```
program norminv2
!
!*******************************************************************************80
! Program NorMinV2 accepts partial extraction chemical analyses and
! calculates a corresponding normative copper-iron sulfide content. This
! program was written in FORTRAN 90 for Minera Escondida Ltda. using the
! GNU Fortran compiler (MinGW.org GCC-8.2.0-3.
! User documentation is provided in the accompanying report
!   < NorMinV2 Programming and User Guide >
!
! Copyright (C) 2020  Richard K. Preece
!
! This program is free software: you can redistribute it and/or modify
! it under the terms of the GNU General Public License as published by
! the Free Software Foundation, either version 3 of the License, or
! any later version
!
! This program is distributed in the hope that it will be useful,
! but WITHOUT ANY WARRANTY; without even the implied warranty of
! MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
! General Public License for more details at <https://www.gnu.org/licenses/>.
!
! File:   NorMinV2.f90
! Author: Richard Preece
!
! VERSION 1 Created August 16, 2019, NORMINESC to calculate 3-component
! sulfide assemblages. Two different mineral models were available for
! calculation of CP-CC-CV and CP-BN-(CC-CV) mineral models. The program
! reproduced algorithm, capabilities and options previously coded as
! Microsoft EXCEL spreadsheets. Additional capabilities include
!     - user selection of sulfide mineral model (with or without bornite)
!     - capability to read variably-formatted csv input files
!     - capability to write csv output file
!     - calculation of volume percent mineral content
! VERSION 2 Created February 2020, NORMINV2 to add capabilities for calculating
! 4-component oxide-sulfide mineral models. Program automatically selects mineral
! model based on the Minera Escondida Mineral Zone coding structure and certain
! analytical criteria.
! VERSION 2.1 Modified February 2021, Corrected bug in get_real subroutine that
! failed to read input data when in the last value field of the line
! VERSION 2.2 Modified June 2021, Revised MinZone codes for Leach Cap and Oxide
! for consistency with standard MEL geological codes
!           Leach Cap: Old Code= 1  New Code= 0
!           Oxide:     Old Code= 2  New Code= 1
! VERSION 2.3 Modified August 2022, Added TCu, Fe and S2 values to output file
!
! Define variable types and values
!
    integer err_flag
    real (kind = 8), DIMENSION(3,3) :: m_cpcccv, m_cpccbn, m_cpcvbn
    real (kind = 8), dimension(4,4) :: m_goccbrxc, m_cpccbrxc, m_cpcccvbr
    real (kind = 8), dimension(4,4) :: m_min1, m_min2, m_min3, m_min4
    real (kind = 8) :: d_cpcccv, d_cpccbn, d_cpcvbn
```

```
    real (kind = 8) :: d_gocbrxc, d_cpccbrxc, d_cpcccvbr
    real (kind = 8) :: d_min1, d_min2, d_min3, d_min4
    real (kind = 8) :: m33det, m44det

    character (len = 25) :: paramfile, inputcsv, outputcsv
    character (len = 25) :: dhname, sampid
    character (len = 1) :: yesno
    character (len = 80) :: dumline
    character (len = 256) :: record, rec_out

! Current program version number 23 Aug 2022
    vers = 2.3
    idebug = 0        ! Debug switch. Set = 1 to print raw & final CSR values

!===============================================================================
!    START DATA INPUT SECTION
!    To be replaced by direct IO to data base
!===============================================================================
! Welcome message to screen
    call timestamp()
    write ( *, '(a)') ' '
    write ( *, '(a, f5.1,/)') &
        'ESCONDIDA NORMATIVE MINERAL CALCULATIONS version:', vers

! Ask for parameter file name, stop program if not found
    write ( *, '(a)') '  Enter name of parameter file'
    read (*, '(a)') paramfile
    open (52, file = paramfile, status = "old", iostat = ios)
    if (ios > 0) then
        write (*, '(2a)') paramfile, ' is not found'
        stop
    end if

! Read parameter file for mineral definition parameters
    write (*, '(a)') '    Reading mineral data from parameter file'
    read (52, '(a)') dumline
    read (52, 10) cpxcu,cpxfe,cpxs,cpxas,cprfs,cprcn,cpras
    read (52, 10) ccxcu,ccxfe,ccxs,ccxas,ccrfs,ccrcn,ccras
    read (52, 10) cvxcu,cvxfe,cvxs,cvxas,cvrfs,cvrcn,cvras
    read (52, 10) bnxcu,bnxfe,bnxs,bnxas,bnrfs,bnrcn,bnras
    read (52, 10) enxcu,enxfe,enxs,enxas,enrfs,enrcn,enras
    read (52, 10) brxcu,brxfe,brxs,brxas,brrfs,brrcn,brras
    read (52, 10) xcxcu,xcxfe,xcxs,xcxas,xcrfs,xcrcn,xcras
    read (52, 10) goxcu,goxfe,goxs,goxas,gorfs,gorcn,goras
10  format (15x,7f10.3)

! Reading data file structure defined in parameter file
    write (*, '(a)') '      Now reading data file structure'
! Skip header lines
    do i = 1, 3
        read (52, '(a)') dumline
    end do
    read (52, 11) nhdr
    read (52, 11) idh
```

```
    read (52, 11) ismp
    read (52, 11) ifrom
    read (52, 11) ito
    read (52, 11) itcu
    read (52, 11) iscu
    read (52, 11) icncu
    read (52, 11) ifscu
    read (52, 11) ife
    read (52, 11) is2
    read (52, 11) imz
    read (52, 12) ias, ind_as
    read (52, 12) ibn, idef_bn
    read (52, 13) idens, dens_def
11  format (15x,i10)
12  format (15x, 2i10)
13  format (15x, i10,f10.0)
    close (52)

! Ask for data input file name, stop program if not found
    write ( *, '(a)') '  Enter name of input data file'
    read (*, '(a)') inputcsv
    open (51, file = inputcsv, status = "old", iostat = ios)
    if (ios > 0) then
        write (*, '(2a)') inputcsv, ' is not found'
        stop
    end if

! Ask for output file name, check if file currently exists
100 write ( *, '(a)') '  Enter name of output data file'
    read (*, '(a)') outputcsv
    open (61, file = outputcsv, status = "new", iostat = ios, &
          err = 101)

! Error recovery if data file currently exists
101 continue
    if (ios > 0) then
        write (*, '(2a)') outputcsv, ' already exists.'
