3.9 OTHER CONCLUSIONS AND RECOMMENDATIONS

3.9.1 Opinions and Comments

In general, and for each liquefaction technology, most panel members expressed the opinion that the processes that will eventually be commercialized are not those currently under development. In the words of one panel member, "we haven't yet invented the chemistry that will ultimately be commercialized." This thinking is reflected in the highest-priority recommendations, many of which are for fundamental or applied research that will lead to new processes.

On the other hand, the panel recognized that DOE must continue to develop the current processes as a major short-term objective. Process development units (PDUs), although expensive to operate, are necessary to perform this function as proof-of-concept units (POCs). They also serve to test equipment and instrumentation, and provide information for economic evaluations. Most important, these large units are necessary to maintain preparedness for commercialization. Thus, the highest-priority recommendations are relatively balanced between fundamental research and process development/applied research.

Much of the recommended fundamental research is applicable to more than one technology. A prime example is research on coal structure, which applies equally to direct liquefaction, pyrolysis, coprocessing, and bioliquefaction. The list of recommendations contains several other research areas that cut across technology boundaries. Nevertheless, funding for these investigations is included in the budget for a particular technology, and understanding this, the panel has placed each fundamental research recommendation into one of the technologies to which it applies. The reader should understand that many of these fundamental programs should not be limited to the technology category in which they have been placed. Also, fundamental programs may be divided into two areas: supportive (or evolutionary) research, which provides basic

information for processes under development, and explorative (or revolutionary) research, which lays the foundation for new and better processes.

Opinions expressed by individual panel members at the final meeting in McLean, Virginia, on July 13-14, 1988, that disagreed with the consensus of the panel are as follows:

- The area of alternative liquefaction chemistries is not adequately represented in this document because these chemistries are not in the current processing schemes and research on them is not being funded by DOE.
- The DOE liquefaction program lacks the means to test new basic findings at the next scale of development. Without such a capability, research in fundamental areas is dead-ended and will be unable to modify significantly the current processes under development.
- Commercialization of pyrolysis depends on utilization of the char. It is a mistake to fund pyrolysis under liquefaction because the liquefaction community will not address the char utilization problem.
- Direct conversion of methane should be included among the high-priority recommendations because of its potential impact on fuels production.

3.9.2 DOE Procedures and Policies

The panel offered and received recommendations that relate to DOE procedures and policies. These recommendations, if implemented, could have a profound effect on the liquefaction program.

A major concern is that the DOE solicitation procedure channels research and thereby stifles new ideas. This results from the RFP procedure that request research in areas specified by DOE. New ideas in liquefaction, or which can be applied to liquefaction but which are not anticipated by the RFP, would be considered as not responsive. Instead, many of the proposals are for "me too" research in the belief that such work will be funded.

Therefore, the panel recommended that DOE increase funding for unsolicited proposals to encourage and fund research on new liquefaction chemistries.

Other recommendations concerning DOE procedures are:

- DOE should establish standardized procedures to evaluate new chemistries and process concepts in order to weed out programs of limited potential and make the best use of the available funds in programs of high potential.
- Establish a standardized procedure for communication or linking of basic and applied/process development research so that each understands the significant findings or needs of the other. This will speed the application of basic research to process development.
- One panel member recommended that universities should participate in each large development program, such as at LaPorte or Wilsonville. The universities could provide support in such areas as microautoclave tests, catalyst screening, and analyses. In this way, the university becomes familiar with the development program, and graduate students get first-hand experience in industrial programs.

The Wilsonville PDU is the largest-scale unit in operation in the direct liquefaction program. It is the unit that tests the best process(es) under development, and it does so in continuous-flow, integrated, steady-state runs of several weeks duration, and at conditions most closely approximating commercial operation. A concern was voiced about the lack of accessibility of many contractors to fundamental information and materials from this program. There was also an underlying feeling that more information is obtainable than is now being generated at Wilsonville. Consequently, the following recommendations were made to more closely coordinate the laboratory programs with the Wilsonville operation:

- Establish a Wilsonville data and sample bank from which DOE contractors can obtain materials produced at PDU scale and at well-documented conditions. The quantities produced at Wilsonville are sufficiently large to supply several contractors with the identical materials, if necessary,

eliminating many of the problems associated with interpreting results from laboratories that use different solvents, many of which are not representative of streams produced during liquefaction.

- This data/sample bank should be administered by an organization other than the Wilsonville operations, so that its sole responsibility will be the collecting of samples, documenting the run conditions that produced the samples, disseminating information regarding the samples available, and responding to requests for samples.

- More extensive analyses should be performed on the Wilsonville streams (product and internal) to obtain more fundamental understanding of what is happening in the direct liquefaction process. These analyses should be performed by a group with strong organic chemistry expertise.

3.10 PEER REVIEWERS' COMMENTS

A draft of this report was sent to ten outside reviewers who were chosen for their extensive experience in coal-liquefaction research and development. The following people served as peer reviewers of this report:

1. Mr. Seymour Alpert, Electric Power Research institute
2. Dr. Raymond Anderson, National Institute for Petroleum and Energy Research
3. Dr. David Gray, MITRE Corporation
4. Dr. Gerald Huffman, University of Kentucky
5. Dr. Alex Mills, University of Delaware
6. Mr. Eric Reichl, Consultant
7. Dr. George Roberts, Air Products and Chemicals
8. Dr. David Schmalzer, Argonne National Laboratory
9. Dr. Howard Stephens, Sandia National Laboratory
10. Dr. Duayne Whitehurst, Mobil Oil

In most instances peer reviewers' comments were incorporated within this report, particularly those which dealt with corrections or specific changes. Opposing and supporting viewpoints, and comments of a general nature are included in Appendix F. Numerous comments were received from the reviewers that this report provides a comprehensive and authoritative review of the status of coal liquefaction science and technology. The report was also considered to be generally well written by knowledgeable individuals. The choice of panel members was considered excellent and ensured that the total status of coal liquefaction technologies would be exhaustively reviewed. The reviewers also thought that the major advances of recent years are described in a clear manner, including the reasoning underlying individual developments. By and large, the reviewers expressed their agreement with the panel's recommendations and priorities.

The most important opposing comments are abstracted and summarized in the next sections, with replies by the principal investigator. Some of these opposing views concern statements made in the review chapters 4-9.

3.10.1 General (Opposing) Comments

a. There are fundamental problems in the DOE program that virtually assure the failure of basic research finding their way into process development. These problems include the absence of adequate resources at the process development and demonstration level. The panel, therefore, over-emphasized the need for and value of basic research given the inadequate resources provided for meaningful utilization of the products of basic research.

b. Process development and large pilot plant activities must receive greater resources than DOE has provided in recent years if there is to be any substantial likelihood of commercially deployable liquefaction technology.

c. DOE should have a few continuous-flow units in operation for process screening and process parameter studies. These units should range in scale from 0.5 to 2 tons/day to a fully integrated pilot plant of 100 to 200 tons/day that will be of a commercial process configuration.

