    if (newtconverged.and.newtstep > newtstepmin) then
      write(*,'(a,g10.3,a)') "INFO: skipping newtsolver as newtres/newtrestol =
",newtres/newtrestol," using existing unknowns"
      if (convergence_details_file) write(fconverge,'(a,g10.3,a)') "INFO: skipping
newtsolver as newtres/newtrestol = ", &
        newtres/newtrestol," using existing unknowns"
    else if (ptotal == 0) then
      write(*,'(a)') 'INFO: skipping newtsolver as no equations are being solved'
      if (convergence_details_file) write(fconverge,'(a)') 'INFO: skipping netsolver as n
equations are being solved'
    else
      call newtsolver(ierror) ! uses newton's method to solve equations - assumes update
has been done and that there is valid magnitudes and a newtres
    end if

! if there is a problem in the newton loop (including a stop file prior to convergence),
then exit newton loop here
    if (ierror /= 0) then
      write(*,'(a)') 'ERROR: problem completing newtsolver'
      exit newt_loop
    end if

! update any newtient variables if this is a newtient simulation
    if (newtient_simulation) then
      formatline = "(a,"//trim(dindexformat(newtstep))//",a,g10.3,a,g10.3)"
      write(*,fmt=formatline) 'INFO: during newton loop before newtient updates: newtstep
= ',newtstep,': newtres = ',newtres, &
        ': newtres/newtrestol = ',newtres/newtrestol
      if (convergence_details_file) then
        formatline = "(a,"//trim(dindexformat(newtstep))//",a,g16.9,a,g10.3)"
        write(fconverge,fmt=formatline) &
          'INFO: during newton loop before newtient updates: newtstep = ',newtstep,':
newtres = ',newtres, &
          ': newtres/newtrestol = ',newtres/newtrestol
      end if
      call time_process
      call update_and_check_newtients(ierror=ierror)
      call time_process(description='intermediate newton step update and check newtients'
      if (ierror /= 0) then
        write(*,'(a)') 'ERROR: problem completing update_and_check_newtients in newtient
update section'
        exit newt_loop
      end if
      call time_process
      call update_and_check_derived_and_equations(ierror=ierror)
      call time_process(description='intermediate newton step update and check derived an
equations after newtient update')
      if (ierror /= 0) then
        write(*,'(a)') 'ERROR: problem completing update_and_check_derived_and_equations
in newtient update section'
        exit newt_loop
      end if
      call residual(ierror=ierror)
```

```
   if (ierror /= 0) then
    write(*,'(a)') 'ERROR: problem calculating residual in newtient update section'
    exit newt_loop
   end if
  end if

  if (trim(output_step_file) == "newtstep") call output_step(action="write")

! also start writing output files is newtstep >= newtstepdebugout

  if (check_dumpfile("dumpnewt").or.(newtstepout /=
0.and.mod(newtstep,max(newtstepout,1)) == 0).or.newtstep >= newtstepdebugout) then
   write(*,'(a)') 'INFO: user has requested output via a dump file or newtstepout
specification'
   call time_process
   call output(intermediate=.true.)
   call output_step(action="write",do_update_outputs=.false.)
   call time_process(description='output')
  end if

  if (transient_simulation) then
   formatline = "
(a,"//trim(dindexformat(newtstep))//",a,"//trim(dindexformat(timestep))//",a,g10.3,a,g10.
)"
   write(*,fmt=formatline) 'INFO: during newton loop: newtstep = ',newtstep,': timeste
= ',timestep,': newtres = ',newtres, &
     ': newtres/newtrestol = ',newtres/newtrestol
   if (convergence_details_file) then
    formatline = "
(a,"//trim(dindexformat(newtstep))//",a,"//trim(dindexformat(timestep))//",a,g16.9,a,g10.
)"
    write(fconverge,fmt=formatline) &
     'INFO: during newton loop: newtstep = ',newtstep,': timestep = ',timestep,':
newtres = ',newtres, &
     ': newtres/newtrestol = ',newtres/newtrestol
   end if
  else
   formatline = "(a,"//trim(dindexformat(newtstep))//",a,g10.3,a,g10.3)"
   write(*,fmt=formatline) 'INFO: during newton loop: newtstep = ',newtstep,': newtres
= ',newtres,': newtres/newtrestol = ', &
    newtres/newtrestol
   if (convergence_details_file) then
    formatline = "(a,"//trim(dindexformat(newtstep))//",a,g16.9,a,g10.3)"
    write(fconverge,fmt=formatline) &
     'INFO: during newton loop: newtstep = ',newtstep,': newtres = ',newtres,':
newtres/newtrestol = ',newtres/newtrestol
   end if
  end if
  if (convergence_details_file) call flush(fconverge)