        write (*, fmt = '(a)', advance = 'NO') &
        'Enter Y to overwrite current file, N to enter new file name >'
        read (*,'(a1)') yesno
        if (yesno == 'Y' .or. yesno == 'y') then
            open (61, file = outputcsv, status = "unknown", &
              iostat = ios, err = 100)
        else
            go to 100
        end if
    end if
    if (idebug .eq. 1) open (62, file = "debuglist.dat", status = "unknown")

! Write header record for output file
!       Version 2.3, Added TCU, Fe, S2 fields to header record
    write (61,'(6a)') &
     'dhname,f_int,t_int,sampid,err_flag,minzon,ind_bn,tcu,fe,s2,', &
     'csrcp,csrcc,csrcv,', &
```

```
161      'csrcp,csrcc,csrcv,', &
162      'csrbn,csrgo,csrbr,csrxc,cspcp,cspcc,cspcv,cspbn,cspgo,cspbr,cspxc,', &
163      'cspen,cp_wtpct,cc_wtpct,cv_wtpct,bn_wtpct,go_wtpct,br_wtpct,xc_wtpct,', &
164      'en_wtpct,py_best,xfe,cp_volpct,cc_volpct,cv_volpct,bn_volpct,', &
165      'go_volpct,br_volpct,xc_volpct,en_volpct,py_volpct'
166
167 !=============================================================================
168 !     END OF SECTION FOR SETTING UP DATA INPUT FILES
169 !      Begin section for initializing variables and reading data records
170 !=============================================================================
171      ! Initialize record counter and move to first record of input file
172     ndata = 0
173     do i = 1, nhdr
174         read (51,'(a80)') dumline
175     end do
176 !
177 ! Set matrix for end-member extraction rates of cp-cc-cv, calculate determinant
178     do j = 1, 3
179         m_cpcccv (1,j) = 1.0
180     end do
181     m_cpcccv (2,1) = cprcn
182     m_cpcccv (2,2) = ccrcn
183     m_cpcccv (2,3) = cvrcn
184     m_cpcccv (3,1) = cprfs
185     m_cpcccv (3,2) = ccrfs
186     m_cpcccv (3,3) = cvrfs
187     d_cpcccv = M33DET (m_cpcccv)
188
189 ! Set matrix for end-member extraction rates of cp-cc-bn, calculate determinant
190     do j = 1, 3
191         m_cpccbn (1,j) = 1.0
192     end do
193     m_cpccbn (2,1) = cprcn
194     m_cpccbn (2,2) = ccrcn
195     m_cpccbn (2,3) = bnrcn
196     m_cpccbn (3,1) = cprfs
197     m_cpccbn (3,2) = ccrfs
198     m_cpccbn (3,3) = bnrfs
199     d_cpccbn = M33DET (m_cpccbn)
200
201 ! Set matrix for end-member extraction rates of cp-cv-bn, calculate determinant
202     do j = 1, 3
203         m_cpcvbn (1,j) = 1.0
204     end do
205     m_cpcvbn (2,1) = cprcn
206     m_cpcvbn (2,2) = cvrcn
207     m_cpcvbn (2,3) = bnrcn
208     m_cpcvbn (3,1) = cprfs
209     m_cpcvbn (3,2) = cvrfs
210     m_cpcvbn (3,3) = bnrfs
211     d_cpcvbn = M33DET (m_cpcvbn)
212
213 ! Set matrix for end-member extraction rates of CuGoe-cc-br-chrys, calculate determinant
214     do j = 1, 4
```

```
215         m_goccbrxc (1,j) = 1.0
216     end do
217     m_goccbrxc (2,1) = gorcn
218     m_goccbrxc (2,2) = ccrcn
219     m_goccbrxc (2,3) = brrcn
220     m_goccbrxc (2,4) = xcrcn
221     m_goccbrxc (3,1) = gorfs
222     m_goccbrxc (3,2) = ccrfs
223     m_goccbrxc (3,3) = brrfs
224     m_goccbrxc (3,4) = xcrfs
225     m_goccbrxc (4,1) = goras
226     m_goccbrxc (4,2) = ccras
227     m_goccbrxc (4,3) = brras
228     m_goccbrxc (4,4) = xcras
229     d_goccbrxc = M44DET (m_goccbrxc)
230
231 ! Set matrix for end-member extraction rates of cp-cc-br-chrys, calculate determinant
232     do j = 1, 4
233         m_cpccbrxc (1,j) = 1.0
234     end do
235     m_cpccbrxc (2,1) = cprcn
236     m_cpccbrxc (2,2) = ccrcn
237     m_cpccbrxc (2,3) = brrcn
238     m_cpccbrxc (2,4) = xcrcn
239     m_cpccbrxc (3,1) = cprfs
240     m_cpccbrxc (3,2) = ccrfs
241     m_cpccbrxc (3,3) = brrfs
242     m_cpccbrxc (3,4) = xcrfs
243     m_cpccbrxc (4,1) = cpras
244     m_cpccbrxc (4,2) = ccras
245     m_cpccbrxc (4,3) = brras
246     m_cpccbrxc (4,4) = xcras
247     d_cpccbrxc = M44DET (m_cpccbrxc)
248
249 ! Set matrix for end-member extraction rates of cp-cc-cv-br, calculate determinant
250     do j = 1, 4
251         m_cpcccvbr (1,j) = 1.0
252     end do
253     m_cpcccvbr (2,1) = cprcn
254     m_cpcccvbr (2,2) = ccrcn
255     m_cpcccvbr (2,3) = cvrcn
256     m_cpcccvbr (2,4) = brrcn
257     m_cpcccvbr (3,1) = cprfs
258     m_cpcccvbr (3,2) = ccrfs
259     m_cpcccvbr (3,3) = cvrfs
260     m_cpcccvbr (3,4) = brrfs
261     m_cpcccvbr (4,1) = cpras
262     m_cpcccvbr (4,2) = ccras
263     m_cpcccvbr (4,3) = cvras
264     m_cpcccvbr (4,4) = brras
265     d_cpcccvbr = M44DET (m_cpcccvbr)
266 !