   PI Reply: The need for screening units and scale-up facilities has been recognized by DOE. Design and construction of two PDUs will begin in 1989. A direct liquefaction PDU will have a capacity of 200 pounds of coal per day. The indirect liquefaction PDU will have a capacity of one barrel per day. This latter unit will have two independent reactor systems--one Fischer-Tropsch and the other to produce oxygenates.

d. CO/H_2 are produced more economically from natural gas, leading industry toward a concentrated R&D effort in CO/H_2 conversion. There is no need for DOE to interfere with these private-sector efforts.

e. Generally, R&D needs to be more "exploratory" and less "programmatic".

f. More importance should be given to innovative research.

g. Not enough consideration was given to measurement and control instrumentation.

3.10.2 Comments re: Direct Liquefaction

a. The research recommendations do not address the need for the data required for engineering design and process scale-up. In particular there is a lack of thermodynamic data. The panel placed too much emphasis on kinetic data.

b. Even if the (current) catalytic reactors were eliminated, the cost of product would not drop very much. To make further advances implies the discovery of some new approach which might allow operation in the 250-500 psig range.

c. An additional need for the operation of an integrated pilot plant is the development of meaningful environmental, safety, and health information.

d. The statement on Page 4-50, paragraph 3, asserts that the U.S. liquefaction processes use high surface-area supported catalysts operating at lower pressure than European (e.g., German) developers. This is inconsistent with the fact that processes developed in the 1970's included SRC-II, which used native coal minerals as catalyst and SRC-I, which was a thermal process.

PI reply: The statement referred to recent process developments. In the 1980's all liquefaction processes under development in the U.S. have used commercially-produced promoted hydrotreating catalysts.

e. A clarification of the liquid yields claimed in Tables 4-9 and 4-11 is needed. Does this include coal feed to the gasifier to generate hydrogen.

PI reply: These yields reflect the current processing philosophy of squeezing maximum liquid production out of the feed to the liquefaction process. These yields do not include the feed to the gasifier. Hydrogen can be produced by one of several methods, including gasification of coal.

f. An environmental issue not addressed is the restrictions on aromatic content in gasoline, such as the 0.8 percent benzene limit in California.

g. The analytical section of Chapter 4 should contain references to XAFS studies, variable angle spinning (VASS) and depolar dephasing. Low-temperature ashing, followed by x-ray diffraction or FTIR, is not a good way to study mineral matter.

h. More work is needed on the relationship of catalyst structure to catalyst performance.

i. Homogeneous catalysts used in the past, such as $ZnCl_2$, had great activity and deserve renewed attention.

j. Section 4.2.2, although informative, should not be used to present a comprehensive view of research in the chemistry and the mechanism of liquefaction reactions.

3.10.3 Comments re: Indirect Liquefaction

a. The review is narrowly limited to the conversion of CO/H_2, an area that is highly developed and well covered by R&D in private industry. The recommendations will do little to improve the economics of indirect coal liquefaction. The two major reasons are that coal gasification represents 4/5 of the total cost and that the CO/H_2 reaction is so efficient that further improvements will be irrelevant. The important subjects in indirect liquefaction are gasification and gas clean-up.

b. A more extensive evaluation of indirect liquefaction based on sulfur-resistant catalysts for the CO/H_2 conversion step should be recommended.

c. A somewhat more uncertain reduction in the cost of synthesis gas might be found in the use of air in lieu of oxygen.

d. Significant reduction in the cost of indirect liquefaction requires lower-cost synthesis gas, which probably means higher sulfur-content and, possibly, air-blown gas.

e. Given the excellent performance, long life, and low cost of methanol catalysts, there is little economic incentive for developing homogeneous liquid-phase catalysts.

f. The report should have placed greater emphasis on oxygenates, because of the surge in their use in transportation fuels.

g. Chapter 9 does not review overseas developments in indirect liquefaction.

3.10.4 Comments re: Pyrolysis

a. Major recommendations should be "What can be done with the tar and char" and "How is the reactor to be scaled up to get this same yield as obtained in small-scale equipment."

b. The swelling or "caking" tendency of coal is increased enormously by hydrogen, which simply fuses coal when in a dry state. Therefore, one can not be optimistic about the potential of hydropyrolysis.

