Transient Response Analysis for Fault Detection and Pipeline Wall Condition Assessment in Field Water Transmission and Distribution Pipelines and Networks

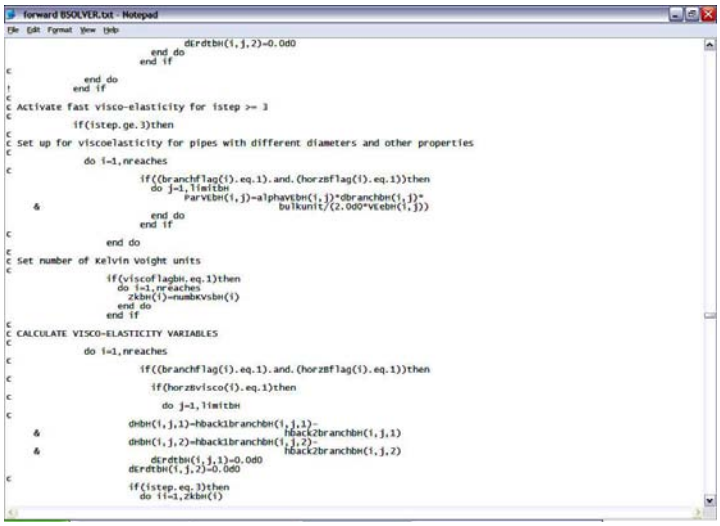
by

Mark Leslie Stephens

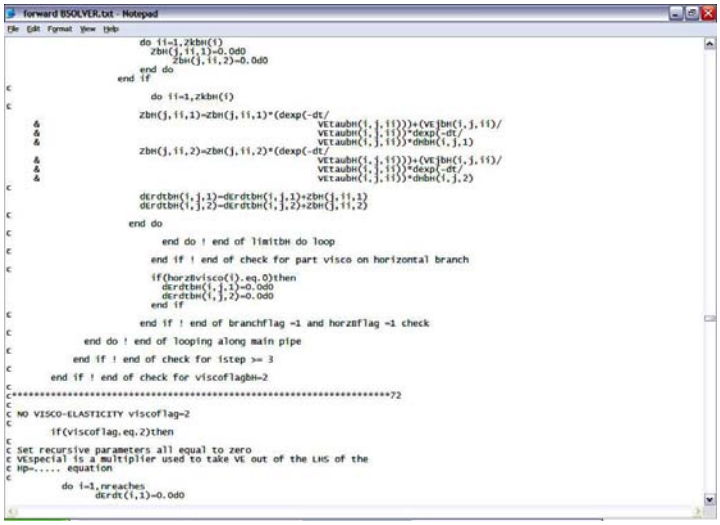
February 2008

A Thesis Submitted for the Degree of Doctor of Philosophy

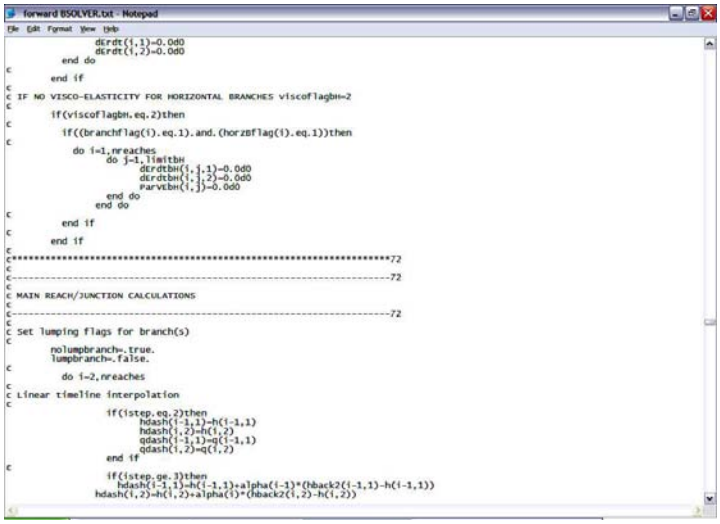
School of Civil and Environmental Engineering
The University of Adelaide, SA 5005
South Australia



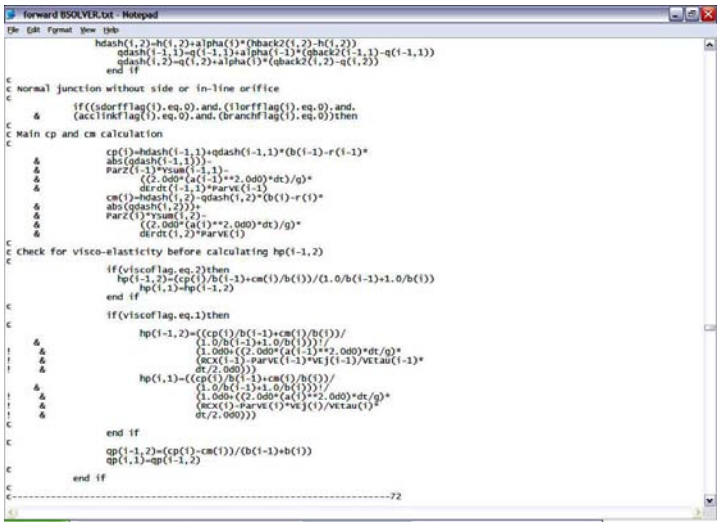
```
forward BSOLVER.txt - Notepad
File  Edit  Format  View  Help
                                dErdtbH(i,j,2)=0.0d0
                              end do
                            end if
c
                end do
!             end if
c
c Activate fast visco-elasticity for istep >= 3
c
              if(istep.ge.3)then
c
c Set up for viscoelasticity for pipes with different diameters and other properties
c
                do i=1,nreaches
c
                        if((branchflag(i).eq.1).and.(horzBflag(i).eq.1))then
                          do j=1,limitbH
                              ParvEbH(i,j)=alphavEbH(i,j)*dbranchbH(i,j)*
     &                                          bulkunit/(2.0d0*vEbH(i,j))
                          end do
                        end if
c
                    end do
c
c Set number of Kelvin Voight units
c
                    if(viscoflagbH.eq.1)then
                      do i=1,nreaches
                        zkbH(i)=numbKvsbH(i)
                      end do
                    end if
c
c CALCULATE VISCO-ELASTICITY VARIABLES
c
                do i=1,nreaches
c
                        if((branchflag(i).eq.1).and.(horzBflag(i).eq.1))then
c
                          if(horzBvisco(i).eq.1)then
c
                            do j=1,limitbH
c
                          dHbH(i,j,1)=hback1branchbH(i,j,1)-
     &                                                  hback2branchbH(i,j,1)
                          dHbH(i,j,2)=hback1branchbH(i,j,2)-
     &                                                  hback2branchbH(i,j,2)
                            dErdtbH(i,j,1)=0.0d0
                          dErdtbH(i,j,2)=0.0d0
c
                          if(istep.eq.3)then
                            do ii=1,zkbH(i)
```



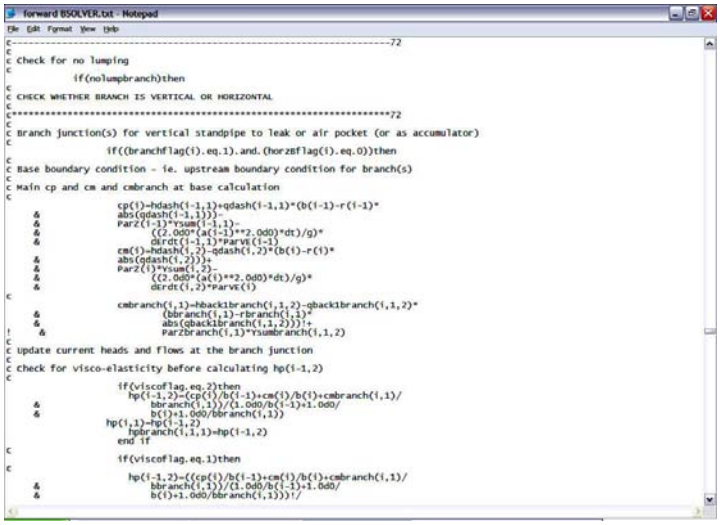
```
forward BSOLVER.txt - Notepad
File  Edit  Format  View  Help
                            do ii=1,zkbH(i)
                              zbH(j,ii,1)=0.0d0
                              zbH(j,ii,2)=0.0d0
                            end do
                          end if
c
                            do ii=1,zkbH(i)
c
                          zbH(j,ii,1)=zbH(j,ii,1)*(dexp(-dt/
     &                                              vEtaubH(i,j,ii)))+(vEjbH(i,j,ii)/
     &                                              vEtaubH(i,j,ii))*dexp(-dt/
     &                                              vEtaubH(i,j,ii))*dHbH(i,j,1)
                          zbH(j,ii,2)=zbH(j,ii,2)*(dexp(-dt/
     &                                              vEtaubH(i,j,ii)))+(vEjbH(i,j,ii)/
     &                                              vEtaubH(i,j,ii))*dexp(-dt/
     &                                              vEtaubH(i,j,ii))*dHbH(i,j,2)
c
                          dErdtbH(i,j,1)=dErdtbH(i,j,1)+zbH(j,ii,1)
                          dErdtbH(i,j,2)=dErdtbH(i,j,2)+zbH(j,ii,2)
c
                        end do
c
                             end do ! end of limitbH do loop
c
                          end if ! end of check for part visco on horizontal branch
c
                          if(horzBvisco(i).eq.0)then
                            dErdtbH(i,j,1)=0.0d0
                            dErdtbH(i,j,2)=0.0d0
                          end if
c
                        end if ! end of branchflag =1 and horzBflag =1 check
c
              end do ! end of looping along main pipe
c
            end if ! end of check for istep >= 3
c
          end if ! end of check for viscoflagbH=2
c
c*********************************************************************72
c
c NO VISCO-ELASTICITY viscoflag=2
c
          if(viscoflag.eq.2)then
c
c Set recursive parameters all equal to zero
c VEspecial is a multiplier used to take VE out of the LHS of the
c Hp=..... equation
c
            do i=1,nreaches
                  dErdt(i,1)=0.0d0
```



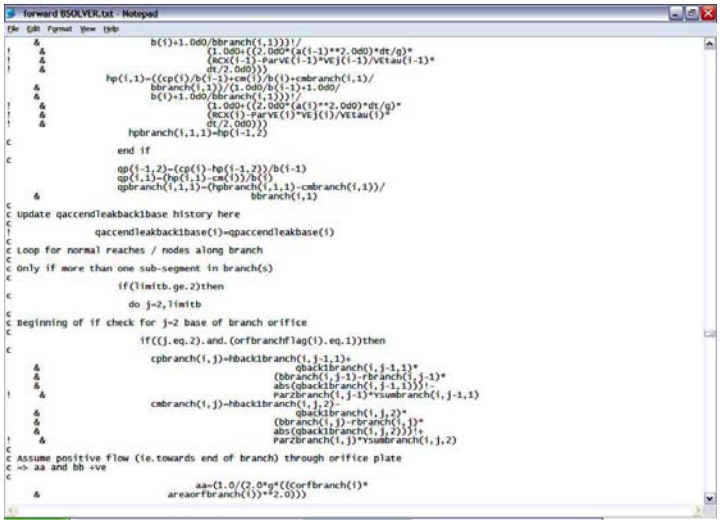
```
forward BSOLVER.txt - Notepad
File  Edit  Format  View  Help
                  dErdt(i,1)=0.0d0
                  dErdt(i,2)=0.0d0
              end do
c
            end if
c
c IF NO VISCO-ELASTICITY FOR HORIZONTAL BRANCHES viscoflagbH=2
c
            if(viscoflagbH.eq.2)then
c
              if((branchflag(i).eq.1).and.(horzBflag(i).eq.1))then
c
                do i=1,nreaches
                      do j=1,limitbH
                        dErdtbH(i,j,1)=0.0d0
                        dErdtbH(i,j,2)=0.0d0
                        ParvEbH(i,j)=0.0d0
                      end do
                    end do
c
              end if
c
            end if
c
c*********************************************************************72
c
c-------------------------------------------------------------------72
c
c MAIN REACH/JUNCTION CALCULATIONS
c
c-------------------------------------------------------------------72
c
c Set lumping flags for branch(s)
c
            nolumpbranch=.true.
            lumpbranch=.false.
c
              do i=2,nreaches
c
c Linear timeline interpolation
c
                    if(istep.eq.2)then
                          hdash(i-1,1)=h(i-1,1)
                          hdash(1,2)=h(1,2)
                          qdash(i-1,1)=q(i-1,1)
                          qdash(1,2)=q(1,2)
                    end if
c
                    if(istep.ge.3)then
                      hdash(i-1,1)=h(i-1,1)+alpha(i-1)*(hback2(i-1,1)-h(i-1,1))
                    hdash(1,2)=h(1,2)+alpha(1)*(hback2(1,2)-h(1,2))
```



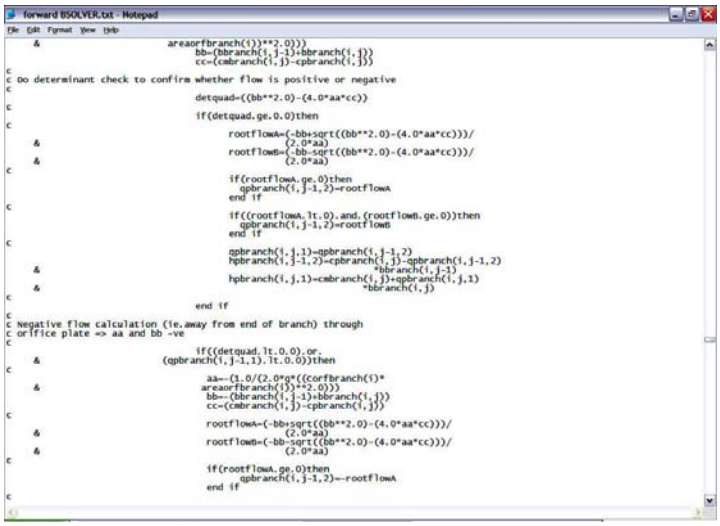

```
                  hdash(1,2)=h(1,2)+alpha(1)*(hback2(1,2)-h(1,2))
                     qdash(1-1,1)=q(1-1,1)+alpha(1-1)*(qback2(1-1,1)-q(1-1,1))
                     qdash(1,2)=q(1,2)+alpha(1)*(qback2(1,2)-q(1,2))
                  end if
C
C Normal junction without side or in-line orifice
C
            if((sdorfflag(i).eq.0).and.(ilorfflag(i).eq.0).and.
     &      (acclinkflag(i).eq.0).and.(branchflag(i).eq.0))then
C
C Main cp and cm calculation
C
                  cp(i)=hdash(i-1,1)+qdash(i-1,1)*(b(i-1)-r(i-1)*
     &            abs(qdash(i-1,1)))-
     &            Parz(i-1)*Ysum(i-1,1)-
     &                  ((2.0d0*(a(i-1)**2.0d0)*dt)/g)*
     &                  dkrdt(i-1,1)*ParVe(i-1)
                  cm(i)=hdash(i,2)-qdash(i,2)*(b(i)-r(i)*
     &            abs(qdash(i,2)))+
     &            Parz(i)*Ysum(i,2)-
     &                  ((2.0d0*(a(i)**2.0d0)*dt)/g)*
     &                  dkrdt(i,2)*ParVe(i)
C
C Check for visco-elasticity before calculating hp(i-1,2)
C
                  if(viscoflag.eq.2)then
                    hp(i-1,2)=(cp(i)/b(i-1)+cm(i)/b(i))/(1.0/b(i-1)+1.0/b(i))
                        hp(1,1)=hp(i-1,2)
                  end if
C
                  if(viscoflag.eq.1)then
C
                        hp(i-1,2)=((cp(i)/b(i-1)+cm(i)/b(i))/
     &                        (1.0/b(i-1)+1.0/b(i)))/
!    &                        (1.0d0+((2.0d0*(a(i-1)**2.0d0)*dt/g)*
!    &                        (RCx(i-1)-ParVe(i-1)*VEj(i-1)/VEtau(i-1)*
!    &                        dt/2.0d0)))
                        hp(i,1)=((cp(i)/b(i-1)+cm(i)/b(i))/
     &                        (1.0/b(i-1)+1.0/b(i)))/
!    &                        (1.0d0+((2.0d0*(a(i)**2.0d0)*dt/g)*
!    &                        (RCx(i)-ParVe(i)*VEj(i)/VEtau(i)*
!    &                        dt/2.0d0)))
C
                  end if
C
                  qp(i-1,2)=(cp(i)-cm(i))/(b(i-1)+b(i))
                  qp(1,1)=qp(i-1,2)
C
            end if
C
C-----------------------------------------------------------------72
```



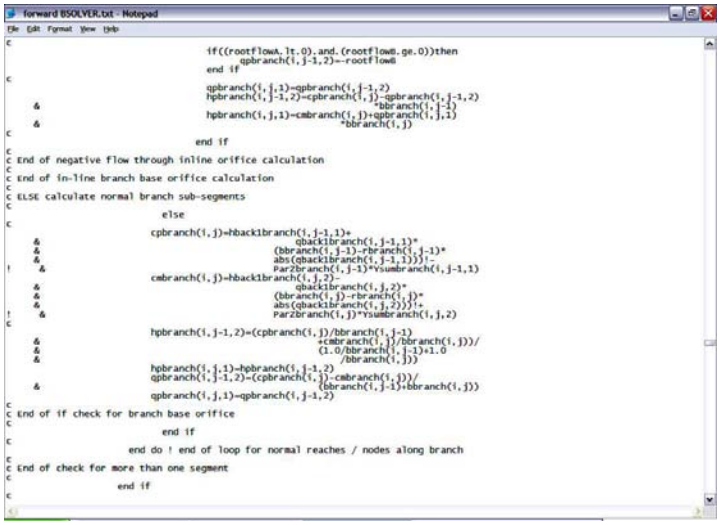

```
C-----------------------------------------------------------------72
C
C Check for no lumping
C
            if(nolumpbranch)then
C
C CHECK WHETHER BRANCH IS VERTICAL OR HORIZONTAL
C
C**********************************************************************72
C
C Branch junction(s) for vertical standpipe to leak or air pocket (or as accumulator)
C
                  if((branchflag(i).eq.1).and.(horzbflag(i).eq.0))then
C
C Base boundary condition - ie. upstream boundary condition for branch(s)
C
C Main cp and cm and cmbranch at base calculation
C
                     cp(i)=hdash(i-1,1)+qdash(i-1,1)*(b(i-1)-r(i-1)*
     &               abs(qdash(i-1,1)))-
     &               Parz(i-1)*Ysum(i-1,1)-
     &                     ((2.0d0*(a(i-1)**2.0d0)*dt)/g)*
     &                     dkrdt(i-1,1)*ParVe(i-1)
                     cm(i)=hdash(i,2)-qdash(i,2)*(b(i)-r(i)*
     &               abs(qdash(i,2)))+
     &               Parz(i)*Ysum(i,2)-
     &                     ((2.0d0*(a(i)**2.0d0)*dt)/g)*
     &                     dkrdt(i,2)*ParVe(i)
C
                     cmbranch(i,1)=hback1branch(i,1,2)-qback1branch(i,1,2)*
     &                     (bbranch(i,1)-rbranch(i,1)*
     &                      abs(qback1branch(i,1,2)))+
!    &                      Parzbranch(i,1)*Ysumbranch(i,1,2)
C
C Update current heads and flows at the branch junction
C
C Check for visco-elasticity before calculating hp(i-1,2)
C
                     if(viscoflag.eq.2)then
                       hp(i-1,2)=(cp(i)/b(i-1)+cm(i)/b(i)+cmbranch(i,1)/
     &                     bbranch(i,1))/(1.0d0/b(i-1)+1.0d0/
     &                     b(i)+1.0d0/bbranch(i,1))
                     hp(i,1)=hp(i-1,2)
                       hpbranch(i,1,1)=hp(i-1,2)
                     end if
C
                     if(viscoflag.eq.1)then
C
                       hp(i-1,2)=((cp(i)/b(i-1)+cm(i)/b(i)+cmbranch(i,1)/
     &                     bbranch(i,1))/(1.0d0/b(i-1)+1.0d0/
     &                     b(i)+1.0d0/bbranch(i,1)))/
```



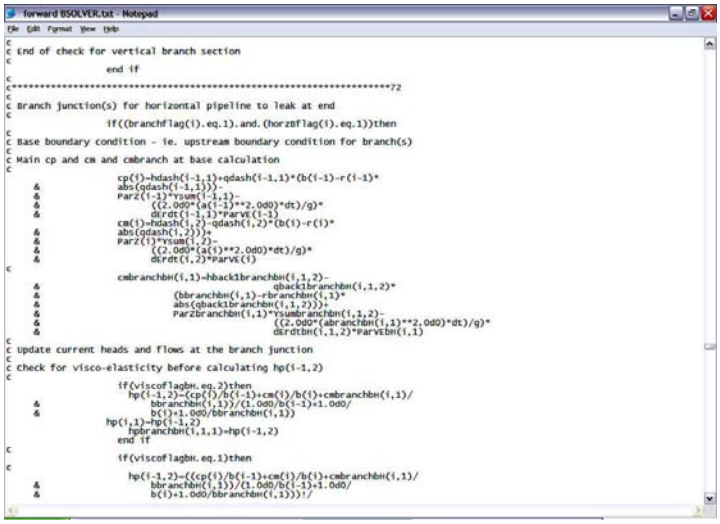

```
     &                        b(i)+1.0d0/bbranch(i,1)))/
!    &                        (1.0d0+((2.0d0*(a(i-1)**2.0d0)*dt/g)*
!    &                        (RCx(i-1)-ParVE(i-1)*VEj(i-1)/VEtau(i-1)*
!    &                        dt/2.0d0)))
                     hp(i,1)=((cp(i)/b(i-1)+cm(i)/b(i)+cmbranch(i,1)/
     &                     bbranch(i,1))/(1.0d0/b(i-1)+1.0d0/
     &                     b(i)+1.0d0/bbranch(i,1)))/
!    &                        (1.0d0+((2.0d0*(a(i)**2.0d0)*dt/g)*
!    &                        (RCx(i)-ParVE(i)*VEj(i)/VEtau(i)*
!    &                        dt/2.0d0)))
                       hpbranch(i,1,1)=hp(i-1,2)
C
                     end if
C
                     qp(i-1,2)=(cp(i)-hp(i-1,2))/b(i-1)
                     qp(i,1)=(hp(i,1)-cm(i))/b(i)
                     qpbranch(i,1,1)=(hpbranch(i,1,1)-cmbranch(i,1))/
     &                                    bbranch(i,1)
C
C Update qaccendleakback1base history here
C
!                 qaccendleakback1base(i)=qpaccendleakbase(i)
C
C Loop for normal reaches / nodes along branch
C
C Only if more than one sub-segment in branch(s)
C
                     if(limitb.ge.2)then
C
                       do j=2,limitb
C
C Beginning of if check for j=2 base of branch orifice
C
                     if((j.eq.2).and.(orfbranchflag(i).eq.1))then
C
                       cpbranch(i,j)=hback1branch(i,j-1,1)+
     &                                    qback1branch(i,j-1,1)*
     &                                 (bbranch(i,j-1)-rbranch(i,j-1)*
     &                                 abs(qback1branch(i,j-1,1)))-
!    &                                 Parzbranch(i,j-1)*Ysumbranch(i,j-1,1)
                       cmbranch(i,j)=hback1branch(i,j,2)-
     &                                    qback1branch(i,j,2)*
     &                                 (bbranch(i,j)-rbranch(i,j)*
     &                                 abs(qback1branch(i,j,2)))+
!    &                                 Parzbranch(i,j)*Ysumbranch(i,j,2)
C
C Assume positive flow (ie.towards end of branch) through orifice plate
C => aa and bb +ve
C
                                 aa=(1.0/(2.0*g*((corfbranch(i)*
     &                           areaorfbranch(i))**2.0)))
```