267 !-----------------------------------------------------------------------
268 !    Program reads and processes one data record at a time
```

```
!    Program reads and processes one data record at a time
!    SECTION FOR READING DATA RECORD FROM CVS FILE
!      Replace with code for reading data base record
!-----------------------------------------------------------------------
! Read new record
200 ndata = ndata + 1
    read (51, 201, end = 900) record
201 format(a)

! Parse record to retrieve data
! Find number of values (n_count) and record length (l_count)
    call csv_value_count(record, istatus, n_count, l_count)
! Call subroutine to extract data, identified by counting number of values
    ! Different subroutines used based on data type (real, integer, character)
    call get_char (dhname,idh,record,n_count,l_count,ios)
    if (ios .gt. 0) write (*,'(2(a,i6))') &
        'Error in DH Name on line ', ndata,'Error code = ',ios
    call get_char (sampid,ismp,record,n_count,l_count,ios)
    if (ios .gt. 0) write (*,'(2(a,i6))') &
        'Error in Sample ID on line ', ndata,'Error code = ',ios
    call get_real (f_int,ifrom,record,n_count,l_count,ios)
    if (ios .gt. 0) write (*,'(2(a,i6))') &
        'Error in From on line ', ndata,'Error code = ',ios
    call get_real (t_int,ito,record,n_count,l_count,ios)
    if (ios .gt. 0) write (*,'(2(a,i6))') &
        'Error in From on line ', ndata,'Error code = ',ios
    call get_real (tcu,itcu,record,n_count,l_count,ios)
    if (ios .gt. 0) write (*,'(2(a,i6))') &
        'Error in TCu on line ', ndata,'Error code = ',ios
    call get_real (scu,iscu,record,n_count,l_count,ios)
    if (ios .gt. 0) write (*,'(2(a,i6))') &
        'Error in SCu on line ', ndata,'Error code = ',ios
    call get_real (cncu,icncu,record,n_count,l_count,ios)
    if (ios .gt. 0) write (*,'(2(a,i6))') &
        'Error in CNCu on line ', ndata,'Error code = ',ios
    call get_real (fscu,ifscu,record,n_count,l_count,ios)
    if (ios .gt. 0) write (*,'(2(a,i6))') &
        'Error in FSCu on line ', ndata,'Error code = ',ios
    call get_real (fe,ife,record,n_count,l_count,ios)
    if (ios .gt. 0) write (*,'(2(a,i6))') &
        'Error in Fe on line ', ndata,'Error code = ',ios
    call get_real (s2,is2,record,n_count,l_count,ios)
    if (ios .gt. 0) write (*,'(2(a,i6))') &
        'Error in S2 on line ', ndata,'Error code = ',ios
    call get_real (as,ias,record,n_count,l_count,ios)
    if (ios .gt. 0) write (*,'(2(a,i6))') &
        'Error in As on line ', ndata,'Error code = ',ios
    call get_integer (minzon,imz,record,n_count,l_count,ios)
    if (ios .gt. 0) write (*,'(2(a,i6))') &
        'Error in MinZone on line ', ndata,'Error code = ',ios
    call get_integer (ind_bn,ibn,record,n_count,l_count,ios)
    if (ind_bn .lt. 0) ind_bn = idef_bn                    ! Set default flag value
    if (ios .gt. 0) write (*,'(2(a,i6))') &
        'Error in Bornite Flag on line ', ndata,'Error code = ',ios
```

```
322     call get_real (density,idens,record,n_count,l_count,ios)
323     if (density .lt. 0) density = dens_def         ! Set default density value
324     if (ios .gt. 0) write (*,'(2(a,i6))') &
325         'Error in Density on line ', ndata,'Error code = ',ios
326 !-------------------------------------------------------------------------
327 !   END SECTION FOR READING DATA RECORD
328 !        Begin section for processing PtXt data and calculation of
329 !         normative mineralogy
330 !-------------------------------------------------------------------------
331 ! Check for complete assay set
332     if (min(tcu,scu,cncu,fscu).lt. 0.0) then
333         write (*,'(a,i6,/,a)') 'Assay data missing in line number ',ndata, &
334                                ' Record not processed '
335         go to 200
336     end if
337 ! Check for valid mineral zone
338     if (minzon .lt. 0 .or. minzon .gt. 10) then
339         write (*,'(a,i6,/,a)') 'Invalid mineral zone in line number ',ndata, &
340                                ' Record not processed '
341         go to 200
342     end if
343 
344     if(idebug .eq. 1) &
345     write(62,'(i6,1x,a,2f10.1,i5)') ndata,dhname,f_int,t_int,minzon
346 ! ************************************************************************
347 ! Start normative mineral calculations
348 ! Six potential mineral associations are defined, 3 for sulfide and 3 for oxide/mixed
349 ! Selection of the mineral association depends on:
350 !    - logged mineral zone
351 !         MinZones 0, 1, 5 routed to oxide/mixed 4-component associations
352 !         Minzones 4, 6, 7, 8, 10 routed to sulfide 3-component associations
353 !    - analytical criteria
354 !         Sulfide sulfur <0.15% S or SCu >= CNCu routed to oxide only association
355 !    - normative mineral errors
356 !         Normative mineral CSR <-0.1 or > 1.1 routed to alternative mineral association
357 ! ************************************************************************
358     if (minzon .le. 2 .or. minzon .eq. 5) go to 400
359 
360 ! -------------------------------------------------------------------------
361 ! Start Sulfide association
362 ! -------------------------------------------------------------------------
363 ! Adjust assays for As-bearing mineral, depending on ind_as flag setting
364 ! Skip adjustment in oxide mineral zones
365     if (as .lt. 0 .or. ind_as .eq. 0) then   ! ind_as = 0 don't process As values
366         as = 0.0
367         en_wtpct = 0.0
368         go to 210
369     else if (ind_as == 1) then        ! ind_as = 1 value input as ppm As, convert to wt. %
370         as = as / 10000
371     end if                            ! ind_as = 2 value as wt. %
372 
373 ! Calculate wt % arsenic mineral from As assay
374 ! Adjust As mineral content for those samples with low TCu and elevated As
375     en_wtpct = min(as/enxas,(tcu-0.005)/enxcu) ! Check for samples with
```

```
376 210 cspen = en_wtpct*enxcu                        ! Calculate CSPen from mineral and Cu content
377 !
378 ! Adjust PtXt for contribution by As mineral
379     tcumod = tcu - cspen
380     fscumod = max(0.001,fscu - (cspen*enrfs))
381     cncumod = max(0.001,cncu - (cspen*enrcn))
382     fscu_rat = fscumod/tcumod
383     cncu_rat = cncumod/tcumod
384 
385 ! Calculation of chalcopyrite-chalcocite-covellite association, executed for
386 ! samples with ind_bn = 0
387 300 continue
388     if (ind_bn == 1) go to 320
389     csrcp_raw =((ccrcn*cvrfs)-(ccrfs*cvrcn)+(cvrcn-ccrcn)*fscu_rat   &
390                 +(ccrfs-cvrfs)*cncu_rat)/d_cpcccv
391     csrcc_raw =((cvrcn*cprfs)-(cvrfs*cprcn)+(cvrfs-cprfs)*cncu_rat   &
392                 +(cprcn-cvrcn)*fscu_rat)/d_cpcccv
393     csrcv_raw = 1-csrcp_raw-csrcc_raw
394     csrbn_raw = 0.0
395 
396 ! Set flags for calculated csr value that exceed tolerances of 10%
397     if (max(csrcp_raw,csrcc_raw,csrcv_raw).le. 1.100 .and. &
398       min(csrcp_raw,csrcc_raw,csrcv_raw).ge. -0.100) then
399         err_flag = 0
400         go to 310
401     end if
402     if (tcu < 0.15 .and. min(csrcp_raw,csrcc_raw,csrcv_raw)< -0.100) then
403         err_flag = 1          ! Low Grade flag, likely round-off errors or high As value
404         if (csrcv_raw < -0.100) then
405             if(idebug .eq. 1) &
406         write(62,'(a,3f7.3)') ' cpcccv raw ->',csrcp_raw,csrcc_raw,csrcv_raw
407             go to 400
408         end if
409     else if (csrcv_raw < -0.100) then
410         err_flag = 2          ! Elevated FSCu relative to CNCu, potentially oxidized
411         if(idebug .eq. 1) &
412         write(62,'(a,3f7.3)') ' cpcccv raw ->',csrcp_raw,csrcc_raw,csrcv_raw
413         go to 400
414     else if (csrcp_raw < -0.100) then
415         err_flag = 3          ! Elevated CNCu relative to FSCu
416     else
417         err_flag = 4          ! Undefined error
418     end if
419 !