c. Catalytic hydropyrolysis may simply be a new buzzword and should really be treated as part of the wider subject of innovative catalysts for hydrogenation at lower temperature.

```
            ymid = yp(pp) + d3y
            call velfind(up, vp, cell, xmid, ymid, vx, vy, dx, dy, nx,&
                         ny, nmax)
            d4x = dt * up
            d4y = dt * vp
            dxp = (d1x + d2x + d2x + d3x + d3x + d4x) / 6.0d+00
            dyp = (d1y + d2y + d2y + d3y + d3y + d4y) / 6.0d+00
         else
            dxp = zero
            dyp = zero
         end if
         xp(pp) = xp(pp) + dxp
         yp(pp) = yp(pp) + dyp
         upp(pp) = dxp / dt
         vpp(pp) = dyp / dt
      end do
   end if

   ! Calculate the number particles and the age of the grout in each
   ! model cell and update the yield stress array.
   yb1 = sum(dy)
   do i = 1, ny
      yb2 = yb1
      yb1 = yb2 - dy(i)
      xb2 = zero
      do j = 1, nx
         xb1 = xb2
         xb2 = xb1 + dx(j)
         if (cell(i,j) /= 0 .and. h(i,j) > delta) then
            pc(i,j) = 0
            if (np > 0) then
               tsum = zero
               do pp = 1, np
                  flag = (xp(pp) > xb1 .and. xp(pp) <= xb2)
                  flag = flag .and. (yp(pp) > yb1)
                  flag = flag .and. (yp(pp) <= yb2)
                  if (flag) then
                     pc(i,j) = pc(i,j) + 1
                     tsum = tsum + tp(pp)
                  end if
               end do
            end if
            if (pc(i,j) > 0) then
               age(i,j) = time - tsum / dble(pc(i,j))
            else
               age(i,j) = zero
            end if
         else
            age(i,j) = zero
            pc(i,j) = 0
         end if
         yldstr(i,j) = yield + stiffrate * age(i,j)
         yldstr(i,j) = dmin1(yldstr(i,j), ultimt)
      end do
   end do

   ! Determine if the rooms have been completely filled.
```

```
fillflag = .true.
do ksweep = 1, active
   i = isweep(ksweep)
   j = jsweep(ksweep)
   fillflag = fillflag .and. (h(i,j) >= ceil(i,j))
   if (.not. fillflag) exit
end do
if (fillflag) then
   write (out,*)
   write (out,*) 'Mine has been completely filled with grout.'
   write (out,*)
end if

! Write the restart file.
restart = time - trestart > trst - dtmin * 1.0d-01
restart = restart .or. time > tmax - dtmin * 1.0d-01
restart = restart .or. fillflag .or. divergent
if (restart) then
   trestart = time
   call hms(tstring, time)
   open (unit=rst, file=trim(jobnam)//'.rst', status='unknown')
   write (rst,cfrmt) '! Restart file for job: ' // trim(jobnam)
   write (rst,cfrmt) '! Simulation time: ' // trim(tstring)
   write (rst,cfrmt) '! Note:'
   write (rst,ifrmt) '! nmax =  ', nmax
   write (rst,ifrmt) '! ncl = ', ncl
   write (rst,ifrmt) '! nqn =   ', nqn
   write (rst,ifrmt) '! npp =   ', npp
   write (rst,*)
   write (rst,cfrmt) '&fdgrid'
   write (rst,ifrmt) 'nx =   ', nx
   write (rst,ifrmt) 'ny =   ', ny
   if (uniform) then
      write (rst,cfrmt) 'uniform = .true.'
      write (rst,rfrmt) 'dx =   ', dx(1)
      write (rst,rfrmt) 'dy =   ', dy(1)
   else
      write (rst,cfrmt) 'uniform = .false.'
      write (rst,rowfrmt) 'dx =   ', (dx(j), j = 1, nx)
      write (rst,rowfrmt) 'dy =   ', (dy(i), i = 1, ny)
   end if
   if (cellopen) then
      write (rst,cfrmt) 'cellopen = .true.'
      write (rst,cfrmt) 'checker = .false.'
   else if (checker) then
      write (rst,cfrmt) 'checker = .true.'
      write (rst,cfrmt) 'cellopen = .false.'
   else
      write (rst,cfrmt) 'byrw = .false.'
      write (rst,cellfrmt) 'cell = ', cell
   end if
   write (rst,cfrmt) '/'
   write (rst,*)
   write (rst,cfrmt) '&simtime'
   write (rst,rfrmt) 'time = ', time
   write (rst,rfrmt) 'tmax = ', tmax
   write (rst,rfrmt) 'dtmax =   ', dtmax
```

```
write (rst,rfrmt) 'dtmin =   ', dtmin
write (rst,rfrmt) 'dtfact =  ', dtfact
write (rst,rfrmt) 'omega =   ', omega
write (rst,rfrmt) 'work = ', work
write (rst,rfrmt) 'rest = ', rest
write (rst,cfrmt) '/'
write (rst,*)
write (rst,cfrmt) '&converg'
write (rst,rfrmt) 'hcvmx =   ', hcvmx
write (rst,rfrmt) 'vermx =   ', vermx
write (rst,rfrmt) 'cnl =  ', cnl
write (rst,ifrmt) 'itermax = ', itermax
write (rst,ifrmt) 'itermin = ', itermin
write (rst,cfrmt) '/'
write (rst,*)
write (rst,cfrmt) '&output'
write (rst,rfrmt) 'trst = ', trst
write (rst,rfrmt) 'tbin = ', tbin
write (rst,rfrmt) 'trep = ', trep
if (clr) then
   write (rst,cfrmt) 'clr = .true.'
else
  write (rst,cfrmt) 'clr = .false.'
end if
write (rst,rfrmt) 'tlog =   ', tlog
write (rst,rowfrmt) 'cl = ', cl
if (vflag) then
   write (rst,cfrmt) 'vflag = .true.'
else
   write (rst,cfrmt) 'vflag = .false.'
end if
if (cnflag) then
   write (rst,cfrmt) 'cnflag = .true.'
else
   write (rst,cfrmt) 'cnflag = .false.'
end if
if (pflag) then
   write (rst,cfrmt) 'pflag = .true.'
else
   write (rst,cfrmt) 'pflag = .false.'
end if
write (rst,cfrmt) '/'
write (rst,*)
write (rst,cfrmt) '&material'
write (rst,rfrmt) 'rho =  ', rho
write (rst,rfrmt) 'g =    ', g
write (rst,rfrmt) 'yield =   ', yield
write (rst,rfrmt) 'r =    ', r
write (rst,ifrmt) 'cr =   ', cr
write (rst,rfrmt) 'tharden = ', tharden
write (rst,rfrmt) 'ultimt =  ', ultimt
write (rst,cfrmt) '/'
write (rst,*)
write (rst,cfrmt) '&geometric'
write (rst,rfrmt) 'hmax = ', hmax
write (rst,rfrmt) 'delta =   ', delta
write (rst,rfrmt) 'eps =  ', eps
```

```
      write (rst,ifrmt) 'slot = ', slot
      write (rst,rfrmt) 'hflg =   ', hflg
      write (rst,browfrmt) 'bott =  ', bott
      write (rst,browfrmt) 'ceil = ', ceil
      write (rst,cfrmt) '/'
      write (rst,*)
      write (rst,cfrmt) '&initial'
      write (rst,rfrmt) 'hs =   ', hs
      write (rst,rowfrmt) 'h =  ', h
      write (rst,rowfrmt) 'vx = ', vx
      write (rst,rowfrmt) 'vy = ', vy
      write (rst,cellfrmt) 'rows =   ', (rows(iq), iq = 1, nq)
      write (rst,cellfrmt) 'cols =   ', (cols(iq), iq = 1, nq)
      write (rst,rowfrmt) 'tqn =  ', (tqn(iq), iq = 1, nq)
      write (rst,rowfrmt) 'qqn =  ', (qqn(iq), iq = 1, nq)
      write (rst,ifrmt) 'nq =   ', nq