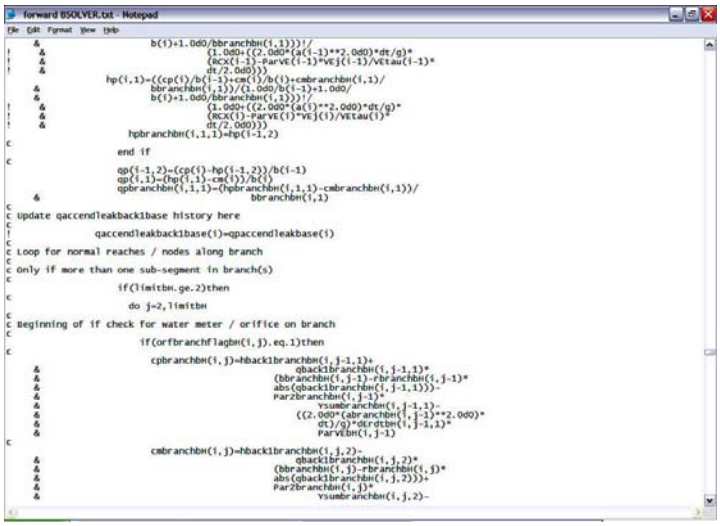
```
forward BSOLVER.txt - Notepad
File  Edit  Format  View  Help
     &                        areaorfbranch(i))**2.0)))
                                    bb=(bbranch(i,j-1)+bbranch(i,j))
                                    cc=(cmbranch(i,j)-cpbranch(i,j))
c
c Do determinant check to confirm whether flow is positive or negative
c
                                    detquad=((bb**2.0)-(4.0*aa*cc))
c
                                    if(detquad.ge.0.0)then
c
                                          rootflowA=(-bb+sqrt((bb**2.0)-(4.0*aa*cc)))/
     &                                                (2.0*aa)
                                          rootflowB=(-bb-sqrt((bb**2.0)-(4.0*aa*cc)))/
     &                                                (2.0*aa)
c
                                          if(rootflowA.ge.0)then
                                            qpbranch(i,j-1,2)=rootflowA
                                          end if
c
                                          if((rootflowA.lt.0).and.(rootflowB.ge.0))then
                                            qpbranch(i,j-1,2)=rootflowB
                                          end if
c
                                          qpbranch(i,j,1)=qpbranch(i,j-1,2)
                                          hpbranch(i,j-1,2)=cpbranch(i,j)-qpbranch(i,j-1,2)
     &                                                      *bbranch(i,j-1)
                                          hpbranch(i,j,1)=cmbranch(i,j)+qpbranch(i,j,1)
     &                                                    *bbranch(i,j)
c
                                    end if
c
c Negative flow calculation (ie.away from end of branch) through
c orifice plate => aa and bb -ve
c
                                      if((detquad.lt.0.0).or.
     &                             (qpbranch(i,j-1,1).lt.0.0))then
c
                                       aa=-(1.0/(2.0*g*((coefbranch(i)*
     &                                areaorfbranch(i))**2.0)))
                                       bb=-(bbranch(i,j-1)+bbranch(i,j))
                                       cc=(cmbranch(i,j)-cpbranch(i,j))
c
                                       rootflowA=(-bb+sqrt((bb**2.0)-(4.0*aa*cc)))/
     &                                                (2.0*aa)
                                       rootflowB=(-bb-sqrt((bb**2.0)-(4.0*aa*cc)))/
     &                                                (2.0*aa)
c
                                       if(rootflowA.ge.0)then
                                          qpbranch(i,j-1,2)=-rootflowA
                                       end if
c
```



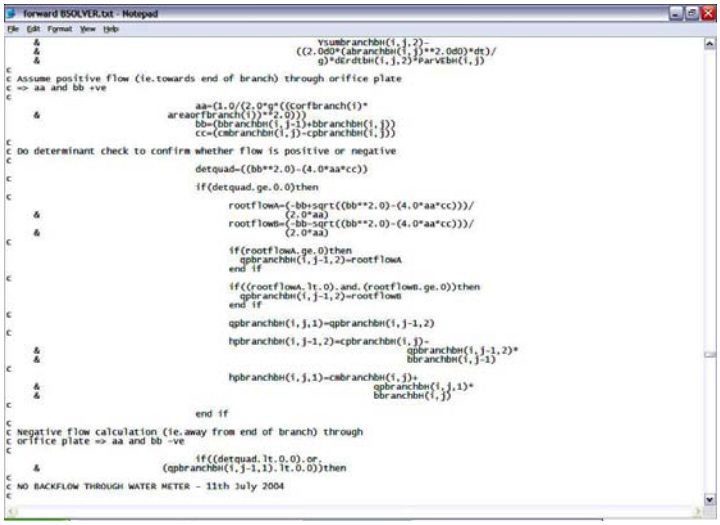
```
forward BSOLVER.txt - Notepad
File  Edit  Format  View  Help
c
                                    if((rootflowA.lt.0).and.(rootflowB.ge.0))then
                                          qpbranch(i,j-1,2)=-rootflowB
                                    end if
c
                                    qpbranch(i,j,1)=qpbranch(i,j-1,2)
                                    hpbranch(i,j-1,2)=cpbranch(i,j)-qpbranch(i,j-1,2)
     &                                                *bbranch(i,j-1)
                                    hpbranch(i,j,1)=cmbranch(i,j)+qpbranch(i,j,1)
     &                                                *bbranch(i,j)
c
                                  end if
c
c End of negative flow through inline orifice calculation
c
c End of in-line branch base orifice calculation
c
c ELSE calculate normal branch sub-segments
c
                                    else
c
                                  cpbranch(i,j)=hback1branch(i,j-1,1)+
     &                                                qback1branch(i,j-1,1)*
     &                                          (bbranch(i,j-1)-rbranch(i,j-1)*
     &                                          abs(qback1branch(i,j-1,1)))-
!    &                                          Parzbranch(i,j-1)*ysumbranch(i,j-1,1)
                                  cmbranch(i,j)=hback1branch(i,j,2)-
     &                                                qback1branch(i,j,2)*
     &                                          (bbranch(i,j)-rbranch(i,j)*
     &                                          abs(qback1branch(i,j,2)))+
!    &                                          Parzbranch(i,j)*ysumbranch(i,j,2)
c
                                  hpbranch(i,j-1,2)=(cpbranch(i,j)/bbranch(i,j-1)
     &                                                +cmbranch(i,j)/bbranch(i,j))/
     &                                                (1.0/bbranch(i,j-1)+1.0
     &                                                  /bbranch(i,j))
                                  hpbranch(i,j,1)=hpbranch(i,j-1,2)
                                  qpbranch(i,j-1,2)=(cpbranch(i,j)-cmbranch(i,j))/
     &                                                (bbranch(i,j-1)+bbranch(i,j))
                                  qpbranch(i,j,1)=qpbranch(i,j-1,2)
c
c End of if check for branch base orifice
c
                                    end if
c
                                end do ! end of loop for normal reaches / nodes along branch
c
c End of check for more than one segment
c
                              end if
c
```



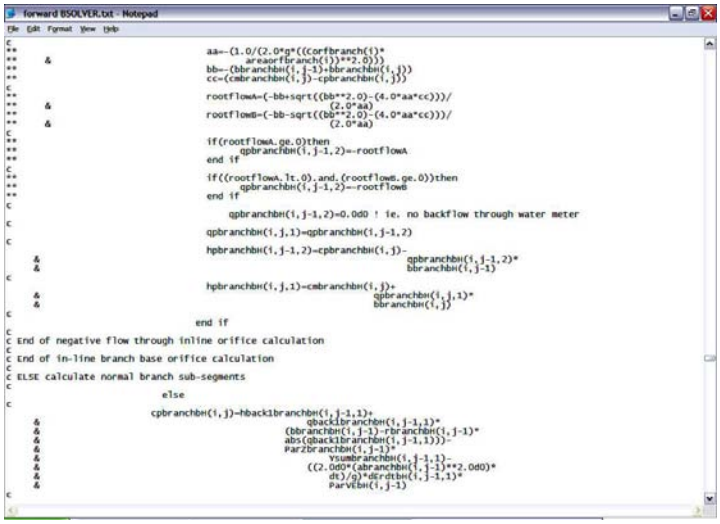
```
forward BSOLVER.txt - Notepad
File  Edit  Format  View  Help
c
c End of check for vertical branch section
c
                              end if
c
c*******************************************************************72
c
c Branch junction(s) for horizontal pipeline to leak at end
c
                              if((branchflag(i).eq.1).and.(horzbflag(i).eq.1))then
c
c Base boundary condition - ie. upstream boundary condition for branch(s)
c
c Main cp and cm and cmbranch at base calculation
c
                              cp(i)=hdash(i-1,1)+qdash(i-1,1)*(b(i-1)-r(i-1)*
     &                        abs(qdash(i-1,1)))-
     &                        Parz(i-1)*ysum(i-1,1)-
     &                              ((2.0d0*(a(i-1)**2.0d0)*dt)/g)*
     &                              dirdt(i-1,1)*Parv(i-1)
                              cm(i)=hdash(i,2)-qdash(i,2)*(b(i)-r(i)*
     &                        abs(qdash(i,2)))+
     &                        Parz(i)*ysum(i,2)-
     &                              ((2.0d0*(a(i)**2.0d0)*dt)/g)*
     &                              dirdt(i,2)*Parv(i)
c
                              cmbranchbh(i,1)=hback1branchbh(i,1,2)-
     &                                                qback1branchbh(i,1,2)*
     &                                    (bbranchbh(i,1)-rbranchbh(i,1)*
     &                                    abs(qback1branchbh(i,1,2)))+
     &                                    Parzbranchbh(i,1)*ysumbranchbh(i,1,2)-
     &                                          ((2.0d0*(abranchbh(i,1)**2.0d0)*dt)/g)*
     &                                          dirdtbh(i,1,2)*Parvbh(i,1)
c
c Update current heads and flows at the branch junction
c
c Check for visco-elasticity before calculating hp(i-1,2)
c
                              if(viscoflagbh.eq.2)then
                                hp(i-1,2)=(cp(i)/b(i-1)+cm(i)/b(i)+cmbranchbh(i,1)/
     &                              bbranchbh(i,1))/(1.0d0/b(i-1)+1.0d0/
     &                              b(i)+1.0d0/bbranchbh(i,1))
                            hp(i,1)=hp(i-1,2)
                              hpbranchbh(i,1,1)=hp(i-1,2)
                              end if
c
                              if(viscoflagbh.eq.1)then
c
                                hp(i-1,2)=((cp(i)/b(i-1)+cm(i)/b(i)+cmbranchbh(i,1)/
     &                              bbranchbh(i,1))/(1.0d0/b(i-1)+1.0d0/
     &                              b(i)+1.0d0/bbranchbh(i,1)))!/
```



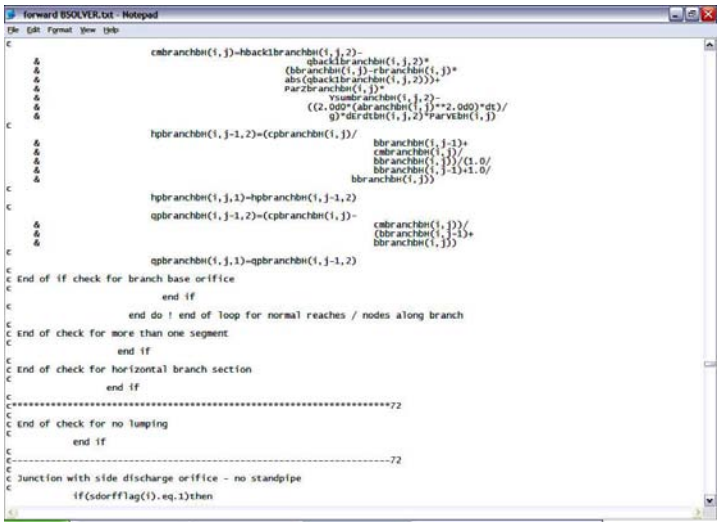
```
forward BSOLVER.txt - Notepad
File  Edit  Format  View  Help
     &                        b(i)+1.0d0/bbranchbn(i,1)))/
!    &                              (1.0d0+((2.0d0*(a(i-1)**2.0d0)*dt/g)*
!    &                              (RCX(i-1)-ParVE(i-1)*VEj(i-1)/VEtau(i-1)*
!    &                              dt/2.0d0)))
                    hp(i,1)=((cp(i)/b(i-1)+cm(i)/b(i)+cmbranchbn(i,1)/
     &                        bbranchbn(i,1))/(1.0d0/b(i-1)+1.0d0/
     &                        b(i)+1.0d0/bbranchbn(i,1)))/
!    &                              (1.0d0+((2.0d0*(a(i)**2.0d0)*dt/g)*
!    &                              (RCX(i)-ParVE(i)*VEj(i)/VEtau(i)*
!    &                              dt/2.0d0)))
                      hpbranchbn(i,1,1)=hp(i-1,2)
C
                     end if
C
                     qp(i-1,2)=(cp(i)-hp(i-1,2))/b(i-1)
                     qp(i,1)=(hp(i,1)-cm(i))/b(i)
                     qpbranchbn(i,1,1)=(hpbranchbn(i,1,1)-cmbranchbn(i,1))/
     &                                      bbranchbn(i,1)
C
C Update qaccendleakback1base history here
C
                qaccendleakback1base(i)=qaccendleakbase(i)
C
C Loop for normal reaches / nodes along branch
C
C Only if more than one sub-segment in branch(s)
C
                     if(limitbn.ge.2)then
C
                       do j=2,limitbn
C
C Beginning of if check for water meter / orifice on branch
C
                      if(orfbranchflagbn(i,j).eq.1)then
C
                        cpbranchbn(i,j)=hback1branchbn(i,j-1,1)+
     &                                          qback1branchbn(i,j-1,1)*
     &                                    (bbranchbn(i,j-1)-rbranchbn(i,j-1)*
     &                                    abs(qback1branchbn(i,j-1,1)))-
     &                                    Par2branchbn(i,j-1)*
     &                                        ysumbranchbn(i,j-1,1)-
     &                                        ((2.0d0*(abranchbn(i,j-1)**2.0d0)*
     &                                          dt)/g)*dErdtbn(i,j-1,1)*
     &                                          ParVEbn(i,j-1)
C
                      cmbranchbn(i,j)=hback1branchbn(i,j,2)-
     &                                          qback1branchbn(i,j,2)*
     &                                    (bbranchbn(i,j)-rbranchbn(i,j)*
     &                                    abs(qback1branchbn(i,j,2)))+
     &                                    Par2branchbn(i,j)*
     &                                        ysumbranchbn(i,j,2)-
```



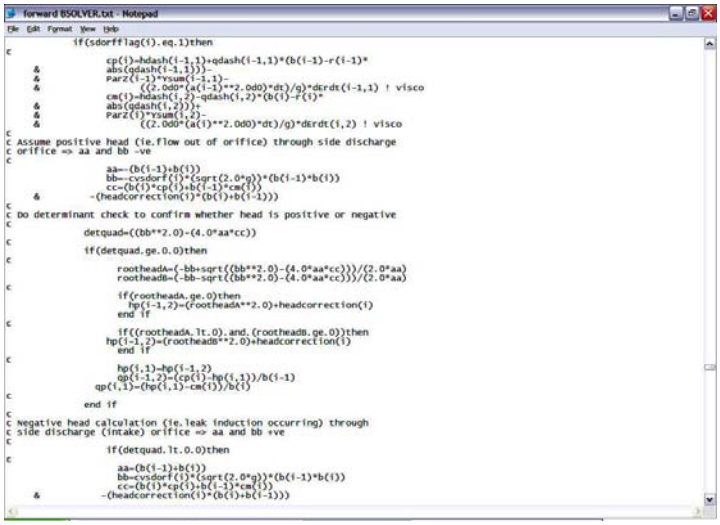
```
forward BSOLVER.txt - Notepad
File  Edit  Format  View  Help
     &                                        ysumbranchbn(i,j,2)-
     &                                  ((2.0d0*(abranchbn(i,j)**2.0d0)*dt)/
     &                                    g)*dErdtbn(i,j,2)*ParVEbn(i,j)
C
C Assume positive flow (ie.towards end of branch) through orifice plate
C => aa and bb +ve
C
                              aa=(1.0/(2.0*g*((corfbranch(i)*
     &                      areaorfbranch(i))**2.0)))
                              bb=(bbranchbn(i,j-1)+bbranchbn(i,j))
                              cc=(cmbranchbn(i,j)-cpbranchbn(i,j))
C
C Do determinant check to confirm whether flow is positive or negative
C
                              detquad=((bb**2.0)-(4.0*aa*cc))
C
                              if(detquad.ge.0.0)then
C
                                   rootflowA=(-bb+sqrt((bb**2.0)-(4.0*aa*cc)))/
     &                                        (2.0*aa)
                                   rootflowB=(-bb-sqrt((bb**2.0)-(4.0*aa*cc)))/
     &                                        (2.0*aa)
C
                                   if(rootflowA.ge.0)then
                                     qpbranchbn(i,j-1,2)=rootflowA
                                   end if
C
                                   if((rootflowA.lt.0).and.(rootflowB.ge.0))then
                                     qpbranchbn(i,j-1,2)=rootflowB
                                   end if
C
                                   qpbranchbn(i,j,1)=qpbranchbn(i,j-1,2)
C
                                   hpbranchbn(i,j-1,2)=cpbranchbn(i,j)-
     &                                                  qpbranchbn(i,j-1,2)*
     &                                                  bbranchbn(i,j-1)
C
                                   hpbranchbn(i,j,1)=cmbranchbn(i,j)+
     &                                              qpbranchbn(i,j,1)*
     &                                              bbranchbn(i,j)
C
                              end if
C
C Negative flow calculation (ie.away from end of branch) through
C orifice plate => aa and bb -ve
C
                                   if((detquad.lt.0.0).or.
     &                          (qpbranchbn(i,j-1,1).lt.0.0))then
C
C NO BACKFLOW THROUGH WATER METER - 11th July 2004
C
```



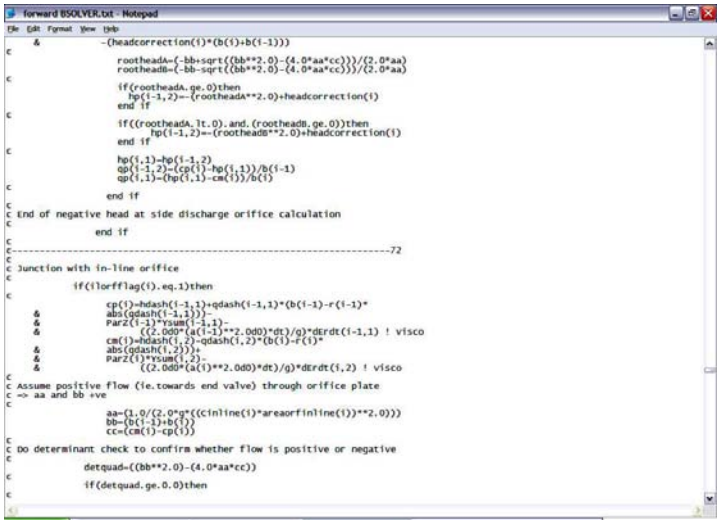
```
forward BSOLVER.txt - Notepad
File  Edit  Format  View  Help
C
**                                 aa=-(1.0/(2.0*g*((corfbranch(i)*
**     &                                 areaorfbranch(i))**2.0)))
**                                 bb=-(bbranchbn(i,j-1)+bbranchbn(i,j))
**                                 cc=-(cmbranchbn(i,j)-cpbranchbn(i,j))
**
**                                 rootflowA=(-bb+sqrt((bb**2.0)-(4.0*aa*cc)))/
**     &                                      (2.0*aa)
**                                 rootflowB=(-bb-sqrt((bb**2.0)-(4.0*aa*cc)))/
**     &                                      (2.0*aa)
C
**                                 if(rootflowA.ge.0)then
**                                   qpbranchbn(i,j-1,2)=-rootflowA
**                                 end if
C
**                                 if((rootflowA.lt.0).and.(rootflowB.ge.0))then
**                                   qpbranchbn(i,j-1,2)=-rootflowB
**                                 end if
C
                                    qpbranchbn(i,j-1,2)=0.0d0 ! ie. no backflow through water meter
C
                                 qpbranchbn(i,j,1)=qpbranchbn(i,j-1,2)
C
                                 hpbranchbn(i,j-1,2)=cpbranchbn(i,j)-
     &                                                qpbranchbn(i,j-1,2)*
     &                                                bbranchbn(i,j-1)
C
                                 hpbranchbn(i,j,1)=cmbranchbn(i,j)+
     &                                            qpbranchbn(i,j,1)*
     &                                            bbranchbn(i,j)
C
                               end if
C
C End of negative flow through inline orifice calculation
C
C End of in-line branch base orifice calculation
C
C ELSE calculate normal branch sub-segments
C
                          else
C
                        cpbranchbn(i,j)=hback1branchbn(i,j-1,1)+
     &                                          qback1branchbn(i,j-1,1)*
     &                                    (bbranchbn(i,j-1)-rbranchbn(i,j-1)*
     &                                    abs(qback1branchbn(i,j-1,1)))-
     &                                    Par2branchbn(i,j-1)*
     &                                        ysumbranchbn(i,j-1,1)-
     &                                        ((2.0d0*(abranchbn(i,j-1)**2.0d0)*
     &                                          dt)/g)*dErdtbn(i,j-1,1)*
     &                                          ParVEbn(i,j-1)
C
```



```
forward BSOLVER.txt - Notepad
File  Edit  Format  View  Help
C                              cmbranchbh(i,j)=hback1branchbh(i,j,2)-
     &                                                qback1branchbh(i,j,2)*
     &                                         (bbranchbh(i,j)-rbranchbh(i,j)*
     &                                         abs(qback1branchbh(i,j,2)))+
     &                                         Parzbranchbh(i,j)*
     &                                              Ysumbranchbh(i,j,2)-
     &                                          ((2.0d0*(abranchbh(i,j)**2.0d0)*dt)/
     &                                              g)*dErdtbh(i,j,2)*ParVEbh(i,j)
C
                               hpbranchbh(i,j-1,2)=(cpbranchbh(i,j)/
     &                                                         bbranchbh(i,j-1)+
     &                                                         cmbranchbh(i,j)/
     &                                                         bbranchbh(i,j))/(1.0/
     &                                                         bbranchbh(i,j-1)+1.0/
     &                                                   bbranchbh(i,j))
C
                               hpbranchbh(i,j,1)=hpbranchbh(i,j-1,2)
C
                               qpbranchbh(i,j-1,2)=(cpbranchbh(i,j)-
     &                                                         cmbranchbh(i,j))/
     &                                                         (bbranchbh(i,j-1)+
     &                                                         bbranchbh(i,j))
C
                               qpbranchbh(i,j,1)=qpbranchbh(i,j-1,2)
C
C End of if check for branch base orifice
C
                                 end if
C
                             end do ! end of loop for normal reaches / nodes along branch
C
C End of check for more than one segment
C
                           end if
C
C End of check for horizontal branch section
C
                         end if
C
C*****************************************************************72
C
C End of check for no lumping
C
                  end if
C
C-----------------------------------------------------------------72
C
C Junction with side discharge orifice - no standpipe
C
                  if(sdorfflag(i).eq.1)then
```

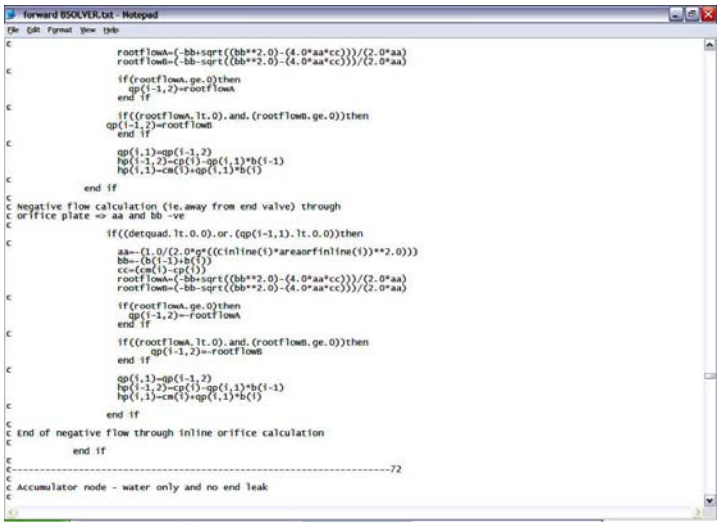


```
forward BSOLVER.txt - Notepad
File  Edit  Format  View  Help
                  if(sdorfflag(i).eq.1)then
C
                        cp(i)=hdash(i-1,1)+qdash(i-1,1)*(b(i-1)-r(i-1)*
     &                  abs(qdash(i-1,1)))-
     &                  Parz(i-1)*ysum(i-1,1)-
     &                        ((2.0d0*(a(i-1)**2.0d0)*dt)/g)*dErdt(i-1,1) ! visco
                        cm(i)=hdash(i,2)-qdash(i,2)*(b(i)-r(i)*
     &                  abs(qdash(i,2)))+
     &                  Parz(i)*ysum(i,2)-
     &                        ((2.0d0*(a(i)**2.0d0)*dt)/g)*dErdt(i,2) ! visco
C
C Assume positive head (ie.flow out of orifice) through side discharge
C orifice => aa and bb -ve
C
                        aa=-(b(i-1)+b(i))
                        bb=-cvsdorf(i)*(sqrt(2.0*g))*(b(i-1)*b(i))
                        cc=(b(i)*cp(i)+b(i-1)*cm(i))
     &                 -(headcorrection(i)*(b(i)+b(i-1)))
C
C Do determinant check to confirm whether head is positive or negative
C
                    detquad=((bb**2.0)-(4.0*aa*cc))
C
                    if(detquad.ge.0.0)then
C
                         rootheadA=(-bb+sqrt((bb**2.0)-(4.0*aa*cc)))/(2.0*aa)
                         rootheadB=(-bb-sqrt((bb**2.0)-(4.0*aa*cc)))/(2.0*aa)
C
                         if(rootheadA.ge.0)then
                           hp(i-1,2)=(rootheadA**2.0)+headcorrection(i)
                         end if
C
                         if((rootheadA.lt.0).and.(rootheadB.ge.0))then
                       hp(i-1,2)=(rootheadB**2.0)+headcorrection(i)
                         end if
C
                         hp(i,1)=hp(i-1,2)
                         qp(i-1,2)=(cp(i)-hp(i,1))/b(i-1)
                      qp(i,1)=(hp(i,1)-cm(i))/b(i)
C
                    end if
C
C Negative head calculation (ie.leak induction occurring) through
C side discharge (intake) orifice => aa and bb +ve
C
                        if(detquad.lt.0.0)then
C
                          aa=(b(i-1)+b(i))
                          bb=cvsdorf(i)*(sqrt(2.0*g))*(b(i-1)*b(i))
                          cc=(b(i)*cp(i)+b(i-1)*cm(i))
     &                 -(headcorrection(i)*(b(i)+b(i-1)))
```