420 ! Adjust mineral csr for csrcp < 0 and csrcv < 0
421 310 continue
422     if (csrcp_raw .lt. 0.000) then
423         csrcp1 = 0.0000
424     else if (csrcv_raw .lt. 0.000) then
425         csrcp1 = csrcp_raw/(csrcp_raw + csrcc_raw)
426     else
427         csrcp1 = csrcp_raw
428     end if
```

```
429
430     if (csrcv_raw .lt. 0.000) then
431         csrcc1 = csrcc_raw/(csrcp_raw + csrcc_raw)
432         csrcv1 = 0.0000
433     else
434         csrcc1 = csrcc_raw
435         csrcv1 = csrcv_raw
436     end if
437!
438! Second-stage adjustment, for csrcp1 > 1 and csrcc1 < 0
439     if (csrcp1 .gt. 1.000) then
440         csrcp = csrcp1/(csrcp1 + csrcv1)
441     else
442         csrcp = csrcp1
443     end if
444
445     if (csrcc1 .lt. 0.000) then
446         csrcc = 0.0000
447     else
448         csrcc = csrcc1/(csrcp1 + csrcc1 + csrcv1)
449     end if
450
451     csrcv = 1 - csrcp - csrcc
452     csrbn = 0.0000
453     csrgo = 0.0000
454     csrbr = 0.0000
455     csrxc = 0.0000
456
457     if(idebug .eq. 1) &
458     write(62,'(a,3f7.3,a,3f7.3)') ' cpcccv raw ->',csrcp_raw,csrcc_raw,csrcv_raw, &
459     ' cpcccv final -> ', csrcp, csrcc, csrcv
460     go to 600
461!
462! Calculation of chalcopyrite-bornite-chalcocite association, executed for
463! samples with ind_bn = 1
464320 continue
465     csrcp_raw =((ccrcn*bnrfs)-(ccrfs*bnrcn)+(bnrcn-ccrcn)*fscu_rat   &
466                 +(ccrfs-bnrfs)*cncu_rat)/d_cpccbn
467     csrcc_raw =((bnrcn*cprfs)-(bnrfs*cprcn)+(bnrfs-cprfs)*cncu_rat   &
468                 +(cprcn-bnrcn)*fscu_rat)/d_cpccbn
469     csrbn_raw = 1-csrcp_raw-csrcc_raw
470     csrcv_raw = 0.0000
471
472! Check for consistency with bn-cc assumption, calculate bn-cv if negative cc
473     if (csrcc_raw .le. -0.005) go to 330
474
475! Set error flag for calculated csr value that exceed tolerances of +/-10%
476     if (max(csrcp_raw,csrcc_raw,csrbn_raw).le. 1.1 .and. &
477       min(csrcp_raw,csrcc_raw,csrbn_raw).ge. -0.1) then
478         err_flag = 0
479         go to 321
480     end if
481
482     if (tcu < 0.15 .and. min(csrcp raw,csrcc raw,csrbn raw)< -0.100) then
```

```
483         err_flag = 1          ! Low Grade flag, likely round-off errors or high As value
484         if (csrbn_raw < -0.100) then
485             if (idebug .eq. 1) &
486           write(62,'(a,3f7.3)') ' cpbncc raw ->',csrcp_raw,csrbn_raw,csrcc_raw
487             go to 400
488         end if
489     else if (csrbn_raw < -0.100) then
490         err_flag = 2          ! Elevated FSCu relative to CNCu, potentially oxidized
491         if (idebug .eq. 1) &
492           write(62,'(a,3f7.3)') ' cpbncc raw ->',csrcp_raw,csrbn_raw,csrcc_raw
493         go to 400
494     else if (csrcp_raw < -0.100) then
495         err_flag = 3          ! Elevated CNCu relative to FSCu
496     else
497         err_flag = 4          ! Undefined error
498     end if
499 !
500 ! Adjust mineral csr for csrcp < 0 and csrbn < 0
501 321 continue
502     if (csrcp_raw .lt. 0.000) then
503         csrcp1 = 0.0000
504     else if (csrbn_raw .lt. 0.000) then
505         csrcp1 = csrcp_raw/(csrcp_raw + csrcc_raw)
506     else
507         csrcp1 = csrcp_raw
508     end if
509
510     if (csrbn_raw .lt. 0.000) then
511         csrcc1 = csrcc_raw/(csrcp_raw + csrcc_raw)
512         csrbn1 = 0.0000
513     else
514         csrcc1 = csrcc_raw
515         csrbn1 = csrbn_raw
516     end if
517 !
518 ! Second-stage adjustment, for csrcp1 > 1 and csrcc1 < 0
519     if (csrcp1 .gt. 1.000) then
520         csrcp = csrcp1/(csrcp1 + csrbn1)
521     else
522         csrcp = csrcp1
523     end if
524
525     if (csrcc1 .lt. 0.000) then
526         csrcc = 0.0000
527     else
528         csrcc = csrcc1/(csrcp1 + csrcc1 + csrbn1)
529     end if
530
531     csrbn = 1 - csrcp - csrcc
532     csrcv = 0.0000
533     csrgo = 0.0000
534     csrbr = 0.0000
535     csrxc = 0.0000
```

```
536
537     if(idebug .eq. 1) &
538     write(62,'(a,3f7.3,a,3f7.3)') ' cpbncc raw ->',csrcp_raw,csrbn_raw,csrcc_raw, &
539     ' cpbncc final -> ', csrcp, csrbn, csrcc
540     go to 600
541 !