      write (rst,cfrmt) '/'
      write (rst,*)
      write (rst,cfrmt) '&particles'
      write (rst,rfrmt) 'tintro =  ', tintro
      write (rst,ifrmt) 'np =   ', np
      write (rst,ifrmt) 'update =  ', update
      if (np > 0) then
         write (rst,rowfrmt) 'xp =   ', (xp(pp), pp = 1, np)
         write (rst,rowfrmt) 'yp =   ', (yp(pp), pp = 1, np)
         write (rst,rowfrmt) 'tp =   ', (tp(pp), pp = 1, np)
      end if
      write (rst,cfrmt) '/'
      close (rst)
   end if

   ! Write report output file.
   globvolerr = volume - totalsource - original
   globvolerr = globvolerr / (original + totalsource)
   numglobvolerr = numvolume - totalsource - original
   numglobvolerr = numglobvolerr / (original + totalsource)
   report = time - trepprev > trep - dtmin * 1.0d-01
   report = report .or. time > tmax - dtmin * 1.0d-01
   report = report .or. fillflag .or. divergent
   if (report) then
      trepprev = time

      ! Calculate the time string and write the numerical arrays.
      call hms(tstring, time)
      write (its,'(i10)') iterate
      its = adjustl(its)
      write (*,tprt)'time = ',trim(tstring),'iterate = ',trim(its),&
                'depth error = ', hconv, 'num vol error = ', numvolerr
      if (single) then
         dum1 = 'Total Depth at Time = ' // trim(tstring)
         call matprn(out, trim(dum1), h, cell, h, nx, ny, ceil, delta,&
                     nmax, 'f', 'd')
      else
         dum1 = 'Incremental Depth at Time = ' // trim(tstring)
         call matprn(out, trim(dum1), h, cell, h, nx, ny, ceil, delta,&
                     nmax, 'f', 'd')
         dum1 = 'Total Depth at Time = ' // trim(tstring)
```

```
   ht = h + bott - datum
   call matprn(out, trim(dum1),ht, cell, ht, nx, ny, top,delta, &
               nmax, 'f', 'd')
end if
flag = .false.
do i = 1, ny
   do j = 1, nx
      if (cell(i,j) == 0) then
         p(i,j) = hflg
      else
         phead = h(i,j) - ceil(i,j)
         if (phead > zero) then
            p(i,j) = phead * rho * g
            flag = .true.
         end if
      end if
   end do
end do
if (flag) then
   call matprn(out, 'Pressure at Time = ' // trim(tstring), p, &
               cell, h, nx, ny, ceil, delta, nmax, 'e', 'n')
end if
if (vflag) then
   call matprn(out, 'X-Velocity at Time = '//trim(tstring), vx, &
               cell, h, nx, ny, ceil, delta, nmax, 'e', 'x')
   call matprn(out, 'Y-Velocity at Time = '//trim(tstring), vy, &
               cell, h, nx, ny, ceil, delta, nmax, 'e', 'y')
end if
if (cnflag) then
   call matprn(out, 'X-Courant at Time = '//trim(tstring), cnx, &
               cell, h, nx, ny, ceil, delta, nmax, 'f', 'x')
   call matprn(out, 'Y-Courant at Time = '//trim(tstring), cny, &
               cell, h, nx, ny, ceil, delta, nmax, 'f', 'y')
end if

! Calculate and write the flow regimes for each model cell.
write (out,*)
write (out,*) 'Flow Regime at Time = ' // trim(tstring)
do i = 1, ny
   do j = 1, nx
      if (cell(i,j) == 0) then
         regime(j) = 'XXXXXXXXXX'
      else if (h(i,j) < delta) then
         regime(j) = '    '
      else
         ip1 = min(i + 1, ny)
         im1 = max(i - 1, 1)
         jp1 = min(j + 1, nx)
         jm1 = max(j - 1, 1)
         vbx = dabs(vx(i,jm1) + vx(i,j)) / two
         vby = dabs(vy(im1,j) + vy(i,j)) / two
         if (vbx > vby) then
            aj = area(h(i,j), dy(i), delta, ceil(i,j), eps, slot)
            if (i == 1) then
               rj = wp(h(i,j), h(i,j),ceil(i,j),ceil(i,j),dy(i),&
                       dx(j), dx(j), eps, int1(0), cell(ip1,j), &
                       int1(0), cell(ip1,j), slot)
```

```
            else if (i == ny) then
               rj = wp(h(i,j),h(i,j),ceil(i,j),ceil(i,j),dy(i), &
                       dx(j),dx(j),eps,cell(im1,j),int1(0),     &
                       cell(im1,j), int1(0), slot)
            else
               rj = wp(h(i,j),h(i,j),ceil(i,j),ceil(i,j),dy(i),&
                       dx(j),dx(j),eps,cell(im1,j),cell(ip1,j),&
                       cell(im1,j),cell(ip1,j),slot)
            end if
            rj = aj / rj
         else
            ai = area(h(i,j), dx(j), delta, ceil(i,j), eps, slot)
            if (j == 1) then
               ri = wp(h(i,j),h(i,j),ceil(i,j),ceil(i,j),dx(j),&
                       dy(i),dy(i),eps,int1(0),cell(i,jp1),    &
                       int1(0), cell(i,jp1), slot)
            else if (j == nx) then
               ri = wp(h(i,j),h(i,j),ceil(i,j),ceil(i,j),dx(j),&
                       dy(i),dy(i), eps, cell(i,jm1), int1(0), &
                       cell(i,jm1), int1(0), slot)
            else
               ri = wp(h(i,j),h(i,j),ceil(i,j),ceil(i,j),dx(j),&
                       dy(i),dy(i),eps,cell(i,jm1),cell(i,jp1),&
                       cell(i,jm1),cell(i,jp1),slot)
            end if
            ri = ai / ri
         end if
         if (vbx * rj > vby * ri) then
            vbar = vbx
            rh = rj
         else
            vbar = vby
            rh = ri
         end if
         re = 4.0d+00 * rh * vbar * rho / r
         he = 1.6d+01 * rh * rh * yield * rho / r
         call critical(rec, he)
         if (re == zero) then
            regime(j) = '  Static  '
         else if (re > rec) then
            regime(j) = '  Trblnt  '
         else
            regime(j) = '  Laminr  '
         end if
      end if
   end do
   write (out,echofmt(6)) (regime(j), j = 1, nx)
end do

! If there are any system particles, write the number of
! particles in each model cell and the grout age.
if (np > 0 .and. pflag) then
   write (out,*)
   dum1 = 'Particle Distribution at Time ='
   write (out,*) trim(dum1) // ' ' // trim(tstring)
   yb1 = sum(dy)
   do i = 1, ny
```

```
      yb2 = yb1
      yb1 = yb2 - dy(i)
      xb2 = zero
      do j = 1, nx
         xb1 = xb2
         xb2 = xb1 + dx(j)
         if (cell(i,j) == 0) then
            regime(j) = 'XXXXXXXXXX'
         else if (h(i,j) < delta) then
            regime(j) = '    '
         else
            regime(j) = '    '
            if (pc(i,j) > 0) then
               write (dum1,*) pc(i,j)