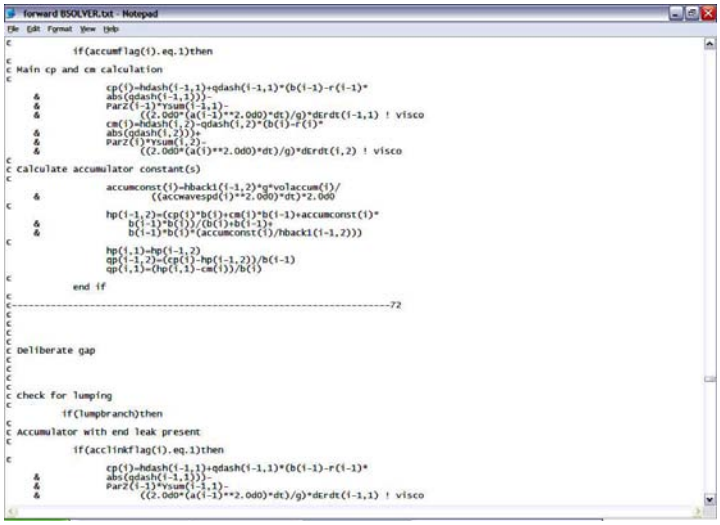


```
forward BSOLVER.txt - Notepad
File  Edit  Format  View  Help
     &                 -(headcorrection(i)*(b(i)+b(i-1)))
C
                       rootheadA=(-bb+sqrt((bb**2.0)-(4.0*aa*cc)))/(2.0*aa)
                       rootheadB=(-bb-sqrt((bb**2.0)-(4.0*aa*cc)))/(2.0*aa)
C
                       if(rootheadA.ge.0)then
                         hp(i-1,2)=-(rootheadA**2.0)+headcorrection(i)
                       end if
C
                       if((rootheadA.lt.0).and.(rootheadB.ge.0))then
                             hp(i-1,2)=-(rootheadB**2.0)+headcorrection(i)
                       end if
C
                       hp(i,1)=hp(i-1,2)
                       qp(i-1,2)=(cp(i)-hp(i,1))/b(i-1)
                       qp(i,1)=(hp(i,1)-cm(i))/b(i)
C
                     end if
C
C End of negative head at side discharge orifice calculation
C
                   end if
C
C-----------------------------------------------------------------72
C
C Junction with in-line orifice
C
                  if(ilorfflag(i).eq.1)then
C
                        cp(i)=hdash(i-1,1)+qdash(i-1,1)*(b(i-1)-r(i-1)*
     &                  abs(qdash(i-1,1)))-
     &                  Parz(i-1)*ysum(i-1,1)-
     &                        ((2.0d0*(a(i-1)**2.0d0)*dt)/g)*dErdt(i-1,1) ! visco
     &                  cm(i)=hdash(i,2)-qdash(i,2)*(b(i)-r(i)*
     &                  abs(qdash(i,2)))+
     &                  Parz(i)*ysum(i,2)-
     &                        ((2.0d0*(a(i)**2.0d0)*dt)/g)*dErdt(i,2) ! visco
C
C Assume positive flow (ie.towards end valve) through orifice plate
C => aa and bb +ve
C
                        aa=(1.0/(2.0*g*((cinline(i)*areaorfinline(i))**2.0)))
                        bb=(b(i-1)+b(i))
                        cc=(cm(i)-cp(i))
C
C Do determinant check to confirm whether flow is positive or negative
C
                    detquad=((bb**2.0)-(4.0*aa*cc))
C
                    if(detquad.ge.0.0)then
C
```

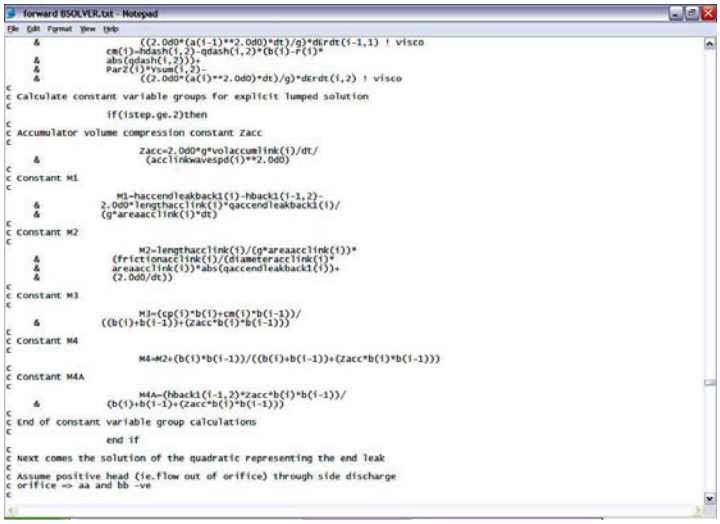
Appendix M – Fortran code for BSOLVER and NLFIT subroutines



```
forward BSOLVER.txt - Notepad
File  Edit  Format  View  Help
C
                    rootflowA=(-bb+sqrt((bb**2.0)-(4.0*aa*cc)))/(2.0*aa)
                    rootflowB=(-bb-sqrt((bb**2.0)-(4.0*aa*cc)))/(2.0*aa)
C
                    if(rootflowA.ge.0)then
                      qp(i-1,2)=rootflowA
                    end if
C
                    if((rootflowA.lt.0).and.(rootflowB.ge.0))then
                  qp(i-1,2)=rootflowB
                    end if
C
                    qp(i,1)=qp(i-1,2)
                    hp(i-1,2)=cp(i)-qp(i,1)*b(i-1)
                    hp(i,1)=cm(i)+qp(i,1)*b(i)
C
               end if
C
C Negative flow calculation (ie.away from end valve) through
C orifice plate => aa and bb -ve
C
                if((detquad.lt.0.0).or.(qp(i-1,1).lt.0.0))then
C
                  aa=-(1.0/(2.0*g*((cinline(i)*areaorfinline(i))**2.0)))
                  bb=-(b(i-1)+b(i))
                  cc=(cm(i)-cp(i))
                  rootflowA=(-bb+sqrt((bb**2.0)-(4.0*aa*cc)))/(2.0*aa)
                  rootflowB=(-bb-sqrt((bb**2.0)-(4.0*aa*cc)))/(2.0*aa)
C
                  if(rootflowA.ge.0)then
                    qp(i-1,2)=-rootflowA
                  end if
C
                  if((rootflowA.lt.0).and.(rootflowB.ge.0))then
                        qp(i-1,2)=-rootflowB
                  end if
C
                  qp(i,1)=qp(i-1,2)
                  hp(i-1,2)=cp(i)-qp(i,1)*b(i-1)
                  hp(i,1)=cm(i)+qp(i,1)*b(i)
C
                end if
C
C End of negative flow through inline orifice calculation
C
           end if
C
C-----------------------------------------------------------------------72
C
C Accumulator node - water only and no end leak
C
```

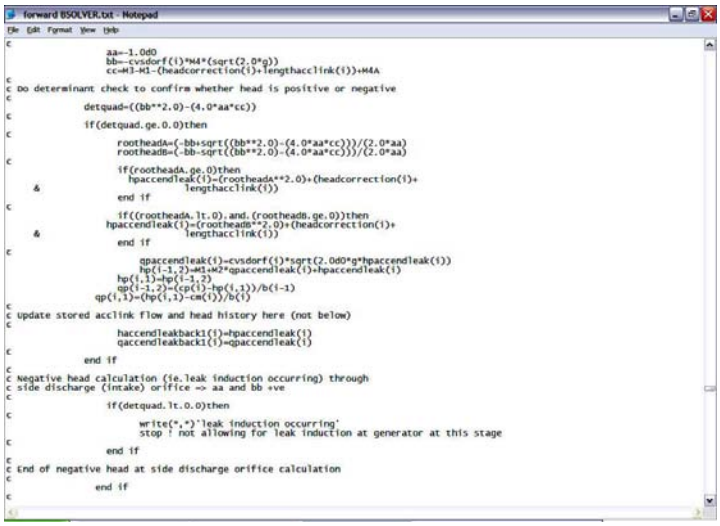


```
forward BSOLVER.txt - Notepad
File  Edit  Format  View  Help
C
           if(accumflag(i).eq.1)then
C
C Main cp and cm calculation
C
                cp(i)=hdash(i-1,1)+qdash(i-1,1)*(b(i-1)-r(i-1)*
     &          abs(qdash(i-1,1)))-
     &          Parz(i-1)*vsum(i-1,1)-
     &                ((2.0d0*(a(i-1)**2.0d0)*dt)/g)*dErdt(i-1,1) ! visco
                cm(i)=hdash(i,2)-qdash(i,2)*(b(i)-r(i)*
     &          abs(qdash(i,2)))+
     &          Parz(i)*vsum(i,2)-
     &                ((2.0d0*(a(i)**2.0d0)*dt)/g)*dErdt(i,2) ! visco
C
C Calculate accumulator constant(s)
C
                accumconst(i)=hback1(i-1,2)*g*volaccum(i)/
     &                ((accwavespd(i)**2.0d0)*dt)*2.0d0
C
                hp(i-1,2)=(cp(i)*b(i)+cm(i)*b(i-1)+accumconst(i)*
     &              b(i-1)*b(i))/(b(i)+b(i-1))+
     &              b(i-1)*b(i)*(accumconst(i)/hback1(i-1,2)))
C
                hp(i,1)=hp(i-1,2)
                qp(i-1,2)=(cp(i)-hp(i-1,2))/b(i-1)
                qp(i,1)=(hp(i,1)-cm(i))/b(i)
C
           end if
C
C-----------------------------------------------------------------------72
C
C
C
C Deliberate gap
C
C
C
C
C Check for lumping
C
         if(lumpbranch)then
C
C Accumulator with end leak present
C
           if(acclinkflag(i).eq.1)then
C
                cp(i)=hdash(i-1,1)+qdash(i-1,1)*(b(i-1)-r(i-1)*
     &          abs(qdash(i-1,1)))-
     &          Par2(i-1)*vsum(i-1,1)-
     &                ((2.0d0*(a(i-1)**2.0d0)*dt)/g)*dErdt(i-1,1) ! visco
```

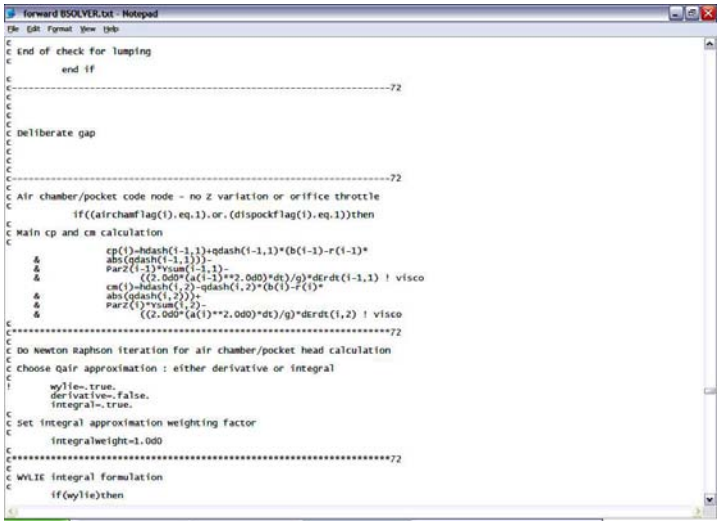


```
forward BSOLVER.txt - Notepad
File  Edit  Format  View  Help
     &                ((2.0d0*(a(i-1)**2.0d0)*dt)/g)*dErdt(i-1,1) ! visco
                cm(i)=hdash(i,2)-qdash(i,2)*(b(i)-r(i)*
     &          abs(qdash(i,2)))+
     &          Par2(i)*vsum(i,2)-
     &                ((2.0d0*(a(i)**2.0d0)*dt)/g)*dErdt(i,2) ! visco
C
C Calculate constant variable group for explicit lumped solution
C
                if(istep.ge.2)then
C
C Accumulator volume compression constant Zacc
C
                    Zacc=2.0d0*g*volaccumlink(i)/dt/
     &                (acclinkwavespd(i)**2.0d0)
C
C Constant M1
C
                  M1=haccendleakback1(i)-hback1(i-1,2)-
     &            2.0d0*lengthacclink(i)*qaccendleakback1(i)/
     &            (g*areaacclink(i)*dt)
C
C Constant M2
C
                    M2=lengthacclink(i)/(g*areaacclink(i))*
     &              (frictionacclink(i)/(diameteracclink(i)*
     &              areaacclink(i))*abs(qaccendleakback1(i))+
     &              (2.0d0/dt))
C
C Constant M3
C
                    M3=(cp(i)*b(i)+cm(i)*b(i-1))/
     &            ((b(i)+b(i-1))+(Zacc*b(i)*b(i-1)))
C
C Constant M4
C
                    M4=M2+(b(i)*b(i-1))/((b(i)+b(i-1))+(Zacc*b(i)*b(i-1)))
C
C Constant M4A
C
                    M4A=(hback1(i-1,2)*Zacc*b(i)*b(i-1))/
     &            (b(i)+b(i-1)+(Zacc*b(i)*b(i-1)))
C
C End of constant variable group calculations
C
                end if
C
C Next comes the solution of the quadratic representing the end leak
C
C Assume positive head (ie.flow out of orifice) through side discharge
C orifice => aa and bb -ve
C
```

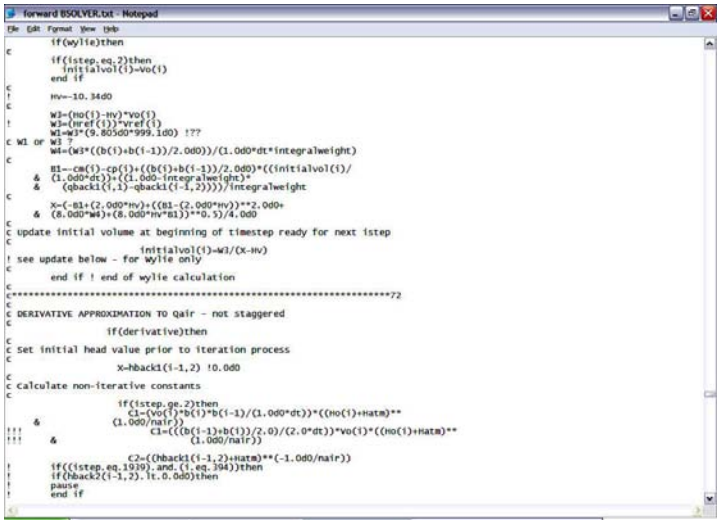
Appendix M – Fortran code for BSOLVER and NLFIT subroutines




```
c
                  aa=-1.0d0
                  bb=-cvsdorf(i)*M4*(sqrt(2.0*g))
                  cc=M3-M1-(headcorrection(i)+lengthacclink(i))+M4A
c
c Do determinant check to confirm whether head is positive or negative
c
            detquad=((bb**2.0)-(4.0*aa*cc))
c
            if(detquad.ge.0.0)then
c
                  rootheadA=(-bb+sqrt((bb**2.0)-(4.0*aa*cc)))/(2.0*aa)
                  rootheadB=(-bb-sqrt((bb**2.0)-(4.0*aa*cc)))/(2.0*aa)
c
                  if(rootheadA.ge.0)then
                    hpaccendleak(i)=(rootheadA**2.0)+(headcorrection(i)+
     &                          lengthacclink(i))
                  end if
c
                  if((rootheadA.lt.0).and.(rootheadB.ge.0))then
                 hpaccendleak(i)=(rootheadB**2.0)+(headcorrection(i)+
     &                          lengthacclink(i))
                  end if
c
                      qpaccendleak(i)=cvsdorf(i)*sqrt(2.0d0*g*hpaccendleak(i))
                      hp(i-1,2)=M1+M2*qpaccendleak(i)+hpaccendleak(i)
                  hp(i,1)=hp(i-1,2)
                  qp(i-1,2)=(cp(i)-hp(i,1))/b(i-1)
               qp(i,1)=(hp(i,1)-cm(i))/b(i)
c
c Update stored acclink flow and head history here (not below)
c
                  haccendleakback1(i)=hpaccendleak(i)
                  qaccendleakback1(i)=qpaccendleak(i)
c
            end if
c
c Negative head calculation (ie.leak induction occurring) through
c side discharge (intake) orifice => aa and bb +ve
c
               if(detquad.lt.0.0)then
c
                     write(*,*)'leak induction occurring'
                     stop ! not allowing for leak induction at generator at this stage
c
               end if
c
c End of negative head at side discharge orifice calculation
c
            end if
c
```

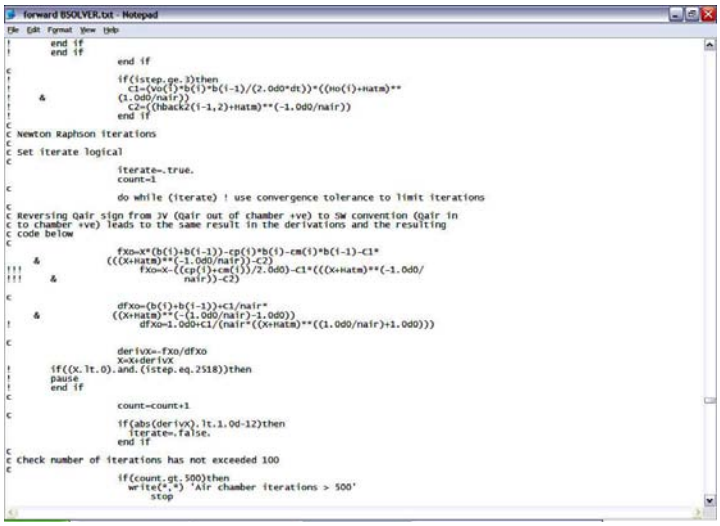



```
c
c End of check for lumping
c
         end if
c
c-------------------------------------------------------------------------72
c
c
c
c
c Deliberate gap
c
c
c
c
c-------------------------------------------------------------------------72
c Air chamber/pocket code node - no Z variation or orifice throttle
c
          if((airchamflag(i).eq.1).or.(dispockflag(i).eq.1))then
c
c Main cp and cm calculation
c
                  cp(i)=hdash(i-1,1)+qdash(i-1,1)*(b(i-1)-r(i-1)*
     &            abs(qdash(i-1,1)))-
     &            Par2(i-1)*vsum(i-1,1)-
     &                  ((2.0d0*(a(i-1)**2.0d0)*dt)/g)*dErdt(i-1,1) ! visco
                  cm(i)=hdash(i,2)-qdash(i,2)*(b(i)-r(i)*
     &            abs(qdash(i,2)))+
     &            Par2(i)*vsum(i,2)-
     &                  ((2.0d0*(a(i)**2.0d0)*dt)/g)*dErdt(i,2) ! visco
c
c**************************************************************************72
c
c Do Newton Raphson iteration for air chamber/pocket head calculation
c
c Choose Qair approximation : either derivative or integral
c
!       wylie=.true.
        derivative=.false.
        integral=.true.
c
c Set integral approximation weighting factor
c
        integralweight=1.0d0
c
c**************************************************************************72
c
c WYLIE integral formulation
c
        if(wylie)then
```

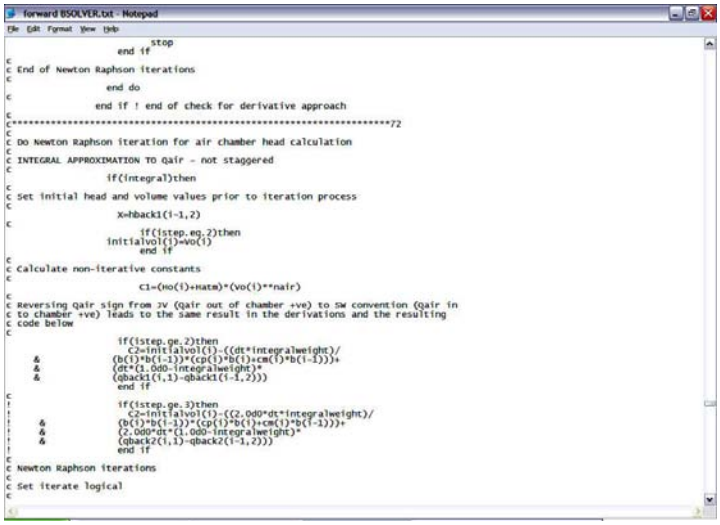



```
        if(wylie)then
c
        if(istep.eq.2)then
          initialvol(i)=Vo(i)
        end if
c
!       Hv=-10.34d0
c
        W3=(Ho(i)-Hv)*Vo(i)
!       W3=(Href(i))*Vref(i)
        W1=W3*(9.805d0*999.1d0) !??
c W1 or W3 ?
        W4=(W3*((b(i)+b(i-1))/2.0d0))/(1.0d0*dt*integralweight)
c
        B1=-cm(i)-cp(i)+((b(i)+b(i-1))/2.0d0)*((initialvol(i)/
     &  (1.0d0*dt))+((1.0d0-integralweight)*
     &    (qback1(i,1)-qback1(i-1,2))))/integralweight
c
        X=(-B1+(2.0d0*Hv)+((B1-(2.0d0*Hv))**2.0d0+
     &  (8.0d0*W4)+(8.0d0*Hv*B1))**0.5)/4.0d0
c
c Update initial volume at beginning of timestep ready for next istep
c
                  initialvol(i)=W3/(X-Hv)
! see update below - for wylie only
c
        end if ! end of wylie calculation
c
c**************************************************************************72
c
c DERIVATIVE APPROXIMATION TO Qair - not staggered
c
                  if(derivative)then
c
c Set initial head value prior to iteration process
c
                  X=hback1(i-1,2) !0.0d0
c
c Calculate non-iterative constants
c
                  if(istep.ge.2)then
                    C1=(Vo(i)*b(i)*b(i-1)/(1.0d0*dt))*((Ho(i)+Hatm)**
     &            (1.0d0/nair))
!!!                 C1=(((b(i-1)+b(1))/2.0)/(2.0*dt))*Vo(i)*((Ho(i)+Hatm)**
!!!   &                       (1.0d0/nair))
                    C2=((hback1(i-1,2)+Hatm)**(-1.0d0/nair))
!       if((istep.eq.1939).and.(i.eq.394))then
!       if(hback2(i-1,2).lt.0.0d0)then
!       pause
        end if
```

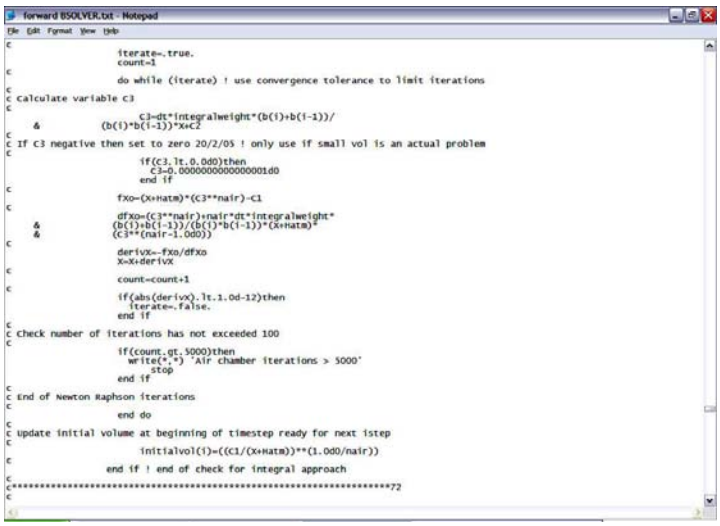
Appendix M – Fortran code for BSOLVER and NLFIT subroutines




```
!       end if
!       end if
                    end if
c
!                   if(istep.ge.3)then
!                     c1=(vo(i)*b(i)*b(i-1)/(2.0d0*dt))*((Ho(i)+Hatm)**
!     &               (1.0d0/nair))
!                     c2=((hback2(i-1,2)+Hatm)**(-1.0d0/nair))
!                   end if
c
c Newton Raphson iterations
c
c Set iterate logical
c
                    iterate=.true.
                    count=1
c
                    do while (iterate) ! use convergence tolerance to limit iterations
c
c Reversing Qair sign from JV (Qair out of chamber +ve) to SW convention (Qair in
c to chamber +ve) leads to the same result in the derivations and the resulting
c code below
c
                    fXo=X*(b(i)+b(i-1))-cp(i)*b(i)-cm(i)*b(i-1)-C1*
     &             (((X+Hatm)**(-1.0d0/nair))-C2)
!!!                   fXo=X-((cp(i)+cm(i))/2.0d0)-C1*(((X+Hatm)**(-1.0d0/
!!!    &                      nair))-C2)
c
                    dfXo=(b(i)+b(i-1))+C1/nair*
     &             ((X+Hatm)**(-(1.0d0/nair)-1.0d0))
!                       dfXo=1.0d0+C1/(nair*((X+Hatm)**((1.0d0/nair)+1.0d0)))
c
                    derivX=-fXo/dfXo
                    X=X+derivX
!       if((X.lt.0).and.(istep.eq.2518))then
!       pause
!       end if
c
                    count=count+1
c
                    if(abs(derivX).lt.1.0d-12)then
                      iterate=.false.
                    end if
c
c Check number of iterations has not exceeded 100
c
                    if(count.gt.500)then
                      write(*,*) 'Air chamber iterations > 500'
                          stop
```