542 ! Calculation of chalcopyrite-bornite-covellite association, executed for
543 ! samples with ind_bn = 1 and negative chalcocite from previous section
544 330 continue
545     csrcp_raw =((cvrcn*bnrfs)-(cvrfs*bnrcn)+(bnrcn-cvrcn)*fscu_rat   &
546                 +(cvrfs-bnrfs)*cncu_rat)/d_cpcvbn
547     csrcv_raw =((bnrcn*cprfs)-(bnrfs*cprcn)+(bnrfs-cprfs)*cncu_rat   &
548                 +(cprcn-bnrcn)*fscu_rat)/d_cpcvbn
549     csrbn_raw = 1-csrcp_raw-csrcv_raw
550     csrcc_raw = 0.0
551
552 ! Set error flag for calculated csr value that exceed tolerances of 10%
553     if (max(csrcp_raw,csrcv_raw,csrbn_raw).le. 1.1 .and. &
554       min(csrcp_raw,csrcv_raw,csrbn_raw).ge. -0.1) then
555         err_flag = 0
556         go to 331
557     end if
558
559     if (tcu < 0.15 .and. min(csrcp_raw,csrcv_raw,csrbn_raw)< -0.1) then
560         err_flag = 1          ! Low Grade flag, likely round-off errors or high As value
561     else if (csrbn_raw > 1.1) then
562         err_flag = 2          ! Elevated FSCu relative to CNCu, in cp-cc-bn association
563     else if (csrcp_raw < -0.1) then
564         err_flag = 3          ! Elevated CNCu relative to FSCu
565     else
566         err_flag = 4          ! Undefined error
567     end if
568 !
569 ! Adjust mineral csr for csrcp < 0 and csrbn < 0
570 331 continue
571     if (csrcp_raw .lt. 0.0) then
572         csrcp1 = 0.0
573     else if (csrcv_raw .lt. 0.0) then
574         csrcp1 = csrcp_raw/(csrcp_raw + csrbn_raw)
575     else
576         csrcp1 = csrcp_raw
577     end if
578
579     if (csrcv_raw .lt. 0.0) then
580         csrbn1 = csrbn_raw/(csrcp_raw + csrbn_raw)
581         csrcv1 = 0.0
582     else
583         csrcv1 = csrcv_raw
584         csrbn1 = csrbn_raw
585     end if
586 !
587 ! Second-stage adjustment, for csrcp1 > 1 and csrbn1 < 0
588     if (csrcp1 .gt. 1.0) then
589         csrcp = csrcp1/(csrcp1 + csrcv1)
```

```
590         else
591             csrcp = csrcp1
592         end if
593
594         if (csrbn1 .lt. 0.0) then
595             csrbn = 0.0
596         else
597             csrbn = csrbn1/(csrcp1 + csrcv1 + csrbn1)
598         end if
599
600         csrcv = 1.0000 - csrcp - csrbn
601         csrcc = 0.0000
602         csrgo = 0.0000
603         csrbr = 0.0000
604         csrxc = 0.0000
605
606         if(idebug .eq. 1) &
607         write(62,'(a,3f7.3,a,3f7.3)') ' cpbncv raw ->',csrcp_raw,csrbn_raw,csrcv_raw, &
608         ' cpbncv final -> ', csrcp, csrbn, csrcv
609         go to 600
610 ! ------------------------------------------------------------------------------
611 ! End of Sulfide mineral associations
612
613 ! Start Oxide/Mixed mineral associations
614 ! ------------------------------------------------------------------------------
615 400 continue
616 ! Initialize modified assay and calculate PtXt ratios
617         scu_rat = scu/tcu
618         fscu_rat = fscu/tcu
619         cncu_rat = cncu/tcu
620         tcumod = tcu
621 ! Initialize bornite and enargite components
622         csrbn = 0.0000
623         en_wtpct = 0.000
624         cspen = 0.000
625
626 ! Check for sulfide sulfur, Oxide to goe-cc-br-xc
627         if (s2 .le. 0.15 .and. s2 .ge. 0) go to 410
628
629 ! Check for cpy as insol mineral, go to cp-cc-br-xc
630         if (cncu > scu) go to 420
631
632 ! Calculation of normative minerals, goe-cc-br-xc mineral association
633 410 continue
634 ! Initialize normative mineral matrices
635 ! Mineral 1 = Cugoethite, Mineral 2 = Chalcocite, Mineral 3 = Brochantite
636 ! Mineral 4 = Chrysocolla
637         m_min1 = m_goccbrxc
638         m_min2 = m_goccbrxc
639         m_min3 = m_goccbrxc
640         m_min4 = m_goccbrxc
641 ! Insert PtXt ratios into the end-member mineral matrices and find determinant
642         m_min1(2,1) = cncu_rat
```

```
643     m_min1(3,1) = fscu_rat
644     m_min1(4,1) = scu_rat
645     d_min1 = m44det (m_min1)
646
647     m_min2(2,2) = cncu_rat
648     m_min2(3,2) = fscu_rat
649     m_min2(4,2) = scu_rat
650     d_min2 = m44det (m_min2)
651
652     m_min3(2,3) = cncu_rat
653     m_min3(3,3) = fscu_rat
654     m_min3(4,3) = scu_rat
655     d_min3 = m44det (m_min3)
656
657     m_min4(2,4) = cncu_rat
658     m_min4(3,4) = fscu_rat
659     m_min4(4,4) = scu_rat
660     d_min4 = m44det (m_min4)
661
662 ! Calculation of raw CSR values
663     csrgo_raw = d_min1/d_goccbrxc
664     csrcc_raw = d_min2/d_goccbrxc
665     csrbr_raw = d_min3/d_goccbrxc
666     csrxc_raw = d_min4/d_goccbrxc
667
668 ! Initialize error flag and check for valid csr calculations
669     err_flag = 0
670     csr_max = max(csrgo_raw,csrcc_raw,csrbr_raw,csrxc_raw)
671     csr_min = min(csrgo_raw,csrcc_raw,csrbr_raw,csrxc_raw)
672     if (csr_max .lt. 1.00001 .and. csr_min .gt. -0.00001) then
673         csrgo1 = csrgo_raw
674         csrcc1 = csrcc_raw
675         csrbr1 = csrbr_raw
676         csrxc1 = csrxc_raw
677         go to 411
678     end if
679
680 ! Check for error in each end-member correct CSR and set error flag if necessary
681     if (csrbr_raw .lt. 0.000) then
682         if (csr_max .gt. 1.100 .or. csr_min .lt. -0.100) err_flag = 5
683         csrbr1 = 0.0000
684         csrcc1 = max (0.0000, csrbr_raw + csrcc_raw)
685         csrxc1 = max (0.0000, csrxc_raw)
686         csrgo1 = max (0.0000, csrgo_raw)
687         go to 411
688     end if
689
690     if (csrcc_raw .lt. 0.000 .or. csrxc_raw .lt. 0.000) then
691         if (csr_max .gt. 1.100 .or. csr_min .lt. -0.100) err_flag = 6
692         csrgo1 = max (0.0000, csrgo_raw)