               dum1 = adjustl(dum1)
               cc = len_trim(dum1)
               ca = (10 - cc) / 2 + 1
               if (mod(cc,2) == 1) ca = ca + 1
               cb = ca + cc - 1
               regime(j)(ca:cb) = trim(dum1)
            end if
         end if
      end do
      write (out,echofmt(6)) (regime(j), j = 1, nx)
   end do
   dum1 = 'Grout Age at Time ='
   call matprn(out, trim(dum1)//' '//trim(tstring), age, cell, &
               h, nx, ny, ceil, delta, nmax, 'e', 'n')
end if
if (pflag) then
   call matprn(out, 'Yield Stress at Time = '//trim(tstring), &
           yldstr, cell, h, nx, ny, ceil, delta, nmax, 'e', 'n')
end if

! Write the volume balance figures.
write (dum1,*) rowinject
dum1 = adjustl(dum1)
write (dum2,*) colinject
dum2 = adjustl(dum2)
dum1 = repeat(' ',12 - len_trim(dum1) - len_trim(dum2)) // &
        '(' // trim(dum1) // ',' // trim(dum2) // ')'
write(out,*)
write(out,*)'Volume Balance at Time = ' // trim(tstring)
write(out,*)'Original Volume =              ', real(original)
write(out,*)'Functional Incr. Volume Error =', real(volerr)
write(out,*)'Functional Total Volume Error =', real(globvolerr)
write(out,*)'Functional Previous Volume =   ', real(previous)
write(out,*)'Functional Present Volume =    ', real(volume)
write(out,*)'Numerical Incr. Volume Error = ', real(numvolerr)
write(out,*)'Numerical Total Volume Error = ',real(numglobvolerr)
write(out,*)'Numerical Previous Volume =    ', real(numprevious)
write(out,*)'Numerical Present Volume =     ', real(numvolume)
write(out,*)'Incremental Net Injection =    ', real(source)
write(out,*)'Injection Model Cell =         ' // trim(dum1)
write(out,*)'Total Net Injection =          ', real(totalsource)
write(out,*)'Present Time Step =            ', real(dt)
write(out,*)
```

```
      if (np > 0) then
         write (dum1,'(i10)') np
         dum1 = 'Number of Particles = ' // trim(adjustl(dum1))
         dum1 = trim(dum1) // ' at Time = ' // trim(tstring)
         write (out,*) trim(dum1)
         write (out,*)
         dum1 = '  pp xp(pp) yp(pp)   upp(pp)   vpp(pp)'
         write (out,*) trim(dum1)
         do pp = 1, np
            write (out,ppfrmt) pp, xp(pp), yp(pp), upp(pp), vpp(pp)
         end do
         write (out,*)
      end if
      write (out,*)
   end if

   ! Write binary output file.
   binary = time - tbinprev > tbin - dtmin * 1.0d-01
   binary = binary .or. time > tmax - dtmin * 1.0d-01
   binary = binary .or. fillflag .or. divergent
   if (binary) then
      tbinprev = time
      write (bin) real(time), iterate
      write (bin) real(volerr), real(globvolerr), np
      ht = h + bott - datum
      write (bin)((real(dmin1(ht(i,j),top(i,j))),j=1,nx),i=1,ny)
      write (bin) ((real(vx(i,j)), j = 1, nx), i = 1, ny)
      write (bin) ((real(vy(i,j)), j = 1, nx), i = 1, ny)
      if (np > 0) then
         write (bin) (real(xp(pp)), pp = 1, np)
         write (bin) (real(yp(pp)), pp = 1, np)
         write (bin) (real(tp(pp)), pp = 1, np)
      end if
      write (bin) ((pc(i,j), j = 1, nx), i = 1, ny)
      write (bin) ((real(age(i,j)), j = 1, nx), i = 1, ny)
      write (bin) ((real(yldstr(i,j)), j = 1, nx), i = 1, ny)
   end if

   ! Store results of calculations.
   previous = volume
   numprevious = numvolume
   ho = h

   ! Exit loop, if goal has been reached.
   if (time > tmax - dtmin * 1.0d-01 .or. fillflag) exit

   ! Write message and exit loop, if calculation has diverged.
   if (divergent) then
      write (out,*) 'Calculations have diverged.'
      write (log,*) 'Calculations have diverged.'
      write (*,*) 'Calculations have diverged.'
      exit
   end if
end do

! Write the stopping and elapsed time.
call stoping(out, scsec)
```

```
! Close output files and end program.
close (out)
close (bin)
close (log)
stop
end

function area(h, b, delta, hmax, eps, slot)
! Area is designed to return the flow area.
! Jim Stiles, 08/09/1997.
implicit none
integer*4 slot
real*8 h, b, delta, area, hmax, eps
if (slot == 1) then
   if (h >= hmax) then
      area = hmax * b + eps * b * (h - hmax)
   else
      area = dmax1(h, delta) * b
   end if
else
   area = b * dmin1(dmax1(h, delta), hmax)
end if
return
end

subroutine js(q, sf, area, hyradius, rho, g, yield, r)
! Js is designed to calculate the discharge under the specified
! uniform conditions.  Js will automatically determine if the
! flow is turbulent or laminar.  Js uses equations derived from
! the Bingham fluid equations presented by Steffe's (1996) book on
! food rheology.  Jim Stiles, 09/12/1997.
implicit none
real*8 q, sf, area, hyradius, rho, g, yield, r, re, he, v, f, &
       recritical, gamma, hyradius2, small, fp, c, chezy
data small /1.0d-05/

! Calculate the flow using the laminar uniform flow equation.
gamma = rho * g
hyradius2 = hyradius * hyradius
v = 2.0d+00 * gamma * hyradius2 / (3.0d+00 * r)
v = v * (sf - yield / (gamma * hyradius))
q = v * area

! Calculate the Reynolds's and Hedstrom's numbers.
re = 4.0d+00 * hyradius * v * rho / r
he = 1.6d+01 * hyradius2 * yield * rho / r

! Calculate the critical Reynolds's numbers.
call critical(recritical, he)

! If the flow is laminar, then the calculated laminar flow is
! the correct discharge to return to the calling program.
if (re <= recritical) return

! Since the flow is turbulent, calculate the darcy
! friction factor using the laminar value of the friction factor
```

```
! as an initial guess.
f = 6.4d+01 * (6.0d+00 * re + he) / (6.0d+00 * re * re)
c = 1.0d+00 - yield / (gamma * hyradius * sf)
do
   fp = f
   f = 2.265d+00 * dlog10(c * re * dsqrt(fp)) - 2.982d+00
   f = 1.0d+00 / (f * f)
   chezy = dsqrt(8.0d+00 * g / f)
   v = chezy * dsqrt(hyradius * sf)
   re = 4.0d+00 * hyradius * v * rho / r
   if (dabs(f - fp) < small) exit
end do

! Calculate the discharge and return to calling program.
q = v * area
return
end

subroutine critical(recritical, he)
! Critical is designed to calculate the critical Re number for
! a Bingham fluid flow.  Jim Stiles, 09/21/1997.
real*8 recritical, he, c1, c2, small, cc, hem
data small /1.0d-05/
c1 = small
c2 = 1.0d+00 - small