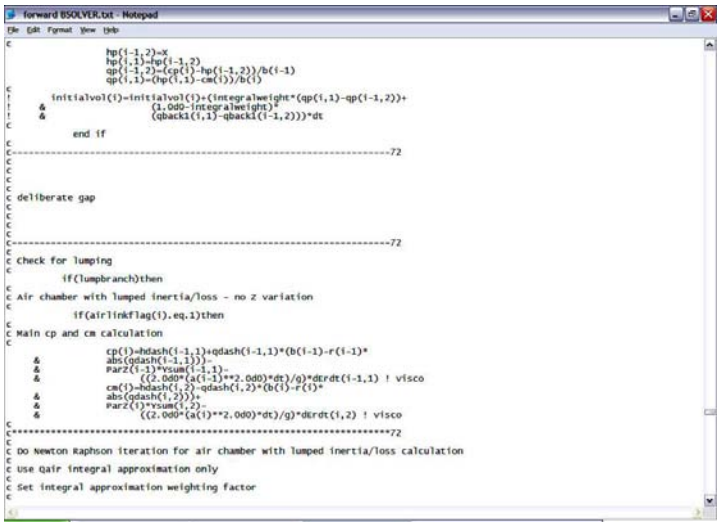



```
                          stop
                    end if
c
c End of Newton Raphson iterations
c
                  end do
c
                end if ! end of check for derivative approach
c
c**********************************************************************72
c
c Do Newton Raphson iteration for air chamber head calculation
c
c INTEGRAL APPROXIMATION TO Qair - not staggered
c
                if(integral)then
c
c Set initial head and volume values prior to iteration process
c
                    X=hback1(i-1,2)
c
                    if(istep.eq.2)then
                  initialvol(i)=vo(i)
                    end if
c
c Calculate non-iterative constants
c
                    C1=(Ho(i)+Hatm)*(vo(i)**nair)
c
c Reversing Qair sign from JV (Qair out of chamber +ve) to SW convention (Qair in
c to chamber +ve) leads to the same result in the derivations and the resulting
c code below
c
                    if(istep.ge.2)then
                      C2=initialvol(i)-((dt*integralweight)/
     &               (b(i)*b(i-1))*(cp(i)*b(i)+cm(i)*b(i-1)))+
     &               (dt*(1.0d0-integralweight)*
     &               (qback1(i,1)-qback1(i-1,2)))
                    end if
c
!                   if(istep.ge.3)then
!                     C2=initialvol(i)-((2.0d0*dt*integralweight)/
!     &               (b(i)*b(i-1))*(cp(i)*b(i)+cm(i)*b(i-1)))+
!     &               (2.0d0*dt*(1.0d0-integralweight)*
!     &               (qback2(i,1)-qback2(i-1,2)))
!                   end if
c
c Newton Raphson iterations
c
c Set iterate logical
c
```

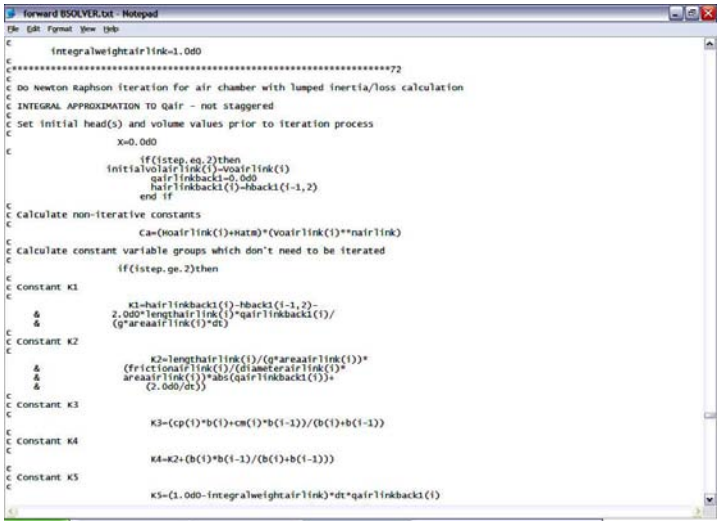



```
c
                    iterate=.true.
                    count=1
c
                    do while (iterate) ! use convergence tolerance to limit iterations
c
c Calculate variable C3
c
                          C3=dt*integralweight*(b(i)+b(i-1))/
     &               (b(i)*b(i-1))*X+C2
c
c If C3 negative then set to zero 20/2/05 ! only use if small vol is an actual problem
c
                        if(C3.lt.0.0d0)then
                          C3=0.0000000000000001d0
                        end if
c
                    fXo=(X+Hatm)*(C3**nair)-C1
c
                    dfXo=(C3**nair)+nair*dt*integralweight*
     &             (b(i)+b(i-1))/(b(i)*b(i-1))*(X+Hatm)*
     &             (C3**(nair-1.0d0))
c
                    derivX=-fXo/dfXo
                    X=X+derivX
c
                    count=count+1
c
                    if(abs(derivX).lt.1.0d-12)then
                      iterate=.false.
                    end if
c
c Check number of iterations has not exceeded 100
c
                    if(count.gt.5000)then
                      write(*,*) 'Air chamber iterations > 5000'
                          stop
                    end if
c
c End of Newton Raphson iterations
c
                    end do
c
c Update initial volume at beginning of timestep ready for next istep
c
                        initialvol(i)=((C1/(X+Hatm))**(1.0d0/nair))
c
                  end if ! end of check for integral approach
c
c**********************************************************************72
c
```

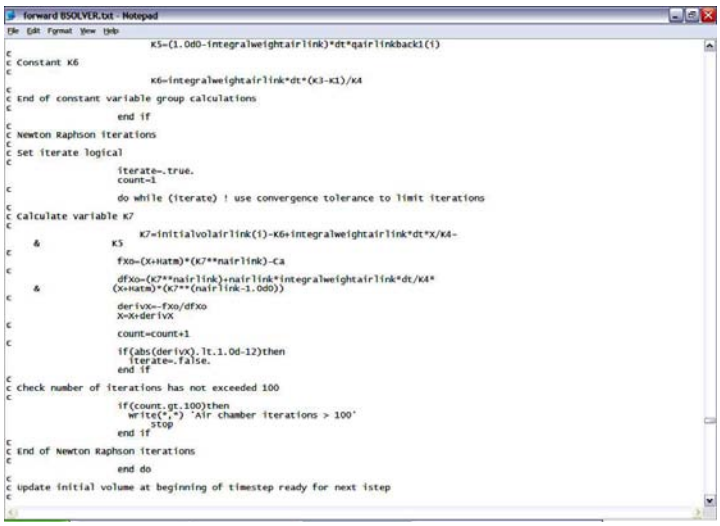
Appendix M – Fortran code for BSOLVER and NLFIT subroutines



```
forward BSOLVER.txt - Notepad
File  Edit  Format  View  Help
C                   hp(i-1,2)=X
                    hp(i,1)=hp(i-1,2)
                    qp(i-1,2)=(cp(i)-hp(i-1,2))/b(i-1)
                    qp(i,1)=(hp(i,1)-cm(i))/b(i)
C
!       initialvol(i)=initialvol(i)+(integralweight*(qp(i,1)-qp(i-1,2))+
!     &                    (1.0d0-integralweight)*
!     &                    (qback1(i,1)-qback1(i-1,2)))*dt
C
            end if
C
C---------------------------------------------------------------------72
C
C
C
C
C deliberate gap
C
C
C
C
C---------------------------------------------------------------------72
C
C Check for lumping
C
          if(lumpbranch)then
C
C Air chamber with lumped inertia/loss - no z variation
C
            if(airlinkflag(i).eq.1)then
C
C Main cp and cm calculation
C
                    cp(i)=hdash(i-1,1)+qdash(i-1,1)*(b(i-1)-r(i-1)*
     &              abs(qdash(i-1,1)))-
     &              Parz(i-1)*ysum(i-1,1)-
     &                    ((2.0d0*(a(i-1)**2.0d0)*dt)/g)*dtrdt(i-1,1) ! visco
                    cm(i)=hdash(i,2)-qdash(i,2)*(b(i)-r(i)*
     &              abs(qdash(i,2)))+
     &              Parz(i)*ysum(i,2)-
     &                    ((2.0d0*(a(i)**2.0d0)*dt)/g)*dtrdt(i,2) ! visco
C
C*********************************************************************72
C
C Do Newton Raphson iteration for air chamber with lumped inertia/loss calculation
C
C Use Qair integral approximation only
C
C Set integral approximation weighting factor
C
```

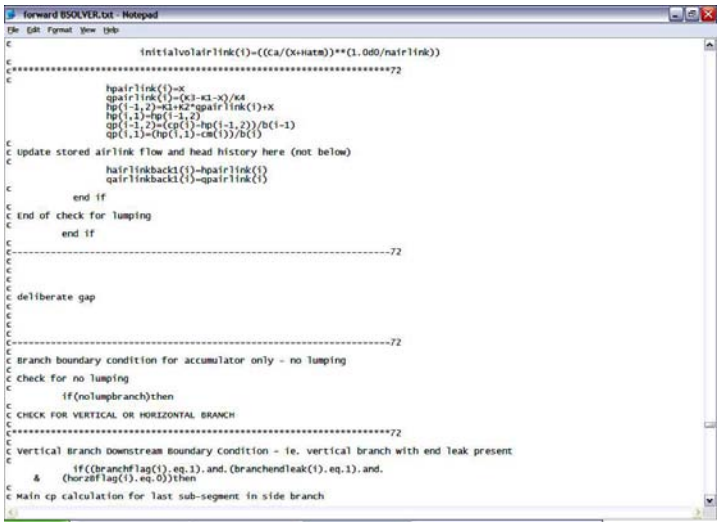


```
forward BSOLVER.txt - Notepad
File  Edit  Format  View  Help
C
            integralweightairlink=1.0d0
C
C*********************************************************************72
C
C Do Newton Raphson iteration for air chamber with lumped inertia/loss calculation
C
C INTEGRAL APPROXIMATION TO Qair - not staggered
C
C Set initial head(s) and volume values prior to iteration process
C
                    X=0.0d0
C
                      if(istep.eq.2)then
                  initialvolairlink(i)=voairlink(i)
                         qairlinkback1=0.0d0
                         hairlinkback1(i)=hback1(i-1,2)
                      end if
C
C Calculate non-iterative constants
C
                      Ca=(Hoairlink(i)+Hatm)*(voairlink(i)**nairlink)
C
C Calculate constant variable groups which don't need to be iterated
C
                     if(istep.ge.2)then
C
C Constant K1
C
                       K1=hairlinkback1(i)-hback1(i-1,2)-
     &                2.0d0*lengthairlink(i)*qairlinkback1(i)/
     &                (g*areaairlink(i)*dt)
C
C Constant K2
C
                       K2=lengthairlink(i)/(g*areaairlink(i))*
     &                (frictionairlink(i)/(diameterairlink(i)*
     &                areaairlink(i))*abs(qairlinkback1(i))+
     &                    (2.0d0/dt))
C
C Constant K3
C
                       K3=(cp(i)*b(i)+cm(i)*b(i-1))/(b(i)+b(i-1))
C
C Constant K4
C
                       K4=K2+(b(i)*b(i-1)/(b(i)+b(i-1)))
C
C Constant K5
C
                       K5=(1.0d0-integralweightairlink)*dt*qairlinkback1(i)
```

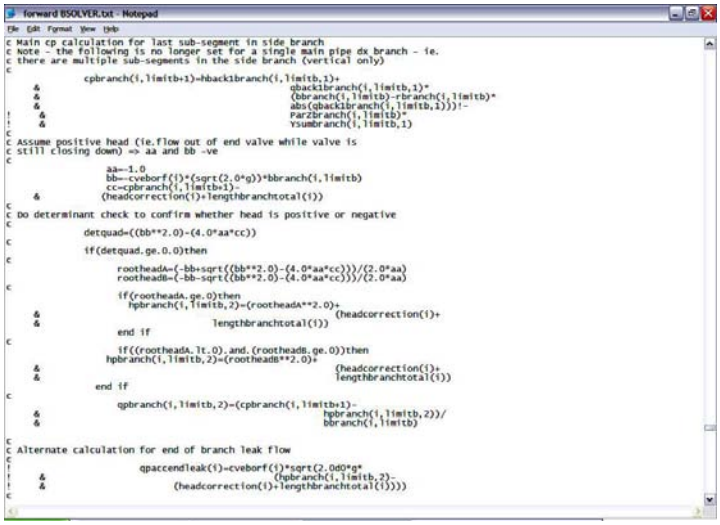


```
forward BSOLVER.txt - Notepad
File  Edit  Format  View  Help
                       K5=(1.0d0-integralweightairlink)*dt*qairlinkback1(i)
C
C Constant K6
C
                       K6=integralweightairlink*dt*(K3-K1)/K4
C
C End of constant variable group calculations
C
                     end if
C
C Newton Raphson iterations
C
C Set iterate logical
C
                     iterate=.true.
                     count=1
C
                     do while (iterate) ! use convergence tolerance to limit iterations
C
C Calculate variable K7
C
                       K7=initialvolairlink(i)-K6+integralweightairlink*dt*X/K4-
     &                K5
C
                     fXo=(X+Hatm)*(K7**nairlink)-Ca
C
                     dfXo=(K7**nairlink)+nairlink*integralweightairlink*dt/K4*
     &                (X+Hatm)*(K7**(nairlink-1.0d0))
C
                     derivX=-fXo/dfXo
                     X=X+derivX
C
                     count=count+1
C
                     if(abs(derivX).lt.1.0d-12)then
                       iterate=.false.
                     end if
C
C Check number of iterations has not exceeded 100
C
                     if(count.gt.100)then
                       write(*,*) 'Air chamber iterations > 100'
                           stop
                     end if
C
C End of Newton Raphson iterations
C
                     end do
C
C Update initial volume at beginning of timestep ready for next istep
C
```

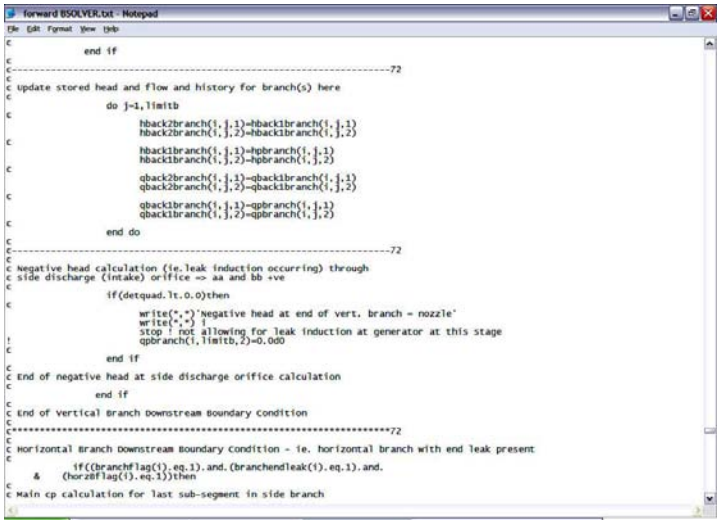
Appendix M – Fortran code for BSOLVER and NLFIT subroutines



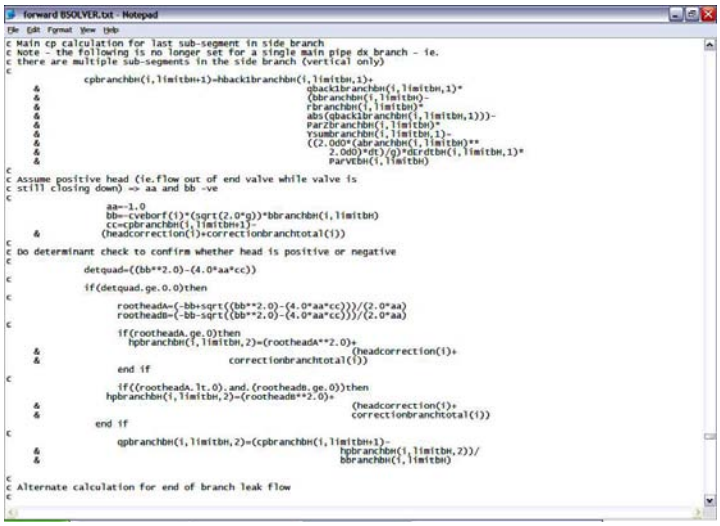

```
c                       initialvolairlink(i)=((Ca/(X+Hatm))**(1.0d0/nairlink))
c
c*************************************************************************72
c
                  hpairlink(i)=x
                  qpairlink(i)=(x1-x1-x)/x4
                  hp(i-1,2)=K1+K2*qpairlink(i)+x
                  hp(i,1)=hp(i-1,2)
                  qp(i-1,2)=(cp(i)-hp(i-1,2))/b(i-1)
                  qp(i,1)=(hp(i,1)-cm(i))/b(i)
c
c Update stored airlink flow and head history here (not below)
c
                  hairlinkback1(i)=hpairlink(i)
                  qairlinkback1(i)=qpairlink(i)
c
            end if
c
c End of check for lumping
c
         end if
c
c-------------------------------------------------------------------------72
c
c
c
c deliberate gap
c
c
c
c-------------------------------------------------------------------------72
c
c Branch boundary condition for accumulator only - no lumping
c
c Check for no lumping
c
         if(nolumpbranch)then
c
c CHECK FOR VERTICAL OR HORIZONTAL BRANCH
c
c*************************************************************************72
c
c Vertical Branch Downstream Boundary Condition - ie. vertical branch with end leak present
c
           if((branchflag(i).eq.1).and.(branchendleak(i).eq.1).and.
     &    (horzbflag(i).eq.0))then
c
c Main cp calculation for last sub-segment in side branch
```



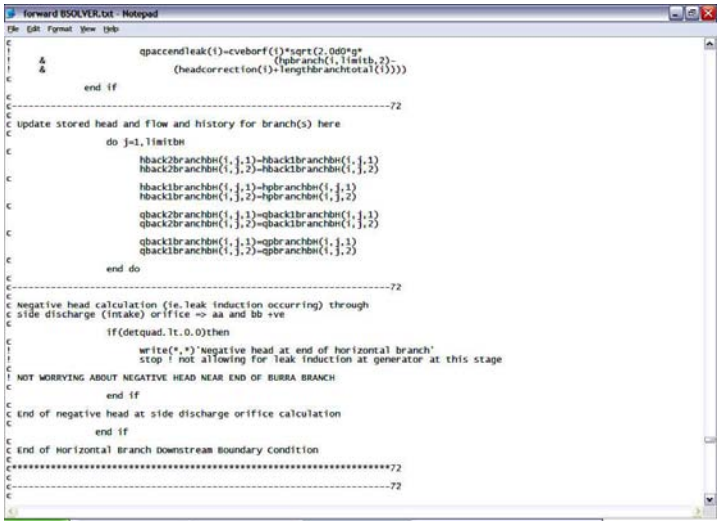

```
c Main cp calculation for last sub-segment in side branch
c Note - the following is no longer set for a single main pipe dx branch - ie.
c there are multiple sub-segments in the side branch (vertical only)
c
              cpbranch(i,limitb+1)=hback1branch(i,limitb,1)+
     &                                          qback1branch(i,limitb,1)*
     &                                          (bbranch(i,limitb)-rbranch(i,limitb)*
     &                                          abs(qback1branch(i,limitb,1)))!-
!    &                                          Par2branch(i,limitb)*
!    &                                          ysumbranch(i,limitb,1)
c
c Assume positive head (ie.flow out of end valve while valve is
c still closing down) => aa and bb -ve
c
                  aa=-1.0
                  bb=-cveborf(i)*(sqrt(2.0*g))*bbranch(i,limitb)
                  cc=cpbranch(i,limitb+1)-
     &           (headcorrection(i)+lengthbranchtotal(i))
c
c Do determinant check to confirm whether head is positive or negative
c
              detquad=((bb**2.0)-(4.0*aa*cc))
c
              if(detquad.ge.0.0)then
c
                   rootheadA=(-bb+sqrt((bb**2.0)-(4.0*aa*cc)))/(2.0*aa)
                   rootheadB=(-bb-sqrt((bb**2.0)-(4.0*aa*cc)))/(2.0*aa)
c
                   if(rootheadA.ge.0)then
                     hpbranch(i,limitb,2)=(rootheadA**2.0)+
     &                                          (headcorrection(i)+
     &                              lengthbranchtotal(i))
                   end if
c
                   if((rootheadA.lt.0).and.(rootheadB.ge.0))then
                  hpbranch(i,limitb,2)=(rootheadB**2.0)+
     &                                          (headcorrection(i)+
     &                                          lengthbranchtotal(i))
                 end if
c
                   qpbranch(i,limitb,2)=(cpbranch(i,limitb+1)-
     &                                          hpbranch(i,limitb,2))/
     &                                          bbranch(i,limitb)

c
c Alternate calculation for end of branch leak flow
c
!                  qpaccendleak(i)=cveborf(i)*sqrt(2.0d0*g*
!    &                                          (hpbranch(i,limitb,2)-
!    &                    (headcorrection(i)+lengthbranchtotal(i))))
c
```



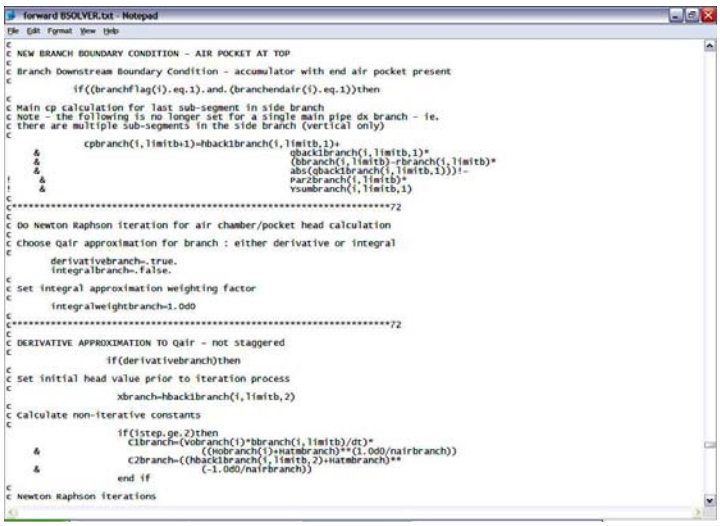

```
c
              end if
c
c-------------------------------------------------------------------------72
c
c Update stored head and flow and history for branch(s) here
c
                  do j=1,limitb
c
                       hback2branch(i,j,1)=hback1branch(i,j,1)
                       hback2branch(i,j,2)=hback1branch(i,j,2)
c
                       hback1branch(i,j,1)=hpbranch(i,j,1)
                       hback1branch(i,j,2)=hpbranch(i,j,2)
c
                       qback2branch(i,j,1)=qback1branch(i,j,1)
                       qback2branch(i,j,2)=qback1branch(i,j,2)
c
                       qback1branch(i,j,1)=qpbranch(i,j,1)
                       qback1branch(i,j,2)=qpbranch(i,j,2)
c
                  end do
c
c-------------------------------------------------------------------------72
c
c Negative head calculation (ie.leak induction occurring) through
c side discharge (intake) orifice => aa and bb +ve
c
                  if(detquad.lt.0.0)then
c
                       write(*,*)'Negative head at end of vert. branch - nozzle'
                       write(*,*) i
                       stop ! not allowing for leak induction at generator at this stage
!                      qpbranch(i,limitb,2)=0.0d0
c
                  end if
c
c End of negative head at side discharge orifice calculation
c
               end if
c
c End of Vertical Branch Downstream Boundary Condition
c
c*************************************************************************72
c
c Horizontal Branch Downstream Boundary Condition - ie. horizontal branch with end leak present
c
           if((branchflag(i).eq.1).and.(branchendleak(i).eq.1).and.
     &    (horzbflag(i).eq.1))then
c
c Main cp calculation for last sub-segment in side branch
```