693         csrcc1 = max (0.0000, csrcc_raw)
694         csrxc1 = max (0.0000, csrxc_raw)
695         csrbr1 = max (0.0000, csrbr_raw)
696         go to 411
```

```
697     end if
698
699     if (csrgo_raw .lt. 0.000) then
700         if (csr_max .gt. 1.100 .or. csr_min .lt. -0.100) err_flag = 7
701         csrgo1 = 0.0000
702         csrcc1 = max (0.000, csrcc_raw)
703         csrxc1 = max (0.000, csrxc_raw)
704         csrbr1 = max (0.000, csrbr_raw)
705         go to 411
706     end if
707
708     if (csrmax .gt. 1.100) err_flag = 8
709     csrgo1 = csrgo_raw
710     csrcc1 = csrcc_raw
711     csrxc1 = csrxc_raw
712     csrbr1 = csrbr_raw
713
714 411 continue
715 ! Normalize adjusted csr value and set missing normative minerals to zero
716     sumcsr1 = csrgo1 + csrcc1 + csrxc1 + csrbr1
717     csrgo = csrgo1/sumcsr1
718     csrcc = csrcc1/sumcsr1
719     csrbr = csrbr1/sumcsr1
720     csrxc = csrxc1/sumcsr1
721     csrcp = 0.0000
722     csrcv = 0.0000
723
724     if(idebug .eq. 1) &
725     write(62,'(a,4f7.3,a,4f7.3)') ' goccbrxc raw ->',csrgo_raw,csrcc_raw,csrbr_raw, &
726     csrxc_raw,' goccbrxc final -> ', csrgo, csrcc, csrbr,csrxc
727     go to 600
728
729 ! Calculation of normative minerals, cp-cc-br-xc mineral association
730 420 continue
731 ! Initialize normative mineral matrices
732 ! Mineral 1 = Chalcopyrite, Mineral 2 = Chalcocite, Mineral 3 = Brochantite
733 ! Mineral 4 = Chrysocolla
734     m_min1 = m_cpccbrxc
735     m_min2 = m_cpccbrxc
736     m_min3 = m_cpccbrxc
737     m_min4 = m_cpccbrxc
738
739 ! Insert PtXt ratios into the end-member mineral matrices and find determinant
740     m_min1(2,1) = cncu_rat
741     m_min1(3,1) = fscu_rat
742     m_min1(4,1) = scu_rat
743     d_min1 = m44det (m_min1)
744
745     m_min2(2,2) = cncu_rat
746     m_min2(3,2) = fscu_rat
747     m_min2(4,2) = scu_rat
748     d_min2 = m44det (m_min2)
749
```

```
750     m_min3(2,3) = cncu_rat
751     m_min3(3,3) = fscu_rat
752     m_min3(4,3) = scu_rat
753     d_min3 = m44det (m_min3)
754
755     m_min4(2,4) = cncu_rat
756     m_min4(3,4) = fscu_rat
757     m_min4(4,4) = scu_rat
758     d_min4 = m44det (m_min4)
759
760 ! Calculation of raw CSR values
761     csrcp_raw = d_min1/d_cpccbrxc
762     csrcc_raw = d_min2/d_cpccbrxc
763     csrbr_raw = d_min3/d_cpccbrxc
764     csrxc_raw = d_min4/d_cpccbrxc
765
766 ! Initialize error flag and check for valid csr calculations
767     err_flag = 0
768     csr_max = max(csrcp_raw,csrcc_raw,csrbr_raw,csrxc_raw)
769     csr_min = min(csrcp_raw,csrcc_raw,csrbr_raw,csrxc_raw)
770     if (csr_max .lt. 1.000 .and. csr_min .gt. 0.000) then
771         csrcp1 = csrcp_raw
772         csrcc1 = csrcc_raw
773         csrbr1 = csrbr_raw
774         csrxc1 = csrxc_raw
775         go to 421
776     end if
777
778 ! Check for error in each end-member correct CSR and set error flag if necessary
779 ! Samples with negative chrysocolla is diagnostic of normative CV, take cpcccvbr branch
780     if (csrxc_raw .lt. -0.010) then
781         if (idebug .eq. 1) &
782         write(62,'(a,4f7.3)') ' cpccbrxc raw ->',csrcp_raw,csrcc_raw,csrbr_raw,csrxc_raw
783         go to 430
784     end if
785
786     if (csrcp_raw .lt. 0.000) then
787         if (csrbr_raw .lt. 0.000) then
788             if (csr_max .gt. 1.100 .or. csr_min .lt. -0.100) err_flag = 9
789             csrxc1 = max (0.0000, csrxc_raw + csrcp_raw)
790             csrcc1 = max (0.0000, csrcc_raw + csrbr_raw)
791             csrcp1 = 0.0000
792             csrbr1 = 0.0000
793             go to 421
794         end if
795         csrcp1 = 0.0000
796         csrcc1 = max (0.0000, csrcc_raw)
797         csrbr1 = max (0.0000, csrbr_raw)
798         csrxc1 = max (0.0000, csrxc_raw)
799         go to 421
800     end if
801
802     if (csr_max .gt. 1.100 .or. csr_min .lt. -0.100) err_flag = 10
803     csrcp1 = max (0.000, csrcp_raw)
```

```
804     csrcc1 = max (0.000, csrcc_raw)
805     csrxc1 = max (0.000, csrxc_raw)
806     csrbr1 = max (0.000, csrbr_raw)
807
808 421 continue
809 ! Normalize adjusted csr value and set missing normative minerals to zer
810     sumcsr1 = csrcp1 + csrcc1 + csrxc1 + csrbr1
811     csrcp = csrcp1/sumcsr1
812     csrcc = csrcc1/sumcsr1
813     csrbr = csrbr1/sumcsr1
814     csrxc = csrxc1/sumcsr1
815     csrgo = 0.0000
816     csrcv = 0.0000
817
818     if(idebug .eq. 1) &
819     write(62,'(a,4f7.3,a,4f7.3)') ' cpccbrxc raw ->',csrcp_raw,csrcc_raw,csrbr_raw, &
820     csrxc_raw,' cpccbrxc final -> ', csrcp, csrcc, csrbr,csrxc
821     go to 600
822
823 ! Calculation of normative minerals, cp-cc-cv-br mineral association
824 430 continue
825 ! Initialize normative mineral matrices
826 ! Mineral 1 = Chalcopyrite, Mineral 2 = Chalcocite, Mineral 3 = Covellite
827 ! Mineral 4 = Brochantite
828     m_min1 = m_cpcccvbr
829     m_min2 = m_cpcccvbr
830     m_min3 = m_cpcccvbr
831     m_min4 = m_cpcccvbr
832
833 ! Insert PtXt ratios into the end-member mineral matrices and find determinant
834     m_min1(2,1) = cncu_rat
835     m_min1(3,1) = fscu_rat
836     m_min1(4,1) = scu_rat
837     d_min1 = m44det (m_min1)
838
839     m_min2(2,2) = cncu_rat
840     m_min2(3,2) = fscu_rat
841     m_min2(4,2) = scu_rat
842     d_min2 = m44det (m_min2)
843
844     m_min3(2,3) = cncu_rat
845     m_min3(3,3) = fscu_rat
846     m_min3(4,3) = scu_rat
847     d_min3 = m44det (m_min3)
848
849     m_min4(2,4) = cncu_rat
850     m_min4(3,4) = fscu_rat
851     m_min4(4,4) = scu_rat
852     d_min4 = m44det (m_min4)
853
854 ! Calculation of raw CSR values
855     csrcp_raw = d_min1/d_cpcccvbr