! check whether solution is converged
  if (newtres <= newtrestol) newtconverged = .true.
  if (.not.newtconverged) then
   if (check_condition("convergence")) newtconverged = .true.
  end if

! only check for stopfile if output isn't converged
  if (.not.newtconverged) then
   if (check_stopfile("stopnewt")) then
    write(*,'(a)') 'INFO: user has requested simulation stop via a stop file'
    ierror = -1 ! negative ierror indicates that user stopped arb before convergence
complete
   end if
  end if

  formatline = "(a,"//trim(dindexformat(newtstep))//",a)"
  write(*,fmt=formatline) repeat('-',newtline)//' newtstep ',newtstep,' ending
'//repeat('-',totalline-newtline+2)
  if (convergence_details_file) then
   write(fconverge,fmt=formatline) repeat('-',newtline)//' newtstep ',newtstep,' endin
'//repeat('-',totalline-newtline+2)
   call flush(fconverge)
  end if

 end do newt_loop
!----------

 if (ierror > 0) then
  formatline = "(a,"//trim(dindexformat(ierror))//")"
  write(*,fmt=formatline) 'ERROR: problem in some solution routine within newton loop:
error number = ',ierror
  exit time_loop
 else if (ierror < 0) then
  write(*,'(a)') 'ERROR: newton solver did not converge due to user created stop file'
  exit time_loop
 else if (newtconverged) then
```

```fortran
    if (newtres <= newtrestol) then
      write(*,'(a)') 'INFO: newton iterations have converged due to newtres condition'
    else
      write(*,'(a)') &
        'INFO: user-specified newton loop convergence condition satisfied'
    end if
  else
    write(*,'(a)') 'ERROR: newton solver did not converge'
    ierror = 5
    exit time_loop
  end if

! if user has requested to halt then write message
  if (transient_simulation.and.check_stopfile("stoptime")) write(*,'(a)') &
    'INFO: user has requested simulation stop via a stop file'

! silly bell functionality!
  if (check_condition("bell")) call ring_bell

! write output if output is due, or we are finishing
  if ((transient_simulation.and.(check_condition("output").or.(timestepout /=
0.and.mod(timestep,max(timestepout,1)) == 0).or. &
    check_condition("stop").or.timestep >=
timestepmax.or.check_stopfile("stoptime").or.check_dumpfile("dumptime"))).or. &
    .not.transient_simulation) then
    if (check_dumpfile("dumptime")) write(*,'(a)') 'INFO: user has requested output via a
dump file'
    call time_process
    if (output_timings.and.output_timings_on_mesh_write.and.(timestepout /=
0.and.mod(timestep,max(timestepout,1)) == 0)) &
      write(*,'(2(a,g10.3))') 'TIMING: total wall time = ',total_wall_time,': total cpu
time = ',total_cpu_time
    call output
    if (trim(output_step_file) == "timestep") call
output_step(action="write",do_update_outputs=.false.)
    call time_process(description='output')
  else
    if (trim(output_step_file) == "timestep") call output_step(action="write")
  end if

  if (transient_simulation) then
    formatline = "(a,"//trim(dindexformat(timestep))//",a)"
    write(*,fmt=formatline) repeat('-',timeline)//' timestep ',timestep,' ending
'//repeat('-',totalline-timeline+2)

    if (convergence_details_file) then
      write(fconverge,fmt=formatline) repeat('-',timeline)//' timestep ',timestep,' endin
'//repeat('-',totalline-timeline+2)
      call flush(fconverge)
    end if
  end if

! if not a transient simulation then exit loop
  if (.not.transient_simulation) exit time_loop

end do time_loop
!-----------------------------------------

if (trim(output_step_file) == "final") call output_step(action="write")

if (output_timings) write(*,'(2(a,g10.3))') 'TIMING: total wall time =
',total_wall_time,': total cpu time = ',total_cpu_time

! if there was an error or earlier stop requested then exit without closing timestep
if (ierror /= 0) then
  write(*,'(a)') "WARNING: the last output is not converged"
  write(*,'(a)') 'INFO: a debug output file (debug.output.msh) is being written that
contains the current values of '// &
    'all variable components'
  call output(debug_dump=.true.)
  if (trim(output_step_file) == "timestep") call
output_step(action="write",do_update_outputs=.false.)
  write(*,'(a)') "ERROR: the simulation was not successful"
else
  write(*,'(a)') "SUCCESS: the simulation finished gracefully"
end if

if (convergence_details_file) close(fconverge)
call output_step(action="close")

if (ierror /= 0) call exit(ierror) ! exit while setting ierror as exit status

end program arb

!-------------------------------------------------------------------
```