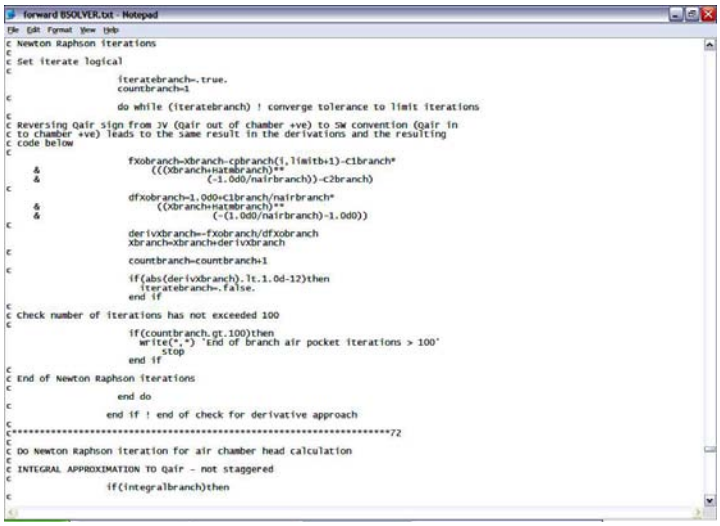
```
forward BSOLVER.txt - Notepad
File  Edit  Format  View  Help
c Main cp calculation for last sub-segment in side branch
c Note - the following is no longer set for a single main pipe dx branch - ie.
c there are multiple sub-segments in the side branch (vertical only)
c
                cpbranchbh(i,limitbh+1)=hback1branchbh(i,limitbh,1)+
     &                                          qback1branchbh(i,limitbh,1)*
     &                                          (bbranchbh(i,limitbh)-
     &                                          rbranchbh(i,limitbh)*
     &                                          abs(qback1branchbh(i,limitbh,1)))-
     &                                          Par2branchbh(i,limitbh)*
     &                                          Ysumbranchbh(i,limitbh,1)-
     &                                          ((2.0d0*(abranchbh(i,limitbh)**
     &                                              2.0d0)*dt)/g)*dEdtbh(i,limitbh,1)*
     &                                              ParVEbh(i,limitbh)
c
c Assume positive head (ie.flow out of end valve while valve is
c still closing down) => aa and bb -ve
c
                 aa=-1.0
                 bb=-cveborf(i)*(sqrt(2.0*g))*bbranchbh(i,limitbh)
                 cc=cpbranchbh(i,limitbh+1)-
     &           (headcorrection(i)+correctionbranchtotal(i))
c
c Do determinant check to confirm whether head is positive or negative
c
                 detquad=((bb**2.0)-(4.0*aa*cc))
c
                 if(detquad.ge.0.0)then
c
                      rootheadA=(-bb+sqrt((bb**2.0)-(4.0*aa*cc)))/(2.0*aa)
                      rootheadB=(-bb-sqrt((bb**2.0)-(4.0*aa*cc)))/(2.0*aa)
c
                      if(rootheadA.ge.0)then
                        hpbranchbh(i,limitbh,2)=(rootheadA**2.0)+
     &                                              (headcorrection(i)+
     &                                  correctionbranchtotal(i))
                      end if
c
                      if((rootheadA.lt.0).and.(rootheadB.ge.0))then
                    hpbranchbh(i,limitbh,2)=(rootheadB**2.0)+
     &                                              (headcorrection(i)+
     &                                              correctionbranchtotal(i))
                    end if
c
                      qpbranchbh(i,limitbh,2)=(cpbranchbh(i,limitbh+1)-
     &                                              hpbranchbh(i,limitbh,2))/
     &                                              bbranchbh(i,limitbh)
c
c Alternate calculation for end of branch leak flow
c
```



```
forward BSOLVER.txt - Notepad
File  Edit  Format  View  Help
c
!                qpaccendleak(i)=cveborf(i)*sqrt(2.0d0*g*
!     &                                          (hpbranch(i,limitb,2)-
!     &                                (headcorrection(i)+lengthbranchtotal(i))))
c
                 end if
c
c----------------------------------------------------------------------72
c
c Update stored head and flow and history for branch(s) here
c
                 do j=1,limitbh
c
                      hback2branchbh(i,j,1)=hback1branchbh(i,j,1)
                      hback2branchbh(i,j,2)=hback1branchbh(i,j,2)
c
                      hback1branchbh(i,j,1)=hpbranchbh(i,j,1)
                      hback1branchbh(i,j,2)=hpbranchbh(i,j,2)
c
                      qback2branchbh(i,j,1)=qback1branchbh(i,j,1)
                      qback2branchbh(i,j,2)=qback1branchbh(i,j,2)
c
                      qback1branchbh(i,j,1)=qpbranchbh(i,j,1)
                      qback1branchbh(i,j,2)=qpbranchbh(i,j,2)
c
                 end do
c
c----------------------------------------------------------------------72
c
c Negative head calculation (ie.leak induction occurring) through
c side discharge (intake) orifice => aa and bb +ve
c
                 if(detquad.lt.0.0)then
c
!                     write(*,*)'Negative head at end of horizontal branch'
!                     stop ! not allowing for leak induction at generator at this stage
c
! NOT WORRYING ABOUT NEGATIVE HEAD NEAR END OF BURRA BRANCH
c
                 end if
c
c End of negative head at side discharge orifice calculation
c
               end if
c
c End of Horizontal Branch Downstream Boundary Condition
c
c**********************************************************************72
c
c----------------------------------------------------------------------72
c
```



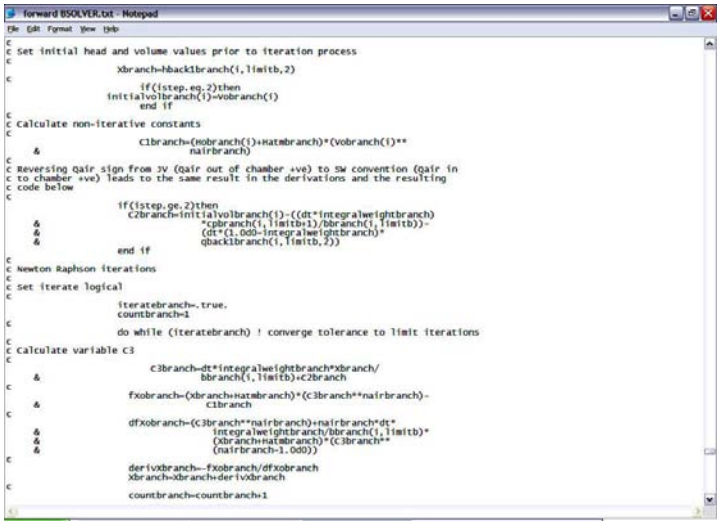
```
forward BSOLVER.txt - Notepad
File  Edit  Format  View  Help
c
c NEW BRANCH BOUNDARY CONDITION - AIR POCKET AT TOP
c
c Branch Downstream Boundary Condition - accumulator with end air pocket present
c
          if((branchflag(i).eq.1).and.(branchendair(i).eq.1))then
c
c Main cp calculation for last sub-segment in side branch
c Note - the following is no longer set for a single main pipe dx branch - ie.
c there are multiple sub-segments in the side branch (vertical only)
c
                cpbranch(i,limitb+1)=hback1branch(i,limitb,1)+
     &                                          qback1branch(i,limitb,1)*
     &                                          (bbranch(i,limitb)-rbranch(i,limitb)*
     &                                          abs(qback1branch(i,limitb,1)))-
!    &                                          Par2branch(i,limitb)*
!    &                                          Ysumbranch(i,limitb,1)
c
c**********************************************************************72
c
c Do Newton Raphson iteration for air chamber/pocket head calculation
c
c Choose Qair approximation for branch : either derivative or integral
c
        derivativebranch=.true.
        integralbranch=.false.
c
c Set integral approximation weighting factor
c
        integralweightbranch=1.0d0
c
c**********************************************************************72
c
c DERIVATIVE APPROXIMATION TO Qair - not staggered
c
                 if(derivativebranch)then
c
c Set initial head value prior to iteration process
c
                      xbranch=hback1branch(i,limitb,2)
c
c Calculate non-iterative constants
c
                      if(istep.ge.2)then
                        c1branch=(vobranch(i)*bbranch(i,limitb)/dt)*
     &                                  ((nobranch(i)+Hatmbranch)**(1.0d0/nairbranch))
                        c2branch=((hback1branch(i,limitb,2)+Hatmbranch)**
     &                                  (-1.0d0/nairbranch))
                      end if
c
c Newton Raphson iterations
```



forward BSOLVER.txt - Notepad

File Edit Format View Help

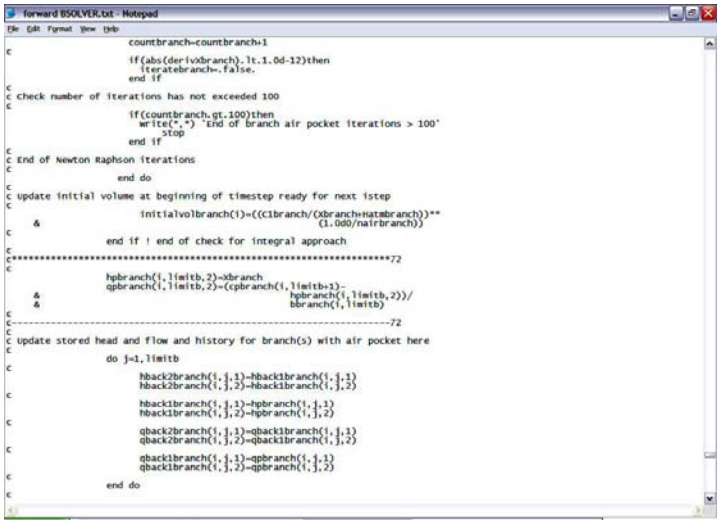
```
C Newton Raphson iterations
C
C Set iterate logical
C
                    iteratebranch=.true.
                    countbranch=1
C
                    do while (iteratebranch) ! converge tolerance to limit iterations
C
C Reversing Qair sign from JV (Qair out of chamber +ve) to SW convention (Qair in
C to chamber +ve) leads to the same result in the derivations and the resulting
C code below
C
                    fXobranch=Xbranch-cpbranch(i,limitb+1)-C1branch*
     &                   (((Xbranch+Hatmbranch)**
     &                        (-1.0d0/nairbranch))-C2branch)
C
                    dfXobranch=1.0d0+C1branch/nairbranch*
     &                  ((Xbranch+Hatmbranch)**
     &                        (-(1.0d0/nairbranch)-1.0d0))
C
                    derivXbranch=-fXobranch/dfXobranch
                    Xbranch=Xbranch+derivXbranch
C
                    countbranch=countbranch+1
C
                    if(abs(derivXbranch).lt.1.0d-12)then
                      iteratebranch=.false.
                    end if
C
C Check number of iterations has not exceeded 100
C
                    if(countbranch.gt.100)then
                      write(*,*) 'End of branch air pocket iterations > 100'
                        stop
                    end if
C
C End of Newton Raphson iterations
C
                    end do
C
                  end if ! end of check for derivative approach
C
C**********************************************************************72
C
C Do Newton Raphson iteration for air chamber head calculation
C
C INTEGRAL APPROXIMATION TO Qair - not staggered
C
                  if(integralbranch)then
C
```



forward BSOLVER.txt - Notepad

File Edit Format View Help

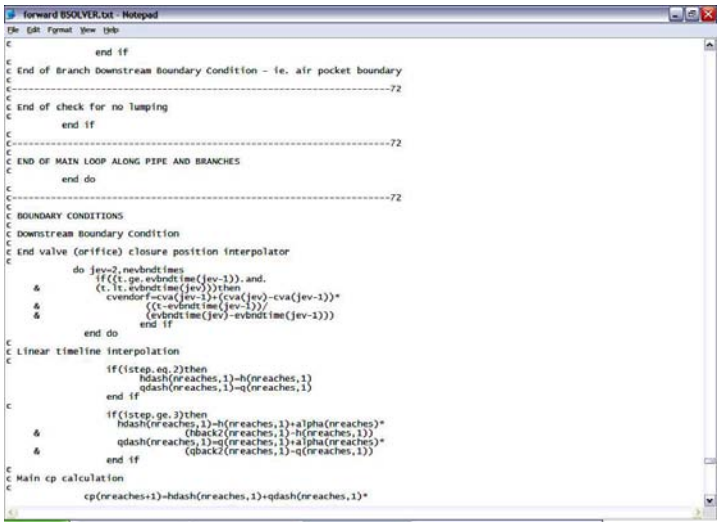
```
C
C Set initial head and volume values prior to iteration process
C
                    Xbranch=hback1branch(i,limitb,2)
C
                      if(istep.eq.2)then
                  initialvolbranch(i)=vobranch(i)
                      end if
C
C Calculate non-iterative constants
C
                      C1branch=(Hobranch(i)+Hatmbranch)*(Vobranch(i)**
     &                          nairbranch)
C
C Reversing Qair sign from JV (Qair out of chamber +ve) to SW convention (Qair in
C to chamber +ve) leads to the same result in the derivations and the resulting
C code below
C
                    if(istep.ge.2)then
                      C2branch=initialvolbranch(i)-((dt*integralweightbranch)
     &                              *cpbranch(i,limitb+1)/bbranch(i,limitb))-
     &                             (dt*(1.0d0-integralweightbranch)*
     &                             qback1branch(i,limitb,2))
                    end if
C
C Newton Raphson iterations
C
C Set iterate logical
C
                    iteratebranch=.true.
                    countbranch=1
C
                    do while (iteratebranch) ! converge tolerance to limit iterations
C
C Calculate variable C3
C
                      C3branch=dt*integralweightbranch*Xbranch/
     &                             bbranch(i,limitb)+C2branch
C
                    fXobranch=(Xbranch+Hatmbranch)*(C3branch**nairbranch)-
     &                          C1branch
C
                    dfXobranch=(C3branch**nairbranch)+nairbranch*dt*
     &                       integralweightbranch/bbranch(i,limitb)*
     &                       (Xbranch+Hatmbranch)*(C3branch**
     &                       (nairbranch-1.0d0))
C
                    derivXbranch=-fXobranch/dfXobranch
                    Xbranch=Xbranch+derivXbranch
C
                    countbranch=countbranch+1
```



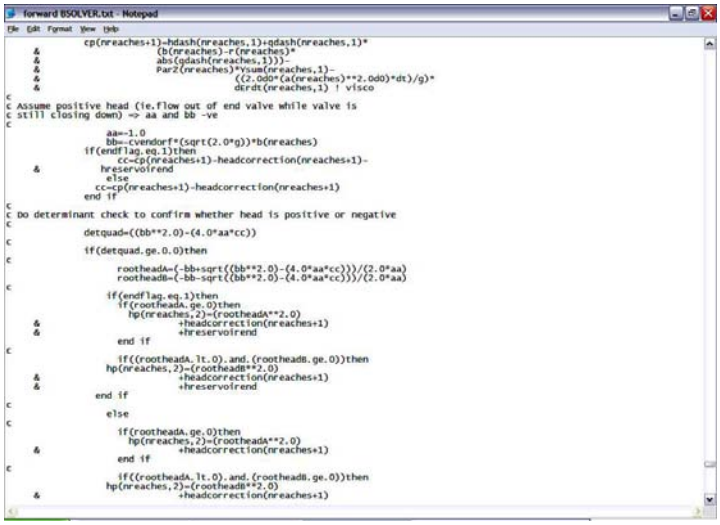
forward BSOLVER.txt - Notepad

File Edit Format View Help

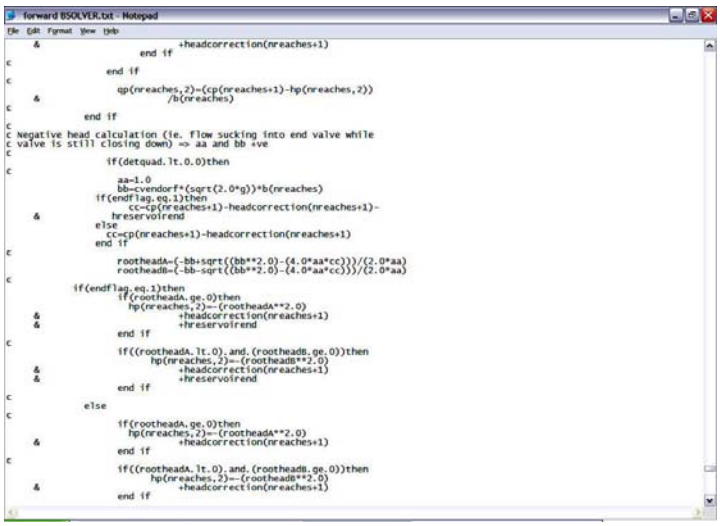
```
                    countbranch=countbranch+1
C
                    if(abs(derivXbranch).lt.1.0d-12)then
                      iteratebranch=.false.
                    end if
C
C Check number of iterations has not exceeded 100
C
                    if(countbranch.gt.100)then
                      write(*,*) 'End of branch air pocket iterations > 100'
                        stop
                    end if
C
C End of Newton Raphson iterations
C
                    end do
C
C Update initial volume at beginning of timestep ready for next istep
C
                      initialvolbranch(i)=(((C1branch/(Xbranch+Hatmbranch))**
     &                                          (1.0d0/nairbranch))
C
                  end if ! end of check for integral approach
C
C**********************************************************************72
C
                  hpbranch(i,limitb,2)=Xbranch
                  qpbranch(i,limitb,2)=(cpbranch(i,limitb+1)-
     &                                        hpbranch(i,limitb,2))/
     &                                        bbranch(i,limitb)
C
C----------------------------------------------------------------------72
C
C Update stored head and flow and history for branch(s) with air pocket here
C
                  do j=1,limitb
C
                    hback2branch(i,j,1)=hback1branch(i,j,1)
                    hback2branch(i,j,2)=hback1branch(i,j,2)
C
                    hback1branch(i,j,1)=hpbranch(i,j,1)
                    hback1branch(i,j,2)=hpbranch(i,j,2)
C
                    qback2branch(i,j,1)=qback1branch(i,j,1)
                    qback2branch(i,j,2)=qback1branch(i,j,2)
C
                    qback1branch(i,j,1)=qpbranch(i,j,1)
                    qback1branch(i,j,2)=qpbranch(i,j,2)
C
                  end do
C
```



```
forward BSOLVER.txt - Notepad
File  Edit  Format  View  Help
C                end if
C
C End of Branch Downstream Boundary Condition - ie. air pocket boundary
C
C-----------------------------------------------------------------------72
C
C End of check for no lumping
C
          end if
C
C-----------------------------------------------------------------------72
C
C END OF MAIN LOOP ALONG PIPE AND BRANCHES
C
          end do
C
C-----------------------------------------------------------------------72
C
C BOUNDARY CONDITIONS
C
C Downstream Boundary Condition
C
C End valve (orifice) closure position interpolator
C
            do jev=2,nevbndtimes
              if((t.ge.evbndtime(jev-1)).and.
     &          (t.lt.evbndtime(jev)))then
                cvendorf=cva(jev-1)+(cva(jev)-cva(jev-1))*
     &                   ((t-evbndtime(jev-1))/
     &                   (evbndtime(jev)-evbndtime(jev-1)))
                end if
            end do
C
C Linear timeline interpolation
C
              if(istep.eq.2)then
                    hdash(nreaches,1)=h(nreaches,1)
                    qdash(nreaches,1)=q(nreaches,1)
              end if
C
              if(istep.ge.3)then
                hdash(nreaches,1)=h(nreaches,1)+alpha(nreaches)*
     &                            (hback2(nreaches,1)-h(nreaches,1))
                qdash(nreaches,1)=q(nreaches,1)+alpha(nreaches)*
     &                            (qback2(nreaches,1)-q(nreaches,1))
              end if
C
C Main cp calculation
C
            cp(nreaches+1)=hdash(nreaches,1)+qdash(nreaches,1)*
```

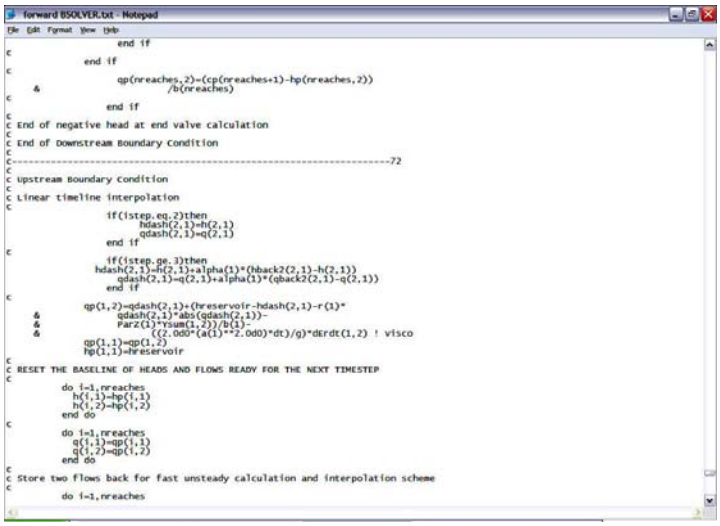


```
forward BSOLVER.txt - Notepad
File  Edit  Format  View  Help
            cp(nreaches+1)=hdash(nreaches,1)+qdash(nreaches,1)*
     &                     (b(nreaches)-r(nreaches)*
     &                     abs(qdash(nreaches,1)))-
     &                     Par2(nreaches)*ysum(nreaches,1)-
     &                                  ((2.0d0*(a(nreaches)**2.0d0)*dt)/g)*
     &                                  dcrdt(nreaches,1) ! visco
C
C Assume positive head (ie.flow out of end valve while valve is
C still closing down) => aa and bb -ve
C
              aa=-1.0
              bb=-cvendorf*(sqrt(2.0*g))*b(nreaches)
            if(endflag.eq.1)then
              cc=cp(nreaches+1)-headcorrection(nreaches+1)-
     &          hreservoirend
              else
             cc=cp(nreaches+1)-headcorrection(nreaches+1)
            end if
C
C Do determinant check to confirm whether head is positive or negative
C
            detquad=((bb**2.0)-(4.0*aa*cc))
C
            if(detquad.ge.0.0)then
C
                rootheadA=(-bb+sqrt((bb**2.0)-(4.0*aa*cc)))/(2.0*aa)
                rootheadB=(-bb-sqrt((bb**2.0)-(4.0*aa*cc)))/(2.0*aa)
C
              if(endflag.eq.1)then
                if(rootheadA.ge.0)then
                  hp(nreaches,2)=(rootheadA**2.0)
     &                           +headcorrection(nreaches+1)
     &                           +hreservoirend
                end if
C
                if((rootheadA.lt.0).and.(rootheadB.ge.0))then
              hp(nreaches,2)=(rootheadB**2.0)
     &                           +headcorrection(nreaches+1)
     &                           +hreservoirend
              end if
C
                else
C
                  if(rootheadA.ge.0)then
                    hp(nreaches,2)=(rootheadA**2.0)
     &                           +headcorrection(nreaches+1)
                  end if
C
                  if((rootheadA.lt.0).and.(rootheadB.ge.0))then
                hp(nreaches,2)=(rootheadB**2.0)
     &                           +headcorrection(nreaches+1)
```

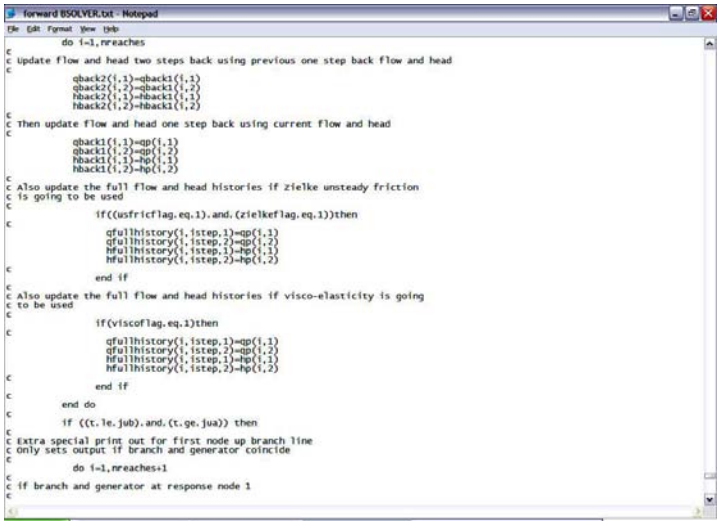


```
forward BSOLVER.txt - Notepad
File  Edit  Format  View  Help
     &                           +headcorrection(nreaches+1)
                    end if
C
              end if
C
                qp(nreaches,2)=(cp(nreaches+1)-hp(nreaches,2))
     &                         /b(nreaches)
C
            end if
C
C Negative head calculation (ie. flow sucking into end valve while
C valve is still closing down) => aa and bb +ve
C
              if(detquad.lt.0.0)then
C
                aa=1.0
                bb=cvendorf*(sqrt(2.0*g))*b(nreaches)
              if(endflag.eq.1)then
                cc=cp(nreaches+1)-headcorrection(nreaches+1)-
     &            hreservoirend
              else
                cc=cp(nreaches+1)-headcorrection(nreaches+1)
              end if
C
                rootheadA=(-bb+sqrt((bb**2.0)-(4.0*aa*cc)))/(2.0*aa)
                rootheadB=(-bb-sqrt((bb**2.0)-(4.0*aa*cc)))/(2.0*aa)
C
            if(endflag.eq.1)then
                if(rootheadA.ge.0)then
                  hp(nreaches,2)=-(rootheadA**2.0)
     &                           +headcorrection(nreaches+1)
     &                           +hreservoirend
                end if
C
                if((rootheadA.lt.0).and.(rootheadB.ge.0))then
                    hp(nreaches,2)=-(rootheadB**2.0)
     &                           +headcorrection(nreaches+1)
     &                           +hreservoirend
                end if
C
              else
C
                  if(rootheadA.ge.0)then
                    hp(nreaches,2)=-(rootheadA**2.0)
     &                           +headcorrection(nreaches+1)
                  end if
C
                  if((rootheadA.lt.0).and.(rootheadB.ge.0))then
                      hp(nreaches,2)=-(rootheadB**2.0)
     &                           +headcorrection(nreaches+1)
                  end if
```