856     csrcc_raw = d_min2/d_cpcccvbr
857     csrcv_raw = d_min3/d_cpcccvbr
```

```
858     csrbr_raw = d_min4/d_cpcccvbr
859
860 ! Initialize error flag and check for valid csr calculations
861     err_flag = 0
862     csr_max = max(csrcp_raw,csrcc_raw,csrbr_raw,csrcv_raw)
863     csr_min = min(csrcp_raw,csrcc_raw,csrbr_raw,csrcv_raw)
864     if (csr_max .lt. 1.000 .and. csr_min .gt. 0.000) then
865         csrcp1 = csrcp_raw
866         csrcc1 = csrcc_raw
867         csrcv1 = csrcv_raw
868         csrbr1 = csrbr_raw
869         go to 431
870     end if
871
872 ! Check for error in each end-member correct CSR and set error flag if necessary
873
874     if (csr_max .gt. 1.100 .or. csr_min .lt. -0.100) err_flag = 11
875     csrcp1 = max (0.000, csrcp_raw)
876     csrcc1 = max (0.000, csrcc_raw)
877     csrcv1 = max (0.000, csrcv_raw)
878     csrbr1 = max (0.000, csrbr_raw)
879
880 431 continue
881 ! Normalize adjusted csr value and set missing normative minerals to zer
882     sumcsr1 = csrcp1 + csrcc1 + csrcv1 + csrbr1
883     csrcp = csrcp1/sumcsr1
884     csrcc = csrcc1/sumcsr1
885     csrbr = csrbr1/sumcsr1
886     csrcv = csrcv1/sumcsr1
887 432 continue
888     csrgo = 0.0000
889     csrxc = 0.0000
890
891     if(idebug .eq. 1) &
892     write(62,'(a,4f7.3,a,4f7.3)') ' cpcccvbr raw ->',csrcp_raw,csrcc_raw,csrcv_raw, &
893     csrbr_raw,' cpcccvbr final -> ', csrcp, csrcc, csrcv,csrbr
894     go to 600
895 !  End of Oxide normative CSR calculations
896 !-------------------------------------------------------------------------------
897 !
898 ! Calculation of CSP, copper mineral abundance, and pyrite
899 600 continue
900 ! Set Low Grade error flag
901     if (err_flag > 0 .and. tcu .lt. 0.15) err_flag = 1
902
903 ! Calculation of Copper Source Percent (CSP))
904     cspcp = tcumod * csrcp
905     cspcc = tcumod * csrcc
906     cspcv = tcumod * csrcv
907     cspbn = tcumod * csrbn
908     cspgo = tcumod * csrgo
909     cspxc = tcumod * csrxc
910     cspbr = tcumod * csrbr
```

```
911 ! Calculation of mineral abundance (wt. %)
912       cp_wtpct = cspcp / cpxcu
913       cc_wtpct = cspcc / ccxcu
914       cv_wtpct = cspcv / cvxcu
915       bn_wtpct = cspbn / bnxcu
916       go_wtpct = cspgo / goxcu
917       xc_wtpct = cspxc / xcxcu
918       br_wtpct = cspbr / brxcu
919
920 ! Calculation of pyrite based on Fe content
921       if (fe .lt. 0) then
922           py_fe = -99.
923           xfe = -99.
924       else
925           cusul_fe = cp_wtpct*cpxfe + cc_wtpct*ccxfe + cv_wtpct*cvxfe &
926                    + bn_wtpct*bnxfe + en_wtpct*enxfe
927           py_fe = max (0.000, (fe - cusul_fe)/0.466)
928       end if
929
930 ! Calculation of pyrite based on S content
931       if (s2 .lt. 0) then
932           py_best = -99.             ! Sample missing both S and Fe assay
933           xfe = -99.
934           go to 610
935       else
936           cusul_s = cp_wtpct*cpxs + cc_wtpct*ccxs + cv_wtpct*cvxs  &
937                   + bn_wtpct*bnxs + en_wtpct*enxs
938           py_s = max (0.0, (s2 - cusul_s)/0.534)
939       end if
940
941 ! Select minimum pyrite value as best estimate.  Pooled analytical uncertainty is 11%
942
943       if (py_fe .lt. 0.0)then
944           py_best = py_s                  ! Check for missing Fe assay only
945           xfe = -99.
946           go to 610
947       end if
948
949       ave_est = (py_fe+py_s)/2.0
950       if (abs(ave_est) .lt. 0.001) then
951           py_best = 0.000                   ! Check for divide by zero
952           go to 610
953       end if
954
955       if ((abs(py_fe-py_s)/ave_est).le. 0.11) then
956           py_best = ave_est
957       else
958           py_best = min(py_fe,py_s)
959       end if
960
961 ! Calculate excess iron (non-sulfide Fe)
962       xfe = fe - cusul_fe - py_best*0.466
963       if (xfe .lt. 0.0) xfe = 0.0
964
```

```
610 continue

! Calculate volume percent sulfides from user-supplied whole rock density and
! mineral densities published by mindat.org
!
! Assignment of mineral densities (g/cc)from www.mindat.com
    sg_py = 5.01     ! Calculated value pyrite
    sg_cp = 4.18     ! Calculated value chalcopyrite
    sg_cc = 5.75     ! Calculated for djurleite, near top of measured cc range
    sg_cv = 4.60     ! Calculated value covellite
    sg_bn = 5.09     ! Calculated value bornite
    sg_en = 4.40     ! Calculated value enargite
    sg_xc = 2.10     ! Mid-range of measured values, chrysocolla
    sg_br = 3.97     ! Measured value brochantite
    sg_go = 4.28     ! Mid-range of measured values, goethite

!
! Rock density provided by data base, with default value assigned where measured
! value is missing
!
! If default density is set to zero, skip calculations
    if (density .lt. 0.01)then
        cp_volpct = -99.
        cc_volpct = -99.
        cv_volpct = -99.
        bn_volpct = -99.
        en_volpct = -99.
        py_volpct = -99.
        xc_volpct = -99.
        br_volpct = -99.
        go_volpct = -99.
        go to 620
    end if
!
! Calculation of volume percent
    cp_volpct = cp_wtpct * density / sg_cp
    cc_volpct = cc_wtpct * density / sg_cc
    cv_volpct = cv_wtpct * density / sg_cv
    bn_volpct = bn_wtpct * density / sg_bn
    en_volpct = en_wtpct * density / sg_en
    xc_volpct = xc_wtpct * density / sg_xc
    br_volpct = br_wtpct * density / sg_br
    go_volpct = go_wtpct * density / sg_go
    if (py_best .lt. 0) then
        py_volpct = -99.
    else
        py_volpct = py_best * density / sg_py
    end if
!