do
   cc = 0.5d+00 * (c1 + c2)
   if (c2 - c1 < small) exit
   hem = 1.0d+00 - cc
   hem = hem * hem * hem
   hem = 1.68d+04 * cc / hem
   if (hem > he) then
      c2 = cc
   else
      c1 = cc
   end if
end do
recritical = -4.0d+00 * cc + cc * cc * cc * cc
recritical = 1.0d+00 + recritical / 3.0d+00
recritical = recritical * he / (8.0d+00 * cc)
return
end

subroutine starting(out, scsec)
! Starting is designed to write the present time to the output file.
! Jim Stiles, 12/12/1997.
implicit none
integer*1 out
integer*4 csec, first, last, scsec, value(8), hour, daysofmonth(12)
character mon*36, cdate*8, ctime*10, czone*5, dummy*24
logical flag, leap
data mon /'JANFEBMARAPRMAYJUNJULAUGSEPOCTNOVDEC'/
data daysofmonth /31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31/
call date_and_time(cdate, ctime, czone, value)
first = 3 * (value(2) - 1) + 1
last = first + 2
csec = int(real(value(8)) * 100.)
```

```
hour = value(5)
flag = (hour == 0 .and. value(6) == 0 .and. value(7) == 0)
flag = flag .and. (value(8) == 0)
if (flag) then
   hour = 24
   value(3) = value(3) - 1
   if (value(3) == 0) then
      value(2) = value(2) - 1
      if (value(2) == 0) then
         value(2) = 12
         value(1) = value(1) - 1
      end if
      leap = (mod(value(1),100) /= 0)
      leap = leap .or. (mod(value(1),400) /= 0)
      leap = leap .and. (mod(value(1),4) == 0)
      leap = leap .and. (value(2) == 2)
      value(3) = daysofmonth(value(2))
      if (leap) value(3) = value(3) + 1
   end if
end if
write (dummy(1:2),'(i2.2)') hour
dummy(3:3) = ':'
write (dummy(4:5),'(i2.2)') value(6)
dummy(6:6) = ':'
write (dummy(7:8),'(i2.2)') value(7)
dummy(9:9) = '.'
write (dummy(10:11),'(i2.2)') csec
dummy(12:13) = ', '
write (dummy(14:15),'(i2.2)') value(3)
dummy(16:20) = '-' // mon(first:last) // '-'
write (dummy(21:24),'(i4.4)') value(1)
write (out,*)
write (out,'(1x,a,t46,a24)') 'Simulation Began at:', dummy
scsec = value(5) * 360000 + value(6) * 6000 + value(7) * 100 + csec
return
end

subroutine stoping(out, scsec)
! Stoping is designed to write the present time and elapsed time to
! the output file.  Jim Stiles, 12/12/1997.
implicit none
integer*1 out
integer*4 csec,first,last,scsec,value(8),hour,daysofmonth(12),diff
real*4 elapsed
character mon*36,cdate*8,ctime*10,czone*5,dummy*24,tstring*20,et*14
logical flag, leap
data mon /'JANFEBMARAPRMAYJUNJULAUGSEPOCTNOVDEC'/
data daysofmonth /31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31/
call date_and_time(cdate, ctime, czone, value)
first = 3 * (value(2) - 1) + 1
last = first + 2
csec = int(real(value(8)) * 100.)
hour = value(5)
flag = (hour == 0 .and. value(6) == 0 .and. value(7) == 0)
flag = flag .and. (value(8) == 0)
if (flag) then
   hour = 24
```

```
      value(3) = value(3) - 1
      if (value(3) == 0) then
         value(2) = value(2) - 1
         if (value(2) == 0) then
            value(2) = 12
            value(1) = value(1) - 1
         end if
         leap = (mod(value(1),100) /= 0)
         leap = leap .or. (mod(value(1),400) /= 0)
         leap = leap .and. (mod(value(1),4) == 0)
         leap = leap .and. (value(2) == 2)
         value(3) = daysofmonth(value(2))
         if (leap) value(3) = value(3) + 1
      end if
   end if
   write (dummy(1:2),'(i2.2)') hour
   dummy(3:3) = ':'
   write (dummy(4:5),'(i2.2)') value(6)
   dummy(6:6) = ':'
   write (dummy(7:8),'(i2.2)') value(7)
   dummy(9:9) = '.'
   write (dummy(10:11),'(i2.2)') csec
   dummy(12:13) = ', '
   write (dummy(14:15),'(i2.2)') value(3)
   dummy(16:20) = '-' // mon(first:last) // '-'
   write (dummy(21:24),'(i4.4)') value(1)
   write (out,'(1x,a,t46,a24)') 'Simulation terminated at:', dummy
   diff = value(5)*360000 + value(6)*6000 + value(7)*100 + csec - scsec
   if (diff < 0) diff = diff + 8640000
   elapsed = real(diff) / 100.0
   call hms(tstring, dble(aint(elapsed)))
   write (dummy(1:2),'(i2.2)') diff - int4(elapsed) * 100
   et = trim(tstring) // '.' // dummy(1:2)
   et = adjustr(et)
   write (out,'(1x,a,t56,a14)') 'Elapsed simulation time:', et
   return
   end

   subroutine hms(tstring, time)
   ! Hms transforms the simulation time variable into the hHH:MM:SS
   ! string format.
   ! Jim Stiles, 09/29/1997.
   implicit none
   character tstring*20
   real*8 time, t
   integer*1 hours, minutes, seconds
   character form1*40
   data form1 /'(i3.2,'':'',i2.2,'':'',i2.2)   '/
   if (time >= 3.5964d+06) then
      tstring = '999:00:00'
   else
      t = dnint(time)
      hours = int1(t / 3.6d+03)
      t = t - dble(hours) * 3.6d+03
      minutes = int1(t / 6.0d+01)
      t = t - dble(minutes) * 6.0d+01
      seconds = int1(t)
```

```
   write (tstring,form1) hours, minutes, seconds
   tstring = adjustl(tstring)
end if
return
end

subroutine matprn(out, hstring, z, cell, h, nx, ny, ceil, hmin, nmax, &
                  form, stagger)
! Matprn is designed to write the active part of a specified
! matrix in a form where the reader can easily determine the
! manner in which the grout is moving in the mine.
! Jim Stiles, 11/20/1997.
implicit none
integer*4 nmax, nx, ny, i, j, lt, im1, jm1
integer*1 cell(nmax,nmax), out
real*8 z(nmax,nmax), h(nmax,nmax), ceil(nmax,nmax), hmin, zero, zz, two
character hstring*(*), field*10, total*1000, form*1, stagger*1
logical flag, eflag
data zero /0.0d+00/, two /2.0d+00/
write (out,*)
write (out,*) trim(hstring)
do i = 1, ny
   im1 = max(i - 1, 1)
   lt = 0
   do j = 1, nx
      jm1 = max(j - 1, 1)
      flag = h(i,j) > hmin .or. stagger == 'b' .or. stagger == 'B'
      if (cell(i,j) == 0) then
         field = 'XXXXXXXXXX'
      else if (flag) then
         if (stagger == 'x' .or. stagger == 'X') then
            zz = (z(i,jm1) + z(i,j)) / two
         else if (stagger == 'y' .or. stagger == 'Y') then
            zz = (z(im1,j) + z(i,j)) / two
         else
            zz = z(i,j)
         end if
         if (zz >= zero) then
            if (stagger == 'd' .or. stagger == 'D') then
               zz = dmin1(ceil(i,j), zz)
            end if
            eflag = zz > 9.9999d+00 .and. stagger /= 'b'