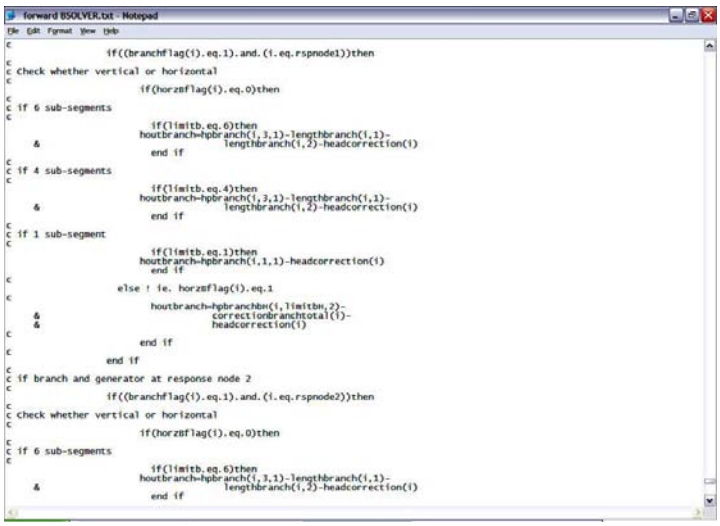
Appendix M – Fortran code for BSOLVER and NLFIT subroutines



```
forward BSOLVER.txt - Notepad
File  Edit  Format  View  Help
                        end if
C
                end if
C
                      qp(nreaches,2)=(cp(nreaches+1)-hp(nreaches,2))
     &                          /b(nreaches)
C
                    end if
C
C End of negative head at end valve calculation
C
C End of Downstream Boundary Condition
C
C--------------------------------------------------------------------72
C
C Upstream Boundary Condition
C
C Linear timeline interpolation
C
                    if(istep.eq.2)then
                          hdash(2,1)=h(2,1)
                          qdash(2,1)=q(2,1)
                    end if
C
                    if(istep.ge.3)then
                  hdash(2,1)=h(2,1)+alpha(1)*(hback2(2,1)-h(2,1))
                    qdash(2,1)=q(2,1)+alpha(1)*(qback2(2,1)-q(2,1))
                    end if
C
               qp(1,2)=qdash(2,1)+(hreservoir-hdash(2,1)-r(1)*
     &              qdash(2,1)*abs(qdash(2,1))-
     &               Parz(1)*Ysum(1,2))/b(1)-
     &                          ((2.0d0*(a(1)**2.0d0)*dt)/g)*dErdt(1,2) ! visco
               qp(1,1)=qp(1,2)
               hp(1,1)=hreservoir
C
C RESET THE BASELINE OF HEADS AND FLOWS READY FOR THE NEXT TIMESTEP
C
            do i=1,nreaches
              h(i,1)=hp(i,1)
              h(i,2)=hp(i,2)
            end do
C
            do i=1,nreaches
              q(i,1)=qp(i,1)
              q(i,2)=qp(i,2)
            end do
C
C Store two flows back for fast unsteady calculation and interpolation scheme
C
            do i=1,nreaches
```

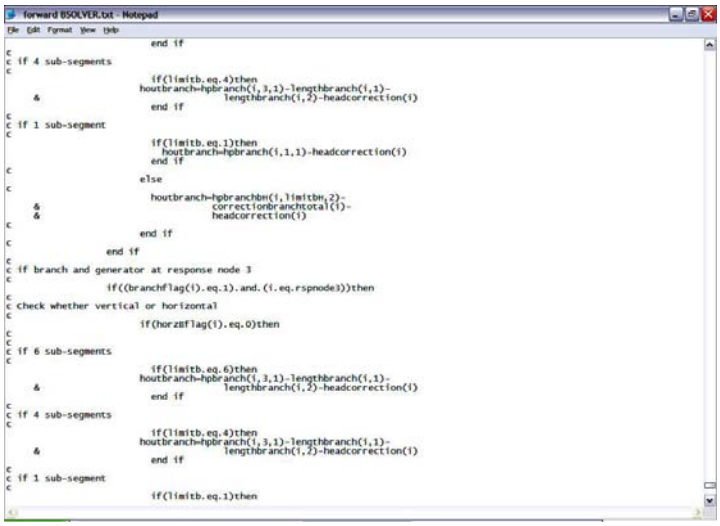


```
forward BSOLVER.txt - Notepad
File  Edit  Format  View  Help
            do i=1,nreaches
C
C Update flow and head two steps back using previous one step back flow and head
C
              qback2(i,1)=qback1(i,1)
              qback2(i,2)=qback1(i,2)
              hback2(i,1)=hback1(i,1)
              hback2(i,2)=hback1(i,2)
C
C Then update flow and head one step back using current flow and head
C
              qback1(i,1)=qp(i,1)
              qback1(i,2)=qp(i,2)
              hback1(i,1)=hp(i,1)
              hback1(i,2)=hp(i,2)
C
C Also update the full flow and head histories if Zielke unsteady friction
C is going to be used
C
                    if((usfricflag.eq.1).and.(zielkeflag.eq.1))then
C
                      qfullhistory(i,istep,1)=qp(i,1)
                      qfullhistory(i,istep,2)=qp(i,2)
                      hfullhistory(i,istep,1)=hp(i,1)
                      hfullhistory(i,istep,2)=hp(i,2)
C
                    end if
C
C Also update the full flow and head histories if visco-elasticity is going
C to be used
C
                    if(viscoflag.eq.1)then
C
                      qfullhistory(i,istep,1)=qp(i,1)
                      qfullhistory(i,istep,2)=qp(i,2)
                      hfullhistory(i,istep,1)=hp(i,1)
                      hfullhistory(i,istep,2)=hp(i,2)
C
                    end if
C
            end do
C
            if ((t.le.jub).and.(t.ge.jua)) then
C
C Extra special print out for first node up branch line
C only sets output if branch and generator coincide
C
              do i=1,nreaches+1
C
C if branch and generator at response node 1
C
```

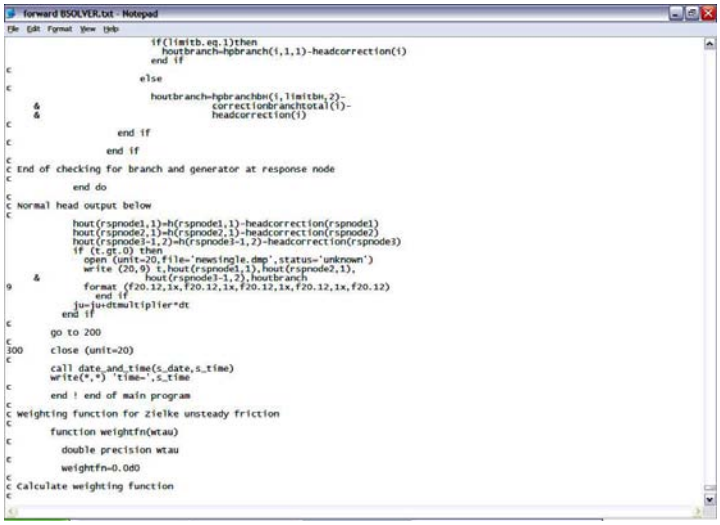


```
forward BSOLVER.txt - Notepad
File  Edit  Format  View  Help
C
                    if((branchflag(i).eq.1).and.(i.eq.rspnode1))then
C
C Check whether vertical or horizontal
C
                          if(horzbflag(i).eq.0)then
C
C if 6 sub-segments
C
                            if(limitb.eq.6)then
                          houtbranch=hpbranch(i,3,1)-lengthbranch(i,1)-
     &                                  lengthbranch(i,2)-headcorrection(i)
                            end if
C
C if 4 sub-segments
C
                            if(limitb.eq.4)then
                          houtbranch=hpbranch(i,3,1)-lengthbranch(i,1)-
     &                                  lengthbranch(i,2)-headcorrection(i)
                            end if
C
C if 1 sub-segment
C
                            if(limitb.eq.1)then
                          houtbranch=hpbranch(i,1,1)-headcorrection(i)
                            end if
C
                        else ! ie. horzbflag(i).eq.1
C
                            houtbranch=hpbranchbn(i,limitbn,2)-
     &                                  correctionbranchtotal(i)-
     &                                  headcorrection(i)
C
                          end if
C
                    end if
C
C if branch and generator at response node 2
C
                    if((branchflag(i).eq.1).and.(i.eq.rspnode2))then
C
C Check whether vertical or horizontal
C
                          if(horzbflag(i).eq.0)then
C
C if 6 sub-segments
C
                            if(limitb.eq.6)then
                          houtbranch=hpbranch(i,3,1)-lengthbranch(i,1)-
     &                                  lengthbranch(i,2)-headcorrection(i)
                            end if
```

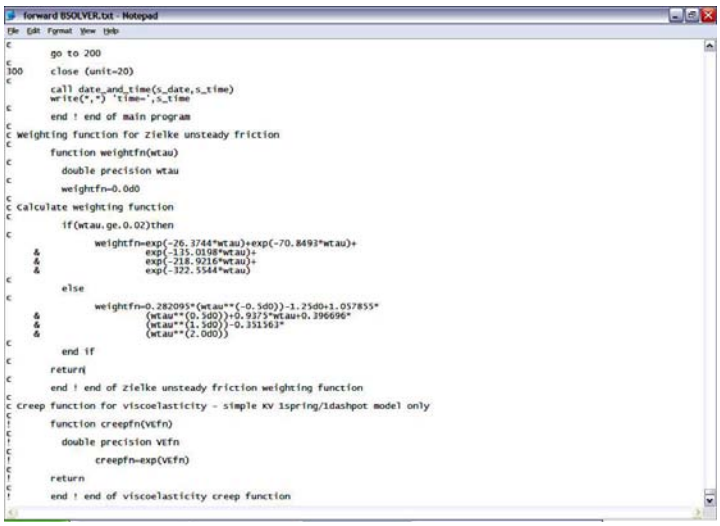
Appendix M – Fortran code for BSOLVER and NLFIT subroutines



```
forward BSOLVER.txt - Notepad
File  Edit  Format  View  Help
                              end if
C
C if 4 sub-segments
C
                              if(limitb.eq.4)then
                            houtbranch=hpbranch(i,3,1)-lengthbranch(i,1)-
     &                                    lengthbranch(i,2)-headcorrection(i)
                              end if
C
C if 1 sub-segment
C
                              if(limitb.eq.1)then
                                houtbranch=hpbranch(i,1,1)-headcorrection(i)
                              end if
C
                            else
C
                              houtbranch=hpbranchbn(i,limitbn,2)-
     &                                    correctionbranchtotal(i)-
     &                                    headcorrection(i)
C
                            end if
C
                          end if
C
C if branch and generator at response node 3
C
                        if((branchflag(i).eq.1).and.(i.eq.rspnode3))then
C
C Check whether vertical or horizontal
C
                            if(horzBflag(i).eq.0)then
C
C
C if 6 sub-segments
C
                              if(limitb.eq.6)then
                            houtbranch=hpbranch(i,3,1)-lengthbranch(i,1)-
     &                                    lengthbranch(i,2)-headcorrection(i)
                              end if
C
C if 4 sub-segments
C
                              if(limitb.eq.4)then
                            houtbranch=hpbranch(i,3,1)-lengthbranch(i,1)-
     &                                    lengthbranch(i,2)-headcorrection(i)
                              end if
C
C if 1 sub-segment
C
                              if(limitb.eq.1)then
```



```
forward BSOLVER.txt - Notepad
File  Edit  Format  View  Help
                              if(limitb.eq.1)then
                                houtbranch=hpbranch(i,1,1)-headcorrection(i)
                              end if
C
                            else
C
                              houtbranch=hpbranchbn(i,limitbn,2)-
     &                                    correctionbranchtotal(i)-
     &                                    headcorrection(i)
C
                          end if
C
                        end if
C
C End of checking for branch and generator at response node
C
                end do
C
C Normal head output below
C
                hout(rspnode1,1)=h(rspnode1,1)-headcorrection(rspnode1)
                hout(rspnode2,1)=h(rspnode2,1)-headcorrection(rspnode2)
                hout(rspnode3-1,2)=h(rspnode3-1,2)-headcorrection(rspnode3)
                if (t.gt.0) then
                  open (unit=20,file='newsingle.dmp',status='unknown')
                  write (20,9) t,hout(rspnode1,1),hout(rspnode2,1),
     &                         hout(rspnode3-1,2),houtbranch
9                 format (f20.12,1x,f20.12,1x,f20.12,1x,f20.12,1x,f20.12)
                    end if
                  ju=ju+dtmultiplier*dt
              end if
C
            go to 200
C
300         close (unit=20)
C
            call date_and_time(s_date,s_time)
            write(*,*) 'time=',s_time
C
            end ! end of main program
C
C Weighting function for Zielke unsteady friction
C
            function weightfn(wtau)
C
              double precision wtau
C
              weightfn=0.0d0
C
C Calculate weighting function
C
```

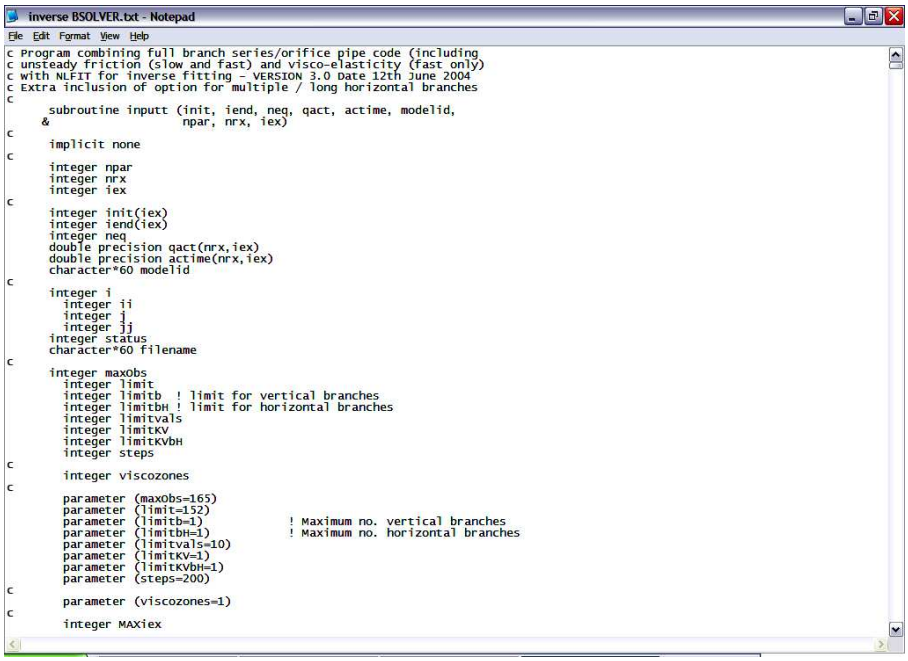


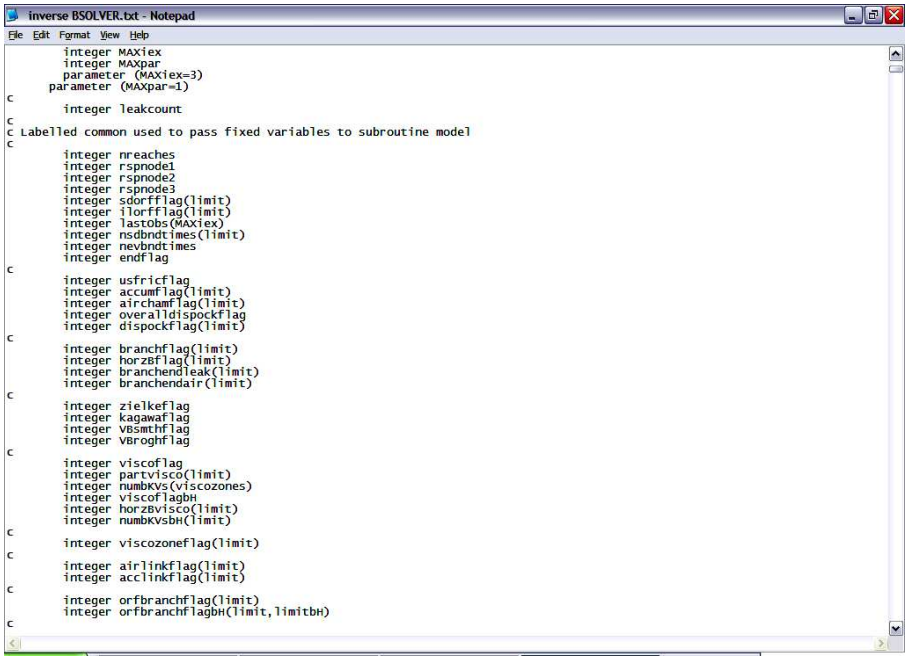
```
forward BSOLVER.txt - Notepad
File  Edit  Format  View  Help
C
            go to 200
C
300         close (unit=20)
C
            call date_and_time(s_date,s_time)
            write(*,*) 'time=',s_time
C
            end ! end of main program
C
C Weighting function for Zielke unsteady friction
C
            function weightfn(wtau)
C
              double precision wtau
C
              weightfn=0.0d0
C
C Calculate weighting function
C
              if(wtau.ge.0.02)then
C
                    weightfn=exp(-26.3744*wtau)+exp(-70.8493*wtau)+
     &                       exp(-135.0198*wtau)+
     &                       exp(-218.9216*wtau)+
     &                       exp(-322.5544*wtau)
C
              else
C
                    weightfn=0.282095*(wtau**(-0.5d0))-1.25d0+1.057855*
     &                       (wtau**(0.5d0))+0.9375*wtau+0.396696*
     &                       (wtau**(1.5d0))-0.331563*
     &                       (wtau**(2.0d0))
C
              end if
C
            return
C
            end ! end of Zielke unsteady friction weighting function
C
C Creep function for viscoelasticity - simple kV 1spring/1dashpot model only
C
!           function creepfn(Vtfn)
C
!             double precision Vtfn
C
!                   creepfn=exp(Vtfn)
C
!           return
C
!           end ! end of viscoelasticity creep function
```

M.2 Inverse transient subroutines for NLFIT

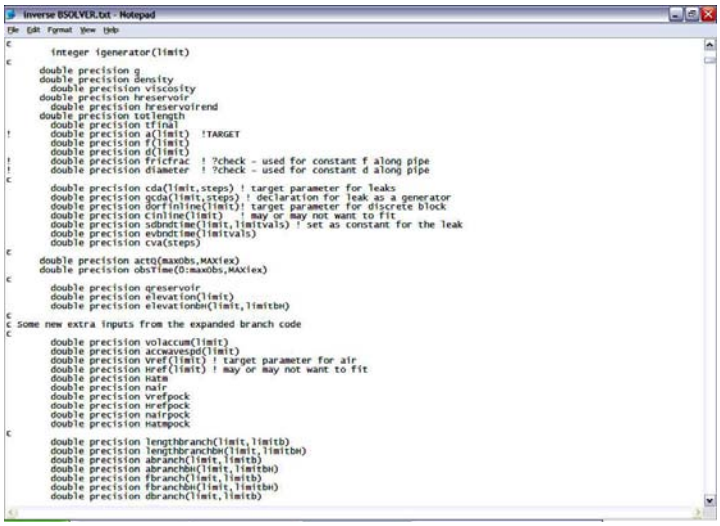
The INPUT and MODEL subroutines required by NLFIT are reproduced below. The subroutine form of BSOLVER, which is called by MODEL, is not reproduced.

Subroutine INPUT

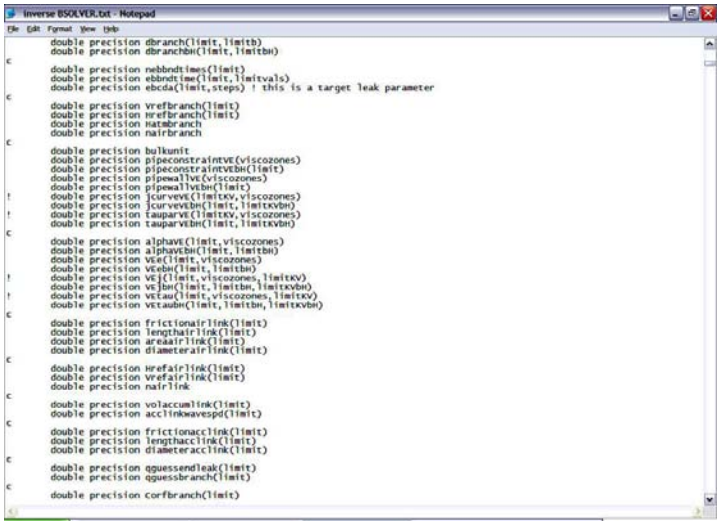




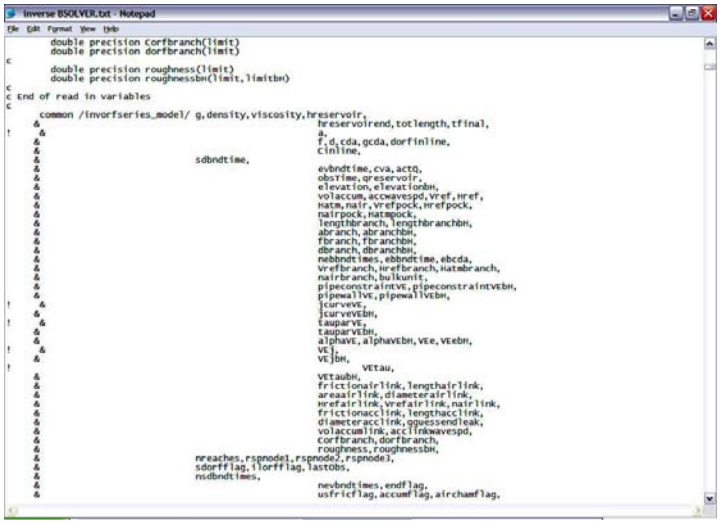
Appendix M – Fortran code for BSOLVER and NLFIT subroutines



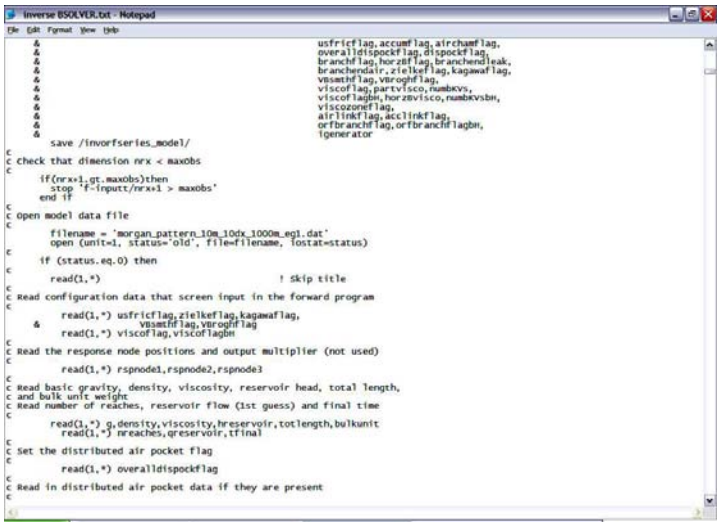
```
inverse BSOLVER.txt - Notepad
File  Edit  Format  View  Help
C
      integer igenerator(limit)
C
      double precision g
      double precision density
        double precision viscosity
      double precision hreservoir
        double precision hreservoirend
      double precision totlength
        double precision tfinal
!       double precision a(limit)  !TARGET
        double precision f(limit)
        double precision d(limit)
!       double precision fricfrac  ! ?check - used for constant f along pipe
!       double precision diameter  ! ?check - used for constant d along pipe
C
        double precision cda(limit,steps) ! target parameter for leaks
        double precision gcda(limit,steps) ! declaration for leak as a generator
        double precision dorfinline(limit)! target parameter for discrete block
        double precision cinline(limit)   ! may or may not want to fit
        double precision sdbndtime(limit,limitvals) ! set as constant for the leak
        double precision evbndtime(limitvals)
        double precision cva(steps)
C
      double precision actq(maxobs,MAXiex)
      double precision obsTime(0:maxobs,MAXiex)
C
        double precision qreservoir
        double precision elevation(limit)
        double precision elevationbh(limit,limitbh)
C
C Some new extra inputs from the expanded branch code
C
        double precision volaccum(limit)
        double precision accwavespd(limit)
        double precision vref(limit) ! target parameter for air
        double precision href(limit) ! may or may not want to fit
        double precision hatm
        double precision nair
        double precision vrefpock
        double precision hrefpock
        double precision nairpock
        double precision hatmpock
C
        double precision lengthbranch(limit,limitb)
        double precision lengthbranchbh(limit,limitbh)
        double precision abranch(limit,limitb)
        double precision abranchbh(limit,limitbh)
        double precision fbranch(limit,limitb)
        double precision fbranchbh(limit,limitbh)
        double precision dbranch(limit,limitb)
```

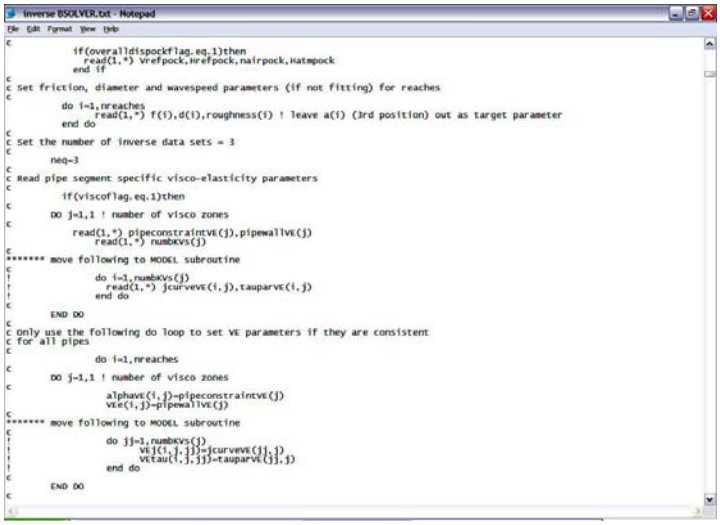


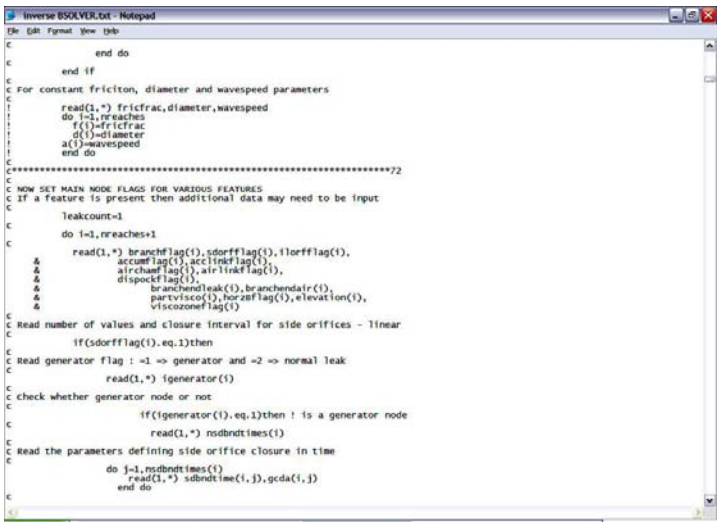
```
inverse BSOLVER.txt - Notepad
File  Edit  Format  View  Help
        double precision dbranch(limit,limitb)
        double precision dbranchbh(limit,limitbh)
C
        double precision nebbndtimes(limit)
        double precision ebbndtime(limit,limitvals)
        double precision ebcda(limit,steps) ! this is a target leak parameter
C
        double precision vrefbranch(limit)
        double precision hrefbranch(limit)
        double precision hatmbranch
        double precision nairbranch
C
        double precision bulkunit
        double precision pipeconstraintVE(viscozones)
        double precision pipeconstraintVEbh(limit)
        double precision pipewallVE(viscozones)
        double precision pipewallVEbh(limit)
!       double precision jcurveVE(limitKV,viscozones)
!       double precision jcurveVEbh(limit,limitKVbh)
!       double precision tauparVE(limitKV,viscozones)
        double precision tauparVEbh(limit,limitKVbh)
C
        double precision alphaVE(limit,viscozones)
        double precision alphaVEbh(limit,limitbh)
        double precision VEe(limit,viscozones)
        double precision VEebh(limit,limitbh)
!       double precision VEj(limit,viscozones,limitKV)
!       double precision VEjbh(limit,limitbh,limitKVbh)
!       double precision VEtau(limit,viscozones,limitKV)
        double precision VEtaubh(limit,limitbh,limitKVbh)
C
        double precision frictionairlink(limit)
        double precision lengthairlink(limit)
        double precision areaairlink(limit)
        double precision diameterairlink(limit)
C
        double precision Hrefairlink(limit)
        double precision Vrefairlink(limit)
        double precision nairlink
C
        double precision volaccumlink(limit)
        double precision acclinkwavespd(limit)
C
        double precision frictionacclink(limit)
        double precision lengthacclink(limit)
        double precision diameteracclink(limit)
C
        double precision qguessendleak(limit)
        double precision qguessbranch(limit)
C
        double precision corfbranch(limit)
```

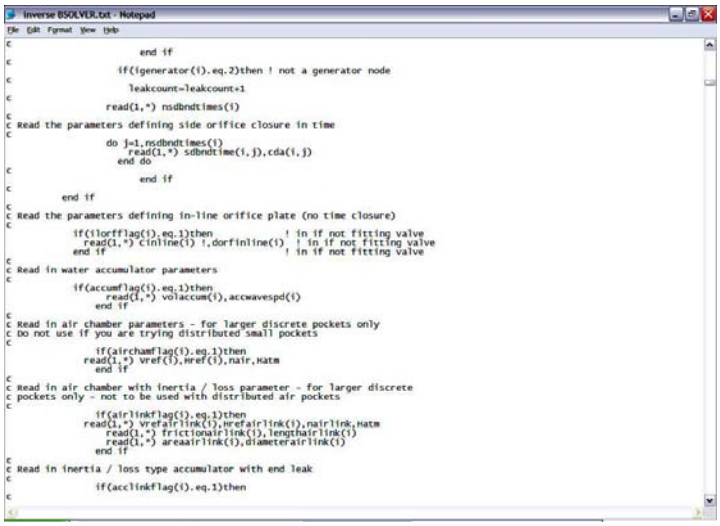


```
inverse BSOLVER.txt - Notepad
File  Edit  Format  View  Help
        double precision corfbranch(limit)
        double precision dorfbranch(limit)
C
        double precision roughness(limit)
        double precision roughnessbh(limit,limitbh)
C
C End of read in variables
C
     common /invorfseries_model/ g,density,viscosity,hreservoir,
     &                            hreservoirend,totlength,tfinal,
!    &                            a,
     &                            f,d,cda,gcda,dorfinline,
     &                            cinline,
     &                  sdbndtime,
     &                            evbndtime,cva,actq,
     &                            obsTime,qreservoir,
     &                            elevation,elevationbh,
     &                            volaccum,accwavespd,vref,href,
     &                            hatm,nair,vrefpock,hrefpock,
     &                            nairpock,hatmpock,
     &                            lengthbranch,lengthbranchbh,
     &                            abranch,abranchbh,
     &                            fbranch,fbranchbh,
     &                            dbranch,dbranchbh,
     &                            nebbndtimes,ebbndtime,ebcda,
     &                            vrefbranch,hrefbranch,hatmbranch,
     &                            nairbranch,bulkunit,
     &                            pipeconstraintVE,pipeconstraintVEbh,
     &                            pipewallVE,pipewallVEbh,
!    &                            jcurveVE,
     &                            jcurveVEbh,
!    &                            tauparVE,
     &                            tauparVEbh,
     &                            alphaVE,alphaVEbh,VEe,VEebh,
!    &                            VEj,
     &                            VEjbh,
!    &                                  VEtau,
     &                            VEtaubh,
     &                            frictionairlink,lengthairlink,
     &                            areaairlink,diameterairlink,
     &                            hrefairlink,vrefairlink,nairlink,
     &                            frictionacclink,lengthacclink,
     &                            diameteracclink,qguessendleak,
     &                            volaccumlink,acclinkwavespd,
     &                            corfbranch,dorfbranch,
     &                            roughness,roughnessbh,
     &                  nreaches,rspnode1,rspnode2,rspnode3,
     &                  sdorfflag,ilorfflag,lastobs,
     &                  nsdbndtimes,
     &                            nevbndtimes,endflag,
     &                            usfricflag,accumflag,airchamflag,
```

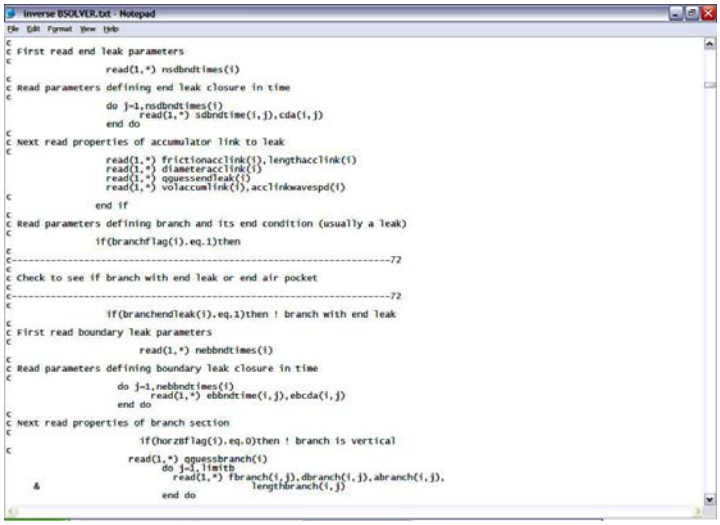

inverse BSOLVER.txt - Notepad
File Edit Format View Help
usfricflag,accumflag,airchamflag,
overalldispockflag,dispockflag,
branchflag,horzBflag,branchendleak,
branchendair,zielkeflag,kagawaflag,
vbsmthflag,vbroghflag,
viscoflag,partvisco,numbKvs,
viscoflagbH,horzBvisco,numbKvsbH,
viscozoneflag,
airlinkflag,acclinkflag,
orfbranchflag,orfbranchflagbH,
igenerator
save /invorfseries_model/
check that dimension nrx < maxobs
if(nrx+1.gt.maxobs)then
stop 'f-inputt/nrx+1 > maxobs'
end if
Open model data file
filename = 'morgan_pattern_10m_10dx_1000m_eg1.dat'
open (unit=1, status='old', file=filename, iostat=status)
if (status.eq.0) then
read(1,*) ! Skip title
Read configuration data that screen input in the forward program
read(1,*) usfricflag,zielkeflag,kagawaflag,
vbsmthflag,vbroghflag
read(1,*) viscoflag,viscoflagbH
Read the response node positions and output multiplier (not used)
read(1,*) rspnode1,rspnode2,rspnode3
Read basic gravity, density, viscosity, reservoir head, total length,
and bulk unit weight
Read number of reaches, reservoir flow (1st guess) and final time
read(1,*) g,density,viscosity,hreservoir,totlength,bulkunit
read(1,*) nreaches,qreservoir,tfinal
Set the distributed air pocket flag
read(1,*) overalldispockflag
Read in distributed air pocket data if they are present


inverse BSOLVER.txt - Notepad
File Edit Format View Help
if(overalldispockflag.eq.1)then
read(1,*) Vrefpock,hrefpock,hairpock,Hatmpock
end if
Set friction, diameter and wavespeed parameters (if not fitting) for reaches
do i=1,nreaches
read(1,*) f(i),d(i),roughness(i) ! leave a(i) (3rd position) out as target parameter
end do
Set the number of inverse data sets = 3
neq=3
Read pipe segment specific visco-elasticity parameters
if(viscoflag.eq.1)then
DO j=1,1 ! number of visco zones
read(1,*) pipeconstraintVE(j),pipewallVE(j)
read(1,*) numbKvs(j)
****** move following to MODEL subroutine
do i=1,numbKvs(j)
read(1,*) jcurveVE(i,j),tauparVE(i,j)
end do
END DO
Only use the following do loop to set VE parameters if they are consistent
for all pipes
do i=1,nreaches
DO j=1,1 ! number of visco zones
alphaVE(i,j)=pipeconstraintVE(j)
VkE(i,j)=pipewallVE(j)
****** move following to MODEL subroutine
do jj=1,numbKvs(j)
VEj(i,j,jj)=jcurveVE(jj,j)
VEtau(i,j,jj)=tauparVE(jj,j)
end do
END DO

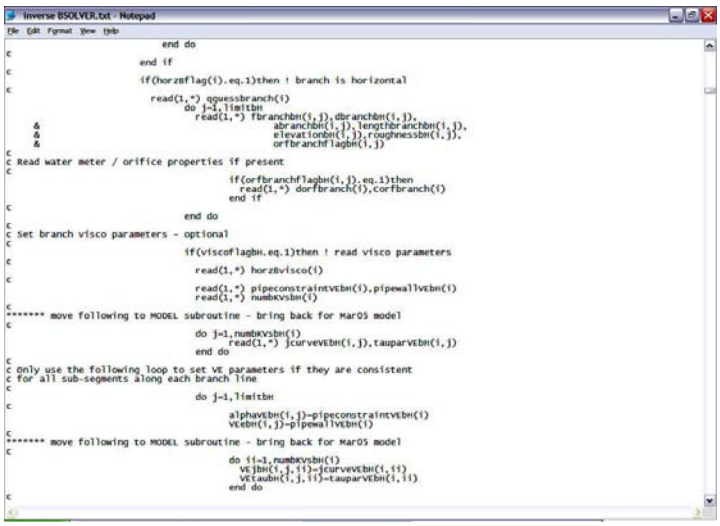

inverse BSOLVER.txt - Notepad
File Edit Format View Help
end do
end if
For constant friciton, diameter and wavespeed parameters
read(1,*) fricfrac,diameter,wavespeed
do i=1,nreaches
f(i)=fricfrac
d(i)=diameter
a(i)=wavespeed
end do
72
NOW SET MAIN NODE FLAGS FOR VARIOUS FEATURES
If a feature is present then additional data may need to be input
leakcount=1
do i=1,nreaches+1
read(1,*) branchflag(i),sdorfflag(i),ilorfflag(i),
accumflag(i),acclinkflag(i),
airchamflag(i),airlinkflag(i),
dispockflag(i),
branchendleak(i),branchendair(i),
partvisco(i),horzBflag(i),elevation(i),
viscozoneflag(i)
Read number of values and closure interval for side orifices - linear
if(sdorfflag(i).eq.1)then
Read generator flag : =1 => generator and =2 => normal leak
read(1,*) igenerator(i)
Check whether generator node or not
if(igenerator(i).eq.1)then ! is a generator node
read(1,*) nsdbndtimes(i)
Read the parameters defining side orifice closure in time
do j=1,nsdbndtimes(i)
read(1,*) sdbndtime(i,j),gcda(i,j)
end do



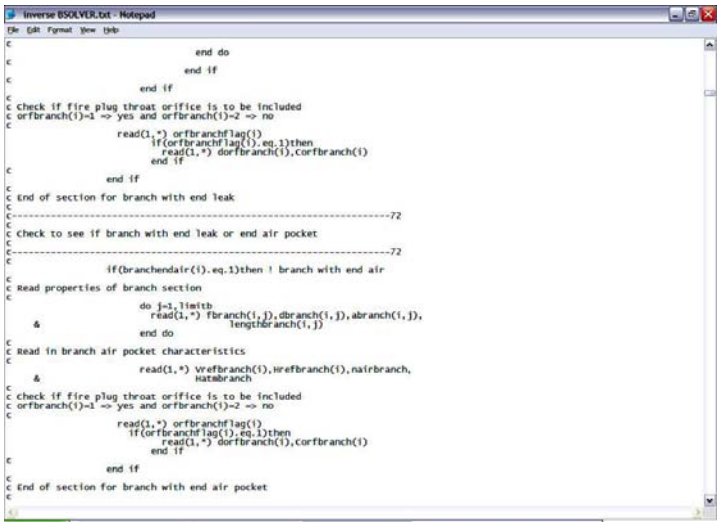
```
inverse BSOLVER.txt - Notepad
File  Edit  Format  View  Help
c                        end if
c
c                      if(igenerator(i).eq.2)then ! not a generator node
c
c                        leakcount=leakcount+1
c
c                      read(1,*) nsdbndtimes(i)
c
c Read the parameters defining side orifice closure in time
c
c                      do j=1,nsdbndtimes(i)
c                        read(1,*) sdbndtime(i,j),cda(i,j)
c                      end do
c
c                        end if
c
c             end if
c
c Read the parameters defining in-line orifice plate (no time closure)
c
c             if(ilorfflag(i).eq.1)then              ! in if not fitting valve
c               read(1,*) cinline(i) !,dorfinline(i)  ! in if not fitting valve
c             end if                                 ! in if not fitting valve
c
c Read in water accumulator parameters
c
c             if(accumflag(i).eq.1)then
c                  read(1,*) volaccum(i),accwavespd(i)
c               end if
c
c Read in air chamber parameters - for larger discrete pockets only
c Do not use if you are trying distributed small pockets
c
c                if(airchamflag(i).eq.1)then
c               read(1,*) vref(i),href(i),nair,Hatm
c                end if
c
c Read in air chamber with inertia / loss parameter - for larger discrete
c pockets only - not to be used with distributed air pockets
c
c                if(airlinkflag(i).eq.1)then
c              read(1,*) vrefairlink(i),hrefairlink(i),nairlink,Hatm
c                 read(1,*) frictionairlink(i),lengthairlink(i)
c                 read(1,*) areaairlink(i),diameterairlink(i)
c               end if
c
c Read in inertia / loss type accumulator with end leak
c
c                if(acclinkflag(i).eq.1)then
c
```



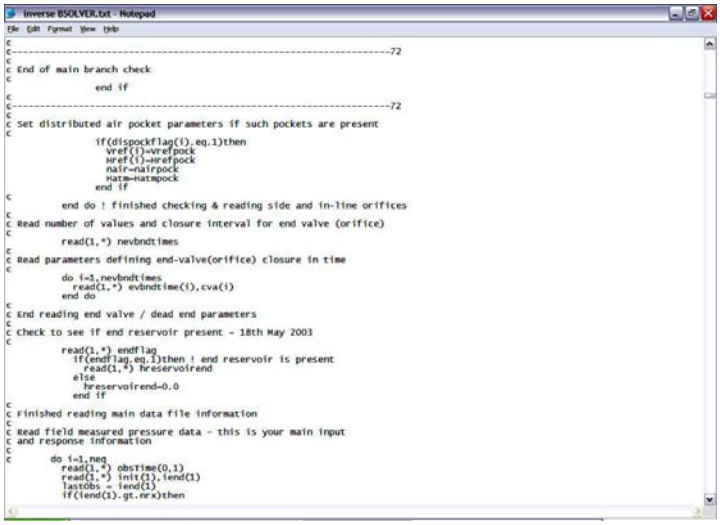
```
inverse BSOLVER.txt - Notepad
File  Edit  Format  View  Help
c
c First read end leak parameters
c
c                 read(1,*) nsdbndtimes(i)
c
c Read parameters defining end leak closure in time
c
c                 do j=1,nsdbndtimes(i)
c                     read(1,*) sdbndtime(i,j),cda(i,j)
c                 end do
c
c Next read properties of accumulator link to leak
c
c                 read(1,*) frictionacclink(i),lengthacclink(i)
c                 read(1,*) diameteracclink(i)
c                 read(1,*) qguessendleak(i)
c                 read(1,*) volaccumlink(i),acclinkwavespd(i)
c
c                end if
c
c Read parameters defining branch and its end condition (usually a leak)
c
c                if(branchflag(i).eq.1)then
c
c-----------------------------------------------------------------------72
c
c Check to see if branch with end leak or end air pocket
c
c-----------------------------------------------------------------------72
c
c                 if(branchendleak(i).eq.1)then ! branch with end leak
c
c First read boundary leak parameters
c
c                       read(1,*) nebbndtimes(i)
c
c Read parameters defining boundary leak closure in time
c
c                  do j=1,nebbndtimes(i)
c                       read(1,*) ebbndtime(i,j),ebcda(i,j)
c                  end do
c
c Next read properties of branch section
c
c                       if(horzbflag(i).eq.0)then ! branch is vertical
c
c                     read(1,*) qguessbranch(i)
c                          do j=1,limitb
c                            read(1,*) fbranch(i,j),dbranch(i,j),abranch(i,j),
c    &                                  lengthbranch(i,j)
c                          end do
```



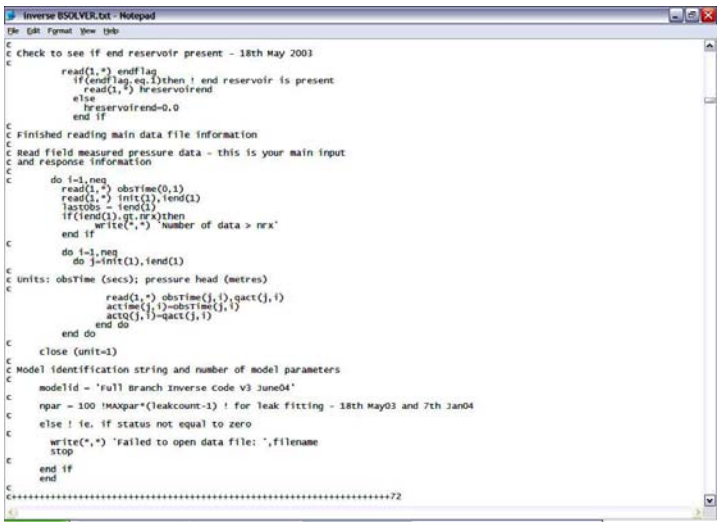
```
inverse BSOLVER.txt - Notepad
File  Edit  Format  View  Help
c                          end do
c
c                       end if
c
c                       if(horzmflag(i).eq.1)then ! branch is horizontal
c
c                         read(1,*) qguessbranch(i)
c                                do j=1,limitbh
c                                  read(1,*) fbranchbh(i,j),dbranchbh(i,j),
c    &                                       abranchbh(i,j),lengthbranchbh(i,j),
c    &                                       elevationbh(i,j),roughnessbh(i,j),
c    &                                       orfbranchflagbh(i,j)
c
c Read water meter / orifice properties if present
c
c                                       if(orfbranchflagbh(i,j).eq.1)then
c                                         read(1,*) dorfbranch(i),corfbranch(i)
c                                       end if
c
c                                end do
c
c Set branch visco parameters - optional
c
c                                if(viscoflagbh.eq.1)then ! read visco parameters
c
c                                  read(1,*) horzbvisco(i)
c
c                                  read(1,*) pipeconstraintvtbh(i),pipewallvtbh(i)
c                                  read(1,*) numbkvsbh(i)
c
****** move following to MODEL subroutine - bring back for MarOS model
c                                  do j=1,numbkvsbh(i)
c                                       read(1,*) jcurvevtbh(i,j),tauparvtbh(i,j)
c                                  end do
c
c Only use the following loop to set VE parameters if they are consistent
c for all sub-segments along each branch line
c                                  do j=1,limitbh
c
c                                       alphavtbh(i,j)=pipeconstraintvtbh(i)
c                                       vtebh(i,j)=pipewallvtbh(i)
c
****** move following to MODEL subroutine - bring back for MarOS model
c                                  do ii=1,numbkvsbh(i)
c                                    vtjbh(i,j,ii)=jcurvevtbh(i,ii)
c                                    vtaubh(i,j,ii)=tauparvtbh(i,ii)
c                                  end do
c
```




```
                                    end do
C
                                  end if
C
                              end if
C
C Check if fire plug throat orifice is to be included
C orfbranch(i)=1 => yes and orfbranch(i)=2 => no
C
                        read(1,*) orfbranchflag(i)
                          if(orfbranchflag(i).eq.1)then
                            read(1,*) dorfbranch(i),corfbranch(i)
                          end if
C
                      end if
C
C End of section for branch with end leak
C
C-----------------------------------------------------------------------72
C
C Check to see if branch with end leak or end air pocket
C
C-----------------------------------------------------------------------72
C
                      if(branchendair(i).eq.1)then ! branch with end air
C
C Read properties of branch section
C
                        do j=1,limitb
                          read(1,*) fbranch(i,j),dbranch(i,j),abranch(i,j),
     &                                lengthbranch(i,j)
                        end do
C
C Read in branch air pocket characteristics
C
                        read(1,*) Vrefbranch(i),Hrefbranch(i),nairbranch,
     &                            Hatmbranch
C
C Check if fire plug throat orifice is to be included
C orfbranch(i)=1 => yes and orfbranch(i)=2 => no
C
                        read(1,*) orfbranchflag(i)
                          if(orfbranchflag(i).eq.1)then
                            read(1,*) dorfbranch(i),corfbranch(i)
                          end if
C
                      end if
C
C End of section for branch with end air pocket
C
```