            eflag = eflag .and. stagger /= 'B'
            eflag = eflag .or. form == 'e' .or. form == 'E'
            eflag = eflag .or. zz > 9.9999d+02
            if (eflag) then
               write (field,'(1pe9.2,1x)') zz
            else
               if (zz >= 1.0d+00) then
                  if (stagger == 'b' .or. stagger == 'B') then
                     write (field,'(2x,f6.2,2x)') zz
                  else
                     write (field,'(2x,f6.4,2x)') zz
                  end if
               else
                  if (stagger == 'b' .or. stagger == 'B') then
                     write (field,'(3x,''0'',f3.2,3x)') zz
```

```
                  else
                     write (field,'(2x,''0'',f5.4,2x)') zz
                  end if
               end if
            end if
         else
            if (stagger == 'd' .or. stagger == 'D') then
               zz = dmin1(ceil(i,j), zz)
            end if
            eflag = zz < -9.999d+00 .and. stagger /= 'b'
            eflag = eflag .and. stagger /= 'B'
            eflag = eflag .or. form == 'e' .or. form == 'E'
            eflag = eflag .or. zz < -9.999d+01
            if (eflag) then
               write (field,'(1pe9.1,1x)') zz
            else
               if (zz <= -1.0d+00) then
                  if (stagger == 'b' .or. stagger == 'B') then
                     write (field,'(2x,f6.2,2x)') zz
                  else
                     write (field,'(2x,f6.3,2x)') zz
                  end if
               else
                  if (stagger == 'b' .or. stagger == 'B') then
                     write (field,'(3x,''-0'',f3.2,2x)') dabs(zz)
                  else
                     write (field,'(2x,''-0'',f4.3,2x)') dabs(zz)
                  end if
               end if
            end if
         end if
      else
         field = '     '
      end if
      total(lt+1:lt+10) = field
      lt = lt + 10
   end do
   write (out,'(1x,a)') total(1:lt)
end do
return
end

subroutine echoinjection(out, rows, cols, tqn, qqn, nq)
! Echoinjection is designed to echo the grout injection schedule.
! Jim Stiles, 03/22/1998.
implicit none
integer*1 out
integer*4 rows(1), cols(1), nq, iq
real*8 tqn(1), qqn(1)
character tstring*20, fmt1*20, fmt2*20
data fmt1 /'(1x,a9,2a10,a14)'/
data fmt2 /'(1x,a9,2i10,1pe14.5)'/
write (out,*) 'Injection Schedule:'
write (out,fmt1) 'Time', 'Row', 'Column', 'Rate(L^3/T)'
do iq = 1, nq
   call hms(tstring, tqn(iq))
   tstring = adjustl(tstring)
```

```
   write (out,fmt2) tstring(1:9), rows(iq), cols(iq), qqn(iq)
end do
return
end

subroutine rdnjct(rows, cols, tqn, qqn, nq, time, row, col, qin)
! Rdnjct is designed to read the grout injection schedule
! from an array.  The data is assumed to be in stair-step format
! and sorted according to increasing time.
! Jim Stiles, 10/12/1997.
implicit none
integer*4 row, col, rows(1), cols(1), iq, nq
real*8 qin, time, tqn(1), qqn(1)
row = 1
col = 1
qin = 0.0d+00
do iq = 1, nq
   if (tqn(iq) <= time) then
      row = rows(iq)
      col = cols(iq)
      qin = qqn(iq)
   end if
   if (tqn(iq) >= time) exit
end do
return
end

subroutine velfind(up, vp, cell, xp, yp, vx, vy, dx, dy, nx, ny, nmax)
! Velfind is designed to calculate the x and y velocity at a given
! point in a velocity field via linear interpolation.
! Jim Stiles, 11/12/1997.
implicit none
integer*4 nx, ny, nmax, i, j
integer*1 cell(nmax,nmax)
real*8 up, vp, xp, yp, vx(nmax,nmax), vy(nmax,nmax), dx(nmax), &
       dy(nmax), y1, y2, zero, x1, x2
logical flag
data zero /0.0d+00/

! Determine the appropriate model row.
y2 = zero
do i = ny, 1, -1
   y1 = y2
   y2 = y2 + dy(i)
   flag = (yp >= y1 .and. yp < y2) .or. i == 1
   flag = flag .or. (i == ny .and. yp < y1)
   if (flag) then
      x2 = zero

      ! Determine the appropriate model column.
      do j = 1, nx
         x1 = x2
         x2 = x2 + dx(j)
         flag = (xp >= x1 .and. xp < x2) .or. j == nx
         flag = flag .or. (j == 1 .and. xp < x1)
         if (flag) then
```

```
            ! Interpolate the x velocity.
            if (j == 1 .or. cell(i,j-1) == 0) then
               up = vx(i,j)
            else
               up = (xp - x1) * vx(i,j) + (x2 - xp) * vx(i,j-1)
               up = up / dx(j)
            end if
            if (cell(i,j) == 0) up = zero

            ! Interpolate the y velocity.
            if (i == 1.or. cell(i-1,j) == 0) then
               vp = vy(i,j)
            else
               vp = (yp - y1) * vy(i-1,j) + (y2 - yp) * vy(i,j)
               vp = vp / dy(i)
            end if
            if (cell(i,j) == 0) vp = zero

            ! Exit the do loops and return to calling program.
            exit
         end if
      end do
      exit
   end if
end do
return
end

subroutine velfield(h, vx, vy, vxold, vyold, cnx, cny, yldstr, dx, &
                    dy, dt, rho, g, r, ceil, eps, delta, cell, nx, &
                    ny, nmax, cr, bott, cflag, slot)
! Velfield is designed to update the velocity arrays from the
! calculated depth array.  Jim Stiles, 11/08/1997.
implicit none
integer*4 nmax
integer*1 cell(nmax,nmax), cr
integer*4 i, j, nxm1, nym1, ip1, jp1, nx, ny, slot, im1, jm1, nbrs
real*8 h(nmax,nmax), wavespeed, vx(nmax,nmax), vy(nmax,nmax), two,    &
   cnx(nmax,nmax), yldstr(nmax,nmax), vxold(nmax,nmax), aj, ai, r,    &
   vyold(nmax,nmax), fctr, dt, dx(nmax), dy(nmax), hmid, ri, rj, wp, &
   cny(nmax,nmax), sg, sox, soy, yieldmid, syield, gamma, q, zero,    &
   area, rho, eps, delta, bott(nmax,nmax), g, cmax, ceil(nmax,nmax)
logical cflag
data zero /0.0d+00/, two /2.0d+00/
nxm1 = nx - 1
nym1 = ny - 1
gamma = rho * g
do i = 1, nym1
do j = 1, nxm1
if (cell(i,j) == 0) then

   ! Ensure zero velocities for inactive cells.
   vxold(i,j) = zero
   vyold(i,j) = zero
   vx(i,j) = zero
   vy(i,j) = zero
   cnx(i,j) = zero
```

```
      cny(i,j) = zero

   else

      ! Store the old velocities.
      vxold(i,j) = vx(i,j)
      vyold(i,j) = vy(i,j)

      ! Calculate the x velocity at i,j+1/2
      fctr = dt / (dx(j) * dy(i))
      ip1 = min(i + 1,ny)
      if (cell(ip1,j) == 0) ip1 = i