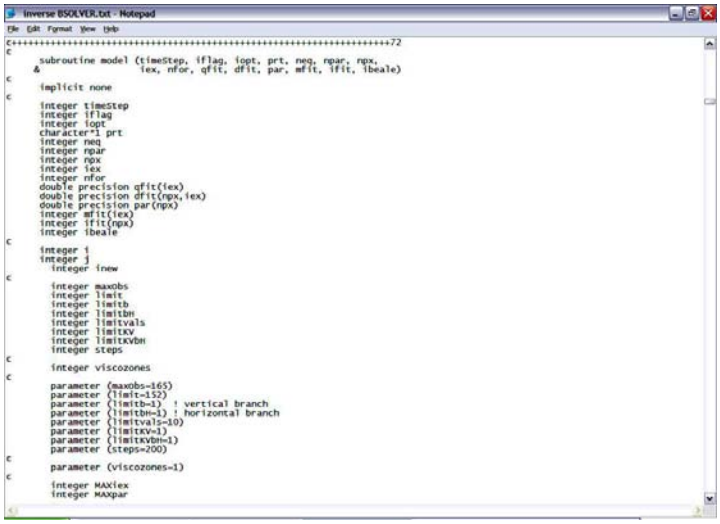



```
C
C-----------------------------------------------------------------------72
C
C End of main branch check
C
                    end if
C
C-----------------------------------------------------------------------72
C
C Set distributed air pocket parameters if such pockets are present
C
                  if(dispockflag(i).eq.1)then
                    Vref(i)=Vrefpock
                    Href(i)=Hrefpock
                    nair=nairpock
                    Hatm=Hatmpock
                  end if
C
          end do ! finished checking & reading side and in-line orifices
C
C Read number of values and closure interval for end valve (orifice)
C
          read(1,*) nevbndtimes
C
C Read parameters defining end-valve(orifice) closure in time
C
          do i=1,nevbndtimes
            read(1,*) evbndtime(i),cva(i)
          end do
C
C End reading end valve / dead end parameters
C
C Check to see if end reservoir present - 18th May 2003
C
          read(1,*) endflag
            if(endflag.eq.1)then ! end reservoir is present
              read(1,*) hreservoirend
            else
              hreservoirend=0.0
            end if
C
C Finished reading main data file information
C
C Read field measured pressure data - this is your main input
C and response information
C
C       do i=1,neq
          read(1,*) obstime(0,1)
          read(1,*) init(1),iend(1)
          lastobs = iend(1)
          if(iend(1).gt.nrx)then
```

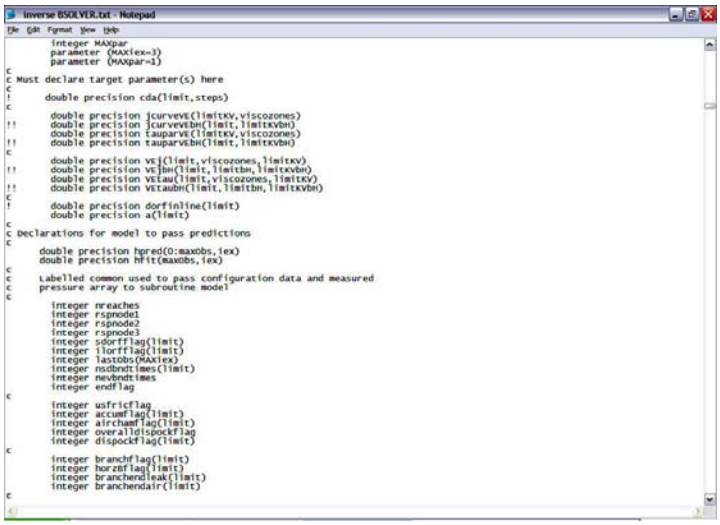



```
C
C Check to see if end reservoir present - 18th May 2003
C
          read(1,*) endflag
            if(endflag.eq.1)then ! end reservoir is present
              read(1,*) hreservoirend
            else
              hreservoirend=0.0
            end if
C
C Finished reading main data file information
C
C Read field measured pressure data - this is your main input
C and response information
C
C       do i=1,neq
          read(1,*) obstime(0,1)
          read(1,*) init(1),iend(1)
          lastobs = iend(1)
          if(iend(1).gt.nrx)then
            write(*,*) 'Number of data > nrx'
          end if
C
          do i=1,neq
            do j=init(1),iend(1)
C
C Units: obsTime (secs); pressure head (metres)
C
              read(1,*) obsTime(j,i),qact(j,i)
              actime(j,i)=obsTime(j,i)
              actq(j,i)=qact(j,i)
            end do
          end do
C
      close (unit=1)
C
C Model identification string and number of model parameters
C
      modelid = 'Full Branch Inverse Code V3 June04'
C
      npar = 100 !MAXpar*(leakcount-1) ! for leak fitting - 18th May03 and 7th Jan04
C
      else ! ie. if status not equal to zero
C
        write(*,*) 'Failed to open data file: ',filename
        stop
C
      end if
      end
C
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++72
```

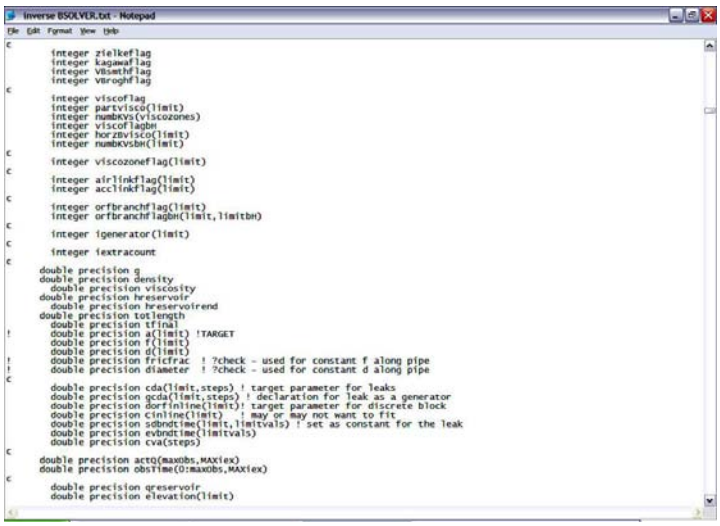
Subroutine MODEL



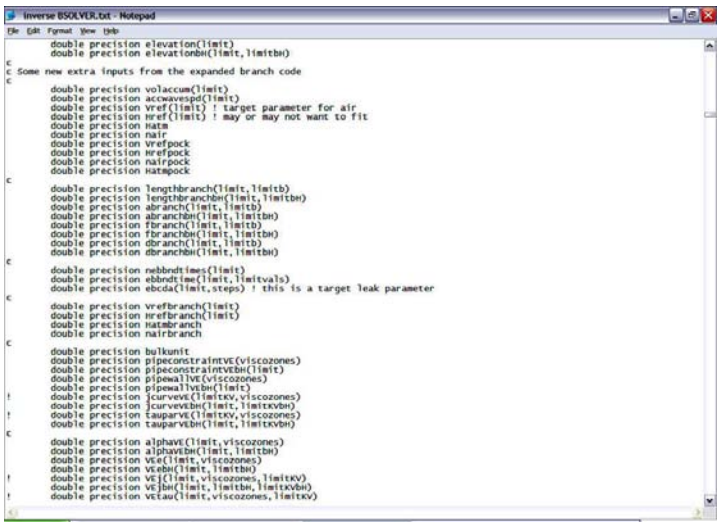

```
c++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++72
c
      subroutine model (timeStep, iflag, iopt, prt, neq, npar, npx,
     &                  iex, nfor, qfit, dfit, par, mfit, ifit, ibeale)
c
      implicit none
c
      integer timeStep
      integer iflag
      integer iopt
      character*1 prt
      integer neq
      integer npar
      integer npx
      integer iex
      integer nfor
      double precision qfit(iex)
      double precision dfit(npx,iex)
      double precision par(npx)
      integer mfit(iex)
      integer ifit(npx)
      integer ibeale
c
      integer i
      integer j
        integer inew
c
        integer maxObs
        integer limit
        integer limitb
        integer limitbH
        integer limitvals
        integer limitKV
        integer limitKVbH
        integer steps
c
        integer viscozones
c
        parameter (maxObs=165)
        parameter (limit=152)
        parameter (limitb=1)  ! vertical branch
        parameter (limitbH=1) ! horizontal branch
        parameter (limitvals=10)
        parameter (limitKV=1)
        parameter (limitKVbH=1)
        parameter (steps=200)
c
        parameter (viscozones=1)
c
        integer MAXiex
        integer MAXpar
```



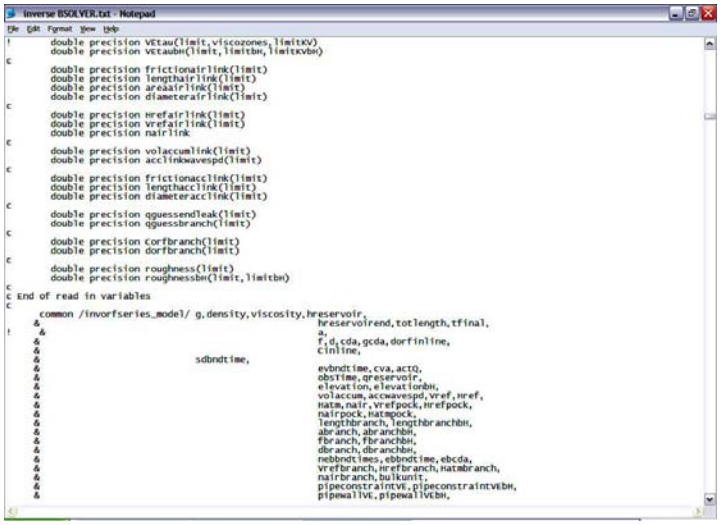

```
        integer MAXpar
        parameter (MAXiex=3)
        parameter (MAXpar=1)
c
c Must declare target parameter(s) here
c
!       double precision cda(limit,steps)
c
        double precision jcurveVE(limitKV,viscozones)
!!      double precision jcurveVEbH(limit,limitKVbH)
        double precision tauparVE(limitKV,viscozones)
!!      double precision tauparVEbH(limit,limitKVbH)
c
        double precision VEj(limit,viscozones,limitKV)
!!      double precision VEjbH(limit,limitbH,limitKVbH)
        double precision VEtau(limit,viscozones,limitKV)
!!      double precision VEtaubH(limit,limitbH,limitKVbH)
c
!       double precision dorfinline(limit)
        double precision a(limit)
c
c Declarations for model to pass predictions
c
      double precision hpred(0:maxObs,iex)
      double precision hfit(maxObs,iex)
c
c     Labelled common used to pass configuration data and measured
c     pressure array to subroutine model
c
        integer nreaches
        integer rspnode1
        integer rspnode2
        integer rspnode3
        integer sdorfflag(limit)
        integer ilorfflag(limit)
        integer lastObs(MAXiex)
        integer nsdbndtimes(limit)
        integer nevbndtimes
        integer endflag
c
        integer usfricflag
        integer accumflag(limit)
        integer airchamflag(limit)
        integer overalldispockflag
        integer dispockflag(limit)
c
        integer branchflag(limit)
        integer horzbflag(limit)
        integer branchendleak(limit)
        integer branchendair(limit)
c
```



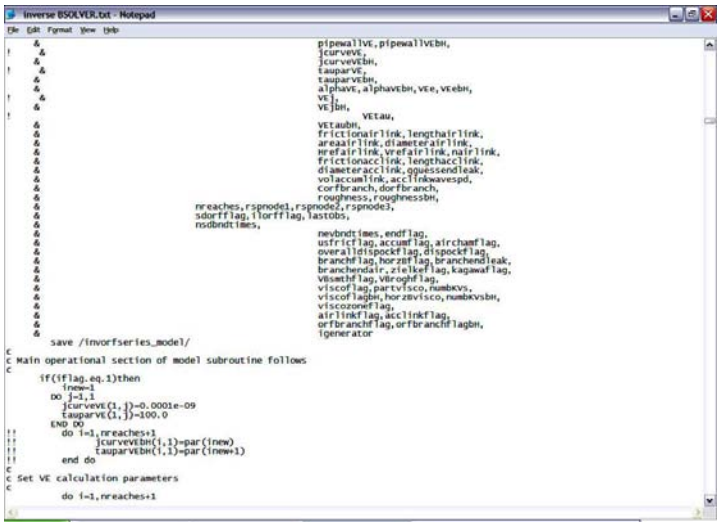

```
c
        integer zielkeflag
        integer kaqawaflag
        integer VBsmthflag
        integer VBroghflag
c
        integer viscoflag
        integer partvisco(limit)
        integer numbKVs(viscozones)
        integer viscoflagbH
        integer horzBvisco(limit)
        integer numbKVsbH(limit)
c
        integer viscozoneflag(limit)
c
        integer airlinkflag(limit)
        integer acclinkflag(limit)
c
        integer orfbranchflag(limit)
        integer orfbranchflagbH(limit,limitbH)
c
        integer igenerator(limit)
c
        integer iextracount
c
      double precision g
      double precision density
        double precision viscosity
      double precision hreservoir
        double precision hreservoirend
      double precision totlength
        double precision tfinal
!       double precision a(limit) !TARGET
        double precision f(limit)
        double precision d(limit)
!       double precision fricfrac  ! ?check - used for constant f along pipe
!       double precision diameter  ! ?check - used for constant d along pipe
c
        double precision cda(limit,steps) ! target parameter for leaks
        double precision gcda(limit,steps) ! declaration for leak as a generator
        double precision dorfinline(limit)! target parameter for discrete block
        double precision cinline(limit)   ! may or may not want to fit
        double precision sdbndtime(limit,limitvals) ! set as constant for the leak
        double precision evbndtime(limitvals)
        double precision cva(steps)
c
      double precision act0(maxObs,MAXiex)
      double precision obstime(0:maxObs,MAXiex)
c
        double precision qreservoir
        double precision elevation(limit)
```



```
inverse BSOLVER.txt - Notepad
File  Edit  Format  View  Help
      double precision elevation(limit)
      double precision elevationbh(limit,limitbh)
c
c Some new extra inputs from the expanded branch code
c
      double precision volaccum(limit)
      double precision accwavespd(limit)
      double precision vref(limit) ! target parameter for air
      double precision nref(limit) ! may or may not want to fit
      double precision natm
      double precision nair
      double precision vrefpock
      double precision nrefpock
      double precision nairpock
      double precision natmpock
c
      double precision lengthbranch(limit,limitb)
      double precision lengthbranchbh(limit,limitbh)
      double precision abranch(limit,limitb)
      double precision abranchbh(limit,limitbh)
      double precision fbranch(limit,limitb)
      double precision fbranchbh(limit,limitbh)
      double precision dbranch(limit,limitb)
      double precision dbranchbh(limit,limitbh)
c
      double precision nebbndtimes(limit)
      double precision ebbndtime(limit,limitvals)
      double precision ebcda(limit,steps) ! this is a target leak parameter
c
      double precision vrefbranch(limit)
      double precision nrefbranch(limit)
      double precision natmbranch
      double precision nairbranch
c
      double precision bulkunit
      double precision pipeconstraintVE(viscozones)
      double precision pipeconstraintVEbh(limit)
      double precision pipewallVE(viscozones)
      double precision pipewallVEbh(limit)
!     double precision jcurveVE(limitKV,viscozones)
      double precision jcurveVEbh(limit,limitKVbh)
!     double precision tauparVE(limitKV,viscozones)
      double precision tauparVEbh(limit,limitKVbh)
c
      double precision alphaVE(limit,viscozones)
      double precision alphaVEbh(limit,limitbh)
      double precision VEe(limit,viscozones)
      double precision VEebh(limit,limitbh)
!     double precision VEj(limit,viscozones,limitKV)
      double precision VEjbh(limit,limitbh,limitKVbh)
!     double precision VEtau(limit,viscozones,limitKV)
```

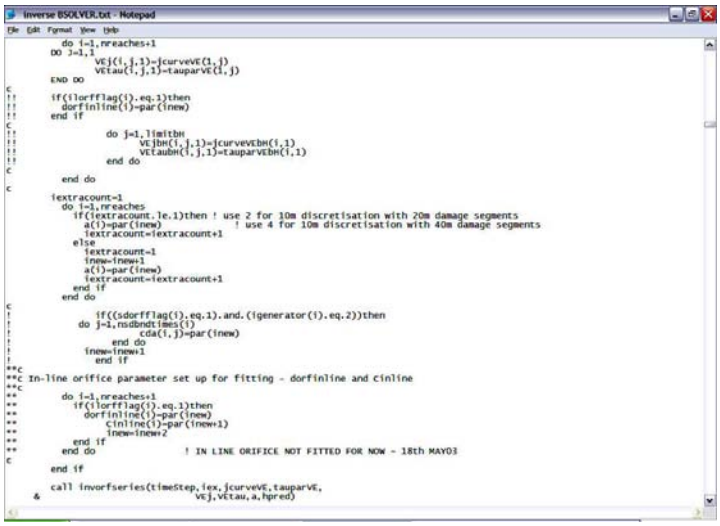


```
inverse BSOLVER.txt - Notepad
File  Edit  Format  View  Help
!     double precision VEtau(limit,viscozones,limitKV)
      double precision VEtaubh(limit,limitbh,limitKVbh)
c
      double precision frictionairlink(limit)
      double precision lengthairlink(limit)
      double precision areaairlink(limit)
      double precision diameterairlink(limit)
c
      double precision nrefairlink(limit)
      double precision vrefairlink(limit)
      double precision nairlink
c
      double precision volaccumlink(limit)
      double precision acclinkwavespd(limit)
c
      double precision frictionacclink(limit)
      double precision lengthacclink(limit)
      double precision diameteracclink(limit)
c
      double precision qguessendleak(limit)
      double precision qguessbranch(limit)
c
      double precision corfbranch(limit)
      double precision dorfbranch(limit)
c
      double precision roughness(limit)
      double precision roughnessbh(limit,limitbh)
c
c End of read in variables
c
      common /invorfseries_model/ g,density,viscosity,hreservoir,
     &                            hreservoirend,totlength,tfinal,
!    &                            a,
     &                            f,d,cda,gcda,dorfinline,
     &                            cinline,
     &                  sdbndtime,
     &                            evbndtime,cva,actq,
     &                            obstime,qreservoir,
     &                            elevation,elevationbh,
     &                            volaccum,accwavespd,vref,nref,
     &                            natm,nair,vrefpock,nrefpock,
     &                            nairpock,natmpock,
     &                            lengthbranch,lengthbranchbh,
     &                            abranch,abranchbh,
     &                            fbranch,fbranchbh,
     &                            dbranch,dbranchbh,
     &                            nebbndtimes,ebbndtime,ebcda,
     &                            vrefbranch,nrefbranch,natmbranch,
     &                            nairbranch,bulkunit,
     &                            pipeconstraintVE,pipeconstraintVEbh,
     &                            pipewallVE,pipewallVEbh,
```

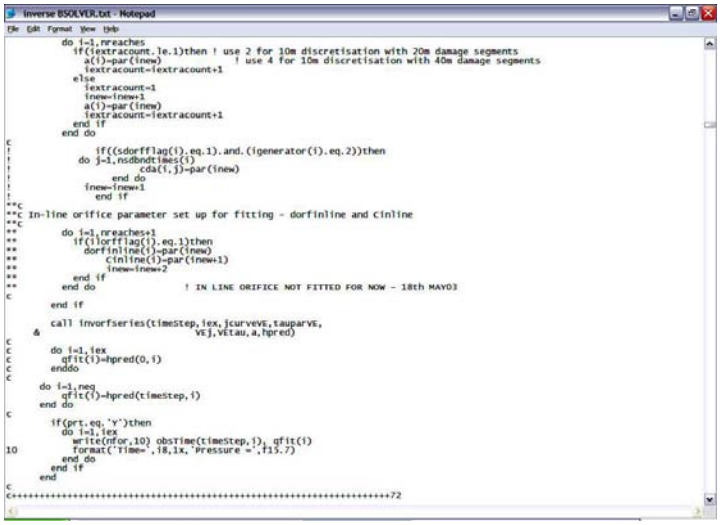


```
inverse BSOLVER.txt - Notepad
File  Edit  Format  View  Help
     &                            pipewallVE,pipewallVEbh,
!    &                            jcurveVE,
     &                            jcurveVEbh,
!    &                            tauparVE,
     &                            tauparVEbh,
     &                            alphaVE,alphaVEbh,VEe,VEebh,
!    &                            VEj,
     &                            VEjbh,
!    &                                  VEtau,
     &                            VEtaubh,
     &                            frictionairlink,lengthairlink,
     &                            areaairlink,diameterairlink,
     &                            nrefairlink,vrefairlink,nairlink,
     &                            frictionacclink,lengthacclink,
     &                            diameteracclink,qguessendleak,
     &                            volaccumlink,acclinkwavespd,
     &                            corfbranch,dorfbranch,
     &                            roughness,roughnessbh,
     &                  nreaches,rspnode1,rspnode2,rspnode3,
     &                  sdorfflag,ilorfflag,lastobs,
     &                  nsdbndtimes,
     &                            nevbndtimes,endflag,
     &                            usfricflag,accumflag,airchamflag,
     &                            overalldispockflag,dispockflag,
     &                            branchflag,horzbflag,branchendleak,
     &                            branchendair,zielkeflag,kagawaflag,
     &                            vbsmthflag,vbroghflag,
     &                            viscoflag,partvisco,numbkvs,
     &                            viscoflagbh,horzbvisco,numbkvsbh,
     &                            viscozoneflag,
     &                            airlinkflag,acclinkflag,
     &                            orfbranchflag,orfbranchflagbh,
     &                            igenerator
      save /invorfseries_model/
c
c Main operational section of model subroutine follows
c
      if(iflag.eq.1)then
        inew=1
        DO j=1,1
          jcurveVE(1,j)=0.0001e-09
          tauparVE(1,j)=100.0
        END DO
!!        do i=1,nreaches+1
!!              jcurveVEbh(i,1)=par(inew)
!!              tauparVEbh(i,1)=par(inew+1)
!!        end do
c
c Set VE calculation parameters
c
          do i=1,nreaches+1
```

Appendix M – Fortran code for BSOLVER and NLFIT subroutines



```
inverse BSOLVER.txt - Notepad
File  Edit  Format  View  Help
          do i=1,nreaches+1
        DO j=1,1
                VEj(i,j,1)=jcurveVE(i,j)
                VEtau(i,j,1)=tauparVE(i,j)
        END DO
c
        if(ilorfflag(i).eq.1)then
          dorfinline(i)=par(inew)
        end if
c
!!                do j=1,limitbH
!!                    VEjbH(i,j,1)=jcurveVEbH(i,1)
!!                    VEtaubH(i,j,1)=tauparVEbH(i,1)
!!                end do
c
          end do
c
        iextracount=1
          do i=1,nreaches
            if(iextracount.le.1)then ! use 2 for 10m discretisation with 20m damage segments
              a(i)=par(inew)                ! use 4 for 10m discretisation with 40m damage segments
              iextracount=iextracount+1
            else
              iextracount=1
              inew=inew+1
              a(i)=par(inew)
              iextracount=iextracount+1
            end if
          end do
c
!               if((sdorfflag(i).eq.1).and.(igenerator(i).eq.2))then
!           do j=1,nsdbndtimes(i)
!                   cda(i,j)=par(inew)
!               end do
!            inew=inew+1
!              end if
**c
**c In-line orifice parameter set up for fitting - dorfinline and cinline
**c
**        do i=1,nreaches+1
**          if(ilorfflag(i).eq.1)then
**            dorfinline(i)=par(inew)
**              cinline(i)=par(inew+1)
**                inew=inew+2
**          end if
**        end do                  ! IN LINE ORIFICE NOT FITTED FOR NOW - 18th MAY03
c
        end if

        call invorfseries(timeStep,iex,jcurveVE,tauparVE,
     &                            VEj,VEtau,a,hpred)
```



```
inverse BSOLVER.txt - Notepad
File  Edit  Format  View  Help
          do i=1,nreaches
            if(iextracount.le.1)then ! use 2 for 10m discretisation with 20m damage segments
              a(i)=par(inew)                ! use 4 for 10m discretisation with 40m damage segments
              iextracount=iextracount+1
            else
              iextracount=1
              inew=inew+1
              a(i)=par(inew)
              iextracount=iextracount+1
            end if
          end do
c
!               if((sdorfflag(i).eq.1).and.(igenerator(i).eq.2))then
!           do j=1,nsdbndtimes(i)
!                   cda(i,j)=par(inew)
!               end do
!            inew=inew+1
!              end if
**c
**c In-line orifice parameter set up for fitting - dorfinline and cinline
**c
**        do i=1,nreaches+1
**          if(ilorfflag(i).eq.1)then
**            dorfinline(i)=par(inew)
**              cinline(i)=par(inew+1)
**                inew=inew+2
**          end if
**        end do                  ! IN LINE ORIFICE NOT FITTED FOR NOW - 18th MAY03
c
        end if

        call invorfseries(timeStep,iex,jcurveVE,tauparVE,
     &                            VEj,VEtau,a,hpred)
c
c       do i=1,iex
c         qfit(i)=hpred(0,i)
c       enddo
c
      do i=1,neq
         qfit(i)=hpred(timeStep,i)
      end do
c
        if(prt.eq.'Y')then
          do i=1,iex
            write(ntor,10) obstime(timeStep,i), qfit(i)
10          format('Time=',i8,1x,'Pressure =',f15.7)
          end do
        end if
      end
c
c++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++72
```