      im1 = max(i - 1,1)
      if (cell(im1,j) == 0) im1 = i
      jp1 = min(j + 1,nx)
      if (cell(i,jp1) == 0) jp1 = j
      jm1 = max(j - 1,1)
      if (cell(i,jm1) == 0) jm1 = j
      hmid = (h(i,j) + h(i,jp1)) / two
      yieldmid = (yldstr(i,j) + yldstr(i,jp1)) / two
      cmax = (ceil(i,j) + ceil(i,jp1)) / two
      aj = area(hmid, dy(i), delta, cmax, eps, slot)
      rj = wp(h(i,j), h(i,jp1), ceil(i,j), ceil(i,jp1), dy(i), dx(j), &
              dx(jp1), eps, cell(i-1,j), cell(i+1,j), cell(i-1,jp1),  &
              cell(i+1,jp1), slot)
      rj = aj / rj
      sox = (bott(i,j) - bott(i,jp1)) * two / (dx(j) + dx(jp1))
      sg = two * (h(i,j) - h(i,jp1)) / (dx(j) + dx(jp1)) + sox
      syield = yieldmid / (gamma * rj)
      if (dabs(sg) > syield) then
         if (cr == 1) then
            call js(q, dabs(sg), aj, rj, rho, g, yieldmid, r)
         else
            nbrs = 0
            if (ip1 == i) nbrs = nbrs + 1
            if (im1 == i) nbrs = nbrs + 1
            call kxw(q,dabs(sg),hmid,dy(i),rho,g,yieldmid,r,cmax,nbrs)
         end if
      else
         q = zero
      end if
      if (sg < zero) q = -q
      vx(i,j) = q / aj
      hmid = dmax1(delta, hmid)
      wavespeed = dabs(vx(i,j))
      if (cflag) wavespeed = wavespeed + dsqrt(hmid * g)
      cnx(i,j) = two * wavespeed * dt / (dx(j) + dx(jp1))

      ! Calculate the y velocity at i+1/2,j
      hmid = (h(i,j) + h(ip1,j)) / two
      yieldmid = (yldstr(i,j) + yldstr(ip1,j)) / two
      cmax = (ceil(i,j) + ceil(ip1,j)) / two
      ai = area(hmid, dx(j), delta, cmax, eps, slot)
      ri = wp(h(i,j), h(ip1,j), ceil(i,j), ceil(ip1,j), dx(j), dy(i), &
              dy(ip1), eps, cell(i,j-1), cell(i,j+1), cell(ip1,j-1),  &
              cell(ip1,j+1), slot)
      ri = ai / ri
```

```
      soy = two * (bott(ip1,j) - bott(i,j)) / (dy(i) + dy(ip1))
      sg = two * (h(ip1,j) - h(i,j)) / (dy(i) + dy(ip1)) + soy
      syield = yieldmid / (gamma * ri)
      if (dabs(sg) > syield .and. ip1 /= i) then
         if (cr == 1) then
            call js(q, dabs(sg), ai, ri, rho, g, yieldmid, r)
         else
            nbrs = 0
            if (jp1 == j) nbrs = nbrs + 1
            if (jm1 == j) nbrs = nbrs + 1
            call kxw(q,dabs(sg),hmid,dx(j),rho,g,yieldmid,r,cmax,nbrs)
         end if
      else
         q = zero
      end if
      if (sg < zero) q = -q
      vy(i,j) = q / ai
      hmid = dmax1(delta, hmid)
      wavespeed = dabs(vy(i,j))
      if (cflag) wavespeed = wavespeed + dsqrt(hmid * g)
      cny(i,j) = two * wavespeed * dt / (dy(i) + dy(ip1))
   end if
   end do
   end do
   return
   end

   function wp(h1, h2, c1, c2, b, l1, l2, ep, t1, t2, t3, t4, slot)
   ! Wp is designed to calculate the wetted perimeter of a flow cross
   ! section between adjacent cells under any and all conditions.
   ! Jim Stiles, 11/22/1997.
   implicit none
   real*8 b, l1, l2, ep, wpi, wp, h1, h2, c1, c2, bmep, zero
   integer*1 t1, t2, t3, t4
   integer*4 slot
   data zero /0.0d+00/
   if (slot == 1) then
      bmep = b - ep
   else
      bmep = dsqrt(b * b / 4.0d+00 + ep * ep) / ep
   end if
   wpi = zero
   if (t1 == 0) then
      wpi = wpi + l1 * h1
      if (h1 > c1) then
         if (slot == 1) then
            wpi = wpi + l1 * bmep
         else if (slot == 2) then
            wpi = wpi + l1 * (c1 - h1)
         else
            wpi = wpi + l1 * (c1 - h1 + (h1 - c1) * bmep)
         end if
      end if
   end if
   if (t2 == 0) then
      wpi = wpi + l1 * h1
      if (h1 > c1) then
```

```
      if (slot == 1) then
         wpi = wpi + l1 * bmep
      else if (slot == 2) then
         wpi = wpi + l1 * (c1 - h1)
      else
         wpi = wpi + l1 * (c1 - h1 + (h1 - c1) * bmep)
      end if
   end if
end if
if (t3 == 0) then
   wpi = wpi + l2 * h2
   if (h2 > c2) then
      if (slot == 1) then
         wpi = wpi + l2 * bmep
      else if (slot == 2) then
         wpi = wpi + l2 * (c2 - h2)
      else
         wpi = wpi + l2 * (c2 - h2 + (h2 - c2) * bmep)
      end if
   end if
end if
if (t4 == 0) then
   wpi = wpi + l2 * h2
   if (h2 > c2) then
      if (slot == 1) then
         wpi = wpi + l2 * bmep
      else if (slot == 2) then
         wpi = wpi + l2 * (c2 - h2)
      else
         wpi = wpi + l2 * (c2 - h2 + (h2 - c2) * bmep)
      end if
   end if
end if
wp = wpi / (l1 + l2) + b
return
end

subroutine kxw(q, sf, hh, b, rho, g, yield, r, hmax, neighbors)
! Kxw is designed to calculate the discharge under the specified
! uniform conditions.  Kxw uses the semi-empirical equations
! presented in Whipple's (1997) paper on Bingham debris flow.
! Because Whipple's semi-empirical equations were derived from
! numerical simulations of one dimensional open channel flow,
! approximate formulas are employed for the more general cases.
! Jim Stiles, 12/19/1997.
implicit none
integer*4 neighbors
real*8 q, sf, b, h, rho, g, yield, r, qn, k, theta, taucr, aa, &
 one, bb, cc, gg, hh, yldn, hmax, three, qfact, two, six, ratio
data one /1.0d+00/, two /2.0d+00/, three /3.0d+00/,six /6.0d+00/

! Determine if the channel cross section is closed at the top.
if (hh >= hmax) then
   qfact = two
   h = hmax / two
else
   qfact = one
```

```
   h = hh
end if

! Calculate the channel shape factor and determine the
! coefficients for the Whipple model equation.
theta = b / h
select case (neighbors)

   case (0)
   aa = one / six
   bb = one / two
   cc = one / three
   k = one
   ratio = one

   case (1)
   gg = (2.5d+00 / theta) ** 1.7d+00 + one
   aa = (0.156d+00 / gg + one / six) / two
   bb = (0.482d+00 / gg + one / two) / two
   cc = (0.326d+00 / gg + one / three) / two
   k = one
   ratio = one / ((1.8d+00 / theta) ** 1.2d+00 + one)
   ratio = (ratio + one) / two

   case (2)
   aa = 0.156d+00
   bb = 0.482d+00
   cc = 0.326d+00
   ratio = one / ((1.8d+00 / theta) ** 1.2d+00 + one)

end select

! Calculate the normalized yield stress.
taucr = ratio * rho * g * h * sf
yldn = yield / taucr

! Calculate the channel discharge.
qn = k * ((aa * yldn * yldn - bb) * yldn + cc)
q = qn * rho * g * sf * b * h * h * h / r
q = q * qfact
return
end
```