!================================================================================
! Calculations complete
!  Print normative mineralogy to file using cvs formatting functions
! THIS SECTION TO BE REPLACED BY DIRECT IO TO DATABASE
```

```
1018 !================================================================================
1019 ! Modified Aug 2022, Version 2.3, Added tcu, fe, and s2 to output record
1020 620 continue
1021     rec_out = ' '
1022 
1023     call csv_record_append_s  (dhname, rec_out)
1024     call csv_record_append_r4 (f_int, rec_out)
1025     call csv_record_append_r4 (t_int , rec_out)
1026     call csv_record_append_s  (sampid, rec_out)
1027     call csv_record_append_i4 (err_flag, rec_out)
1028     call csv_record_append_i4 (minzon, rec_out)
1029     call csv_record_append_i4 (ind_bn, rec_out)
1030     call csv_record_append_r4 (tcu, rec_out)
1031     call csv_record_append_r4 (fe, rec_out)
1032     call csv_record_append_r4 (s2, rec_out)
1033     call csv_record_append_r4 (csrcp, rec_out)
1034     call csv_record_append_r4 (csrcc, rec_out)
1035     call csv_record_append_r4 (csrcv, rec_out)
1036     call csv_record_append_r4 (csrbn, rec_out)
1037     call csv_record_append_r4 (csrgo, rec_out)
1038     call csv_record_append_r4 (csrbr, rec_out)
1039     call csv_record_append_r4 (csrxc, rec_out)
1040     call csv_record_append_r4 (cspcp, rec_out)
1041     call csv_record_append_r4 (cspcc, rec_out)
1042     call csv_record_append_r4 (cspcv, rec_out)
1043     call csv_record_append_r4 (cspbn, rec_out)
1044     call csv_record_append_r4 (cspgo, rec_out)
1045     call csv_record_append_r4 (cspbr, rec_out)
1046     call csv_record_append_r4 (cspxc, rec_out)
1047     call csv_record_append_r4 (cspen, rec_out)
1048     call csv_record_append_r4 (cp_wtpct, rec_out)
1049     call csv_record_append_r4 (cc_wtpct, rec_out)
1050     call csv_record_append_r4 (cv_wtpct, rec_out)
1051     call csv_record_append_r4 (bn_wtpct, rec_out)
1052     call csv_record_append_r4 (go_wtpct, rec_out)
1053     call csv_record_append_r4 (br_wtpct, rec_out)
1054     call csv_record_append_r4 (xc_wtpct, rec_out)
1055     call csv_record_append_r4 (en_wtpct, rec_out)
1056     call csv_record_append_r4 (py_best, rec_out)
1057     call csv_record_append_r4 (xfe, rec_out)
1058     call csv_record_append_r4 (cp_volpct, rec_out)
1059     call csv_record_append_r4 (cc_volpct, rec_out)
1060     call csv_record_append_r4 (cv_volpct, rec_out)
1061     call csv_record_append_r4 (bn_volpct, rec_out)
1062     call csv_record_append_r4 (go_volpct, rec_out)
1063     call csv_record_append_r4 (br_volpct, rec_out)
1064     call csv_record_append_r4 (xc_volpct, rec_out)
1065     call csv_record_append_r4 (en_volpct, rec_out)
1066     call csv_record_append_r4 (py_volpct, rec_out)
1067 
1068     call csv_file_record_write (outputcsv, 61, rec_out )
1069 !================================================================================
1070 ! Record output complete
1071 !================================================================================
```

```
1072
1073 ! Write progress indicator to screen
1074      if (mod(ndata,50).eq. 0) write (*,'(i6,a)') ndata,' records processed'
1075
1076 ! Read next line
1077        go to 200
1078
1079 ! Data file read. End program
1080 900 continue
1081      ndata = ndata - 1
1082      write (*,'(i6,a)') ndata, " records processed. Run completed."
1083      close (51)
1084      close (61)
1085      if (idebug .eq. 1) close (62)
1086      stop
1087 END PROGRAM norminv2
1088 !
1089 !=============================================================================================
1090 !=============================================================================================
1091 ! Begin subroutine and function programming.
1092 ! These are subprograms that are called in the MAIN program
1093 !---------------------------------------------------------------------------------------------
1094 ! Begin subroutines for input and output of CSV formatted files
1095 !---------------------------------------------------------------------------------------------
1096 ! Subroutines to parse data record from input record
1097 ! Separate subroutines for character, integer, and R4 real data types
1098 ! Minor error checking - when error is encountered, variable is set to 'missing'
1099 ! indicator, record_status is set to value > 0, and control is returned to calling
1100 ! statement.
1101 !---------------------------------------------------------------------------------------------
1102
1103 subroutine get_char (xdata,icol,csv_record,csv_value,csv_len,record_status)
1104 ! Subroutine to extract caracter data type from csv record
1105 ! icol is location in reoord, from 1 to value count as per parameter file input
1106 ! Written by Richard K. Preece
1107 ! August 19, 2019
1108
1109
1110      character (len = 25) xdata, cdata
1111      character csv_char
1112      character csv_char_old
1113      integer ( kind = 4) csv_len
1114      integer ( kind = 4) csv_loc
1115      character ( len = *) csv_record
1116      integer ( kind = 4) record_status
1117      integer ( kind = 4) value_count
1118      integer ( kind = 4) csv_value
1119      integer ( kind = 4) word_length
1120
1121 ! Initiate counters
1122      value_count = 0
1123      record_status = 0
1124      word_length = 0
```

```
1125     csv_char_old = ','
1126
1127 ! Check to determine if variable is missing from the input file
1128     if (icol .eq. 0) then
1129         xdata = '-99'
1130         return
1131     end if
1132
1133     do csv_loc = 1, csv_len
1134         csv_char = csv_record(csv_loc:csv_loc)
1135         !
1136         !  Check for start of new value field
1137         if (csv_char_old == ',') then
1138             value_count = value_count + 1
1139             word_length = 1
1140         else
1141              word_length = word_length + 1
1142         end if
1143             if (csv_char == ',') then
1144                 if (value_count == icol) go to 100
1145             end if
1146         csv_char_old = csv_char
1147     end do
1148
1149     if (csv_loc == csv_len) then
1150         ! data column not found, set data to -99 and return with error code
1151         record_status = 1
1152         xdata = ''
1153         return
1154     else
1155         !Unknown reason for early termination, set value to -99 and return with error code
1156         record_status = 9
1157         xdata = ''
1158         go to 101
1159     end if
1160
1161     ! Data column found. Calculate column location, read real value, and return
1162 100 continue
1163     mloc = csv_loc
1164     nloc = mloc - (word_length-1)
1165     read (csv_record(nloc:mloc), *,err=101,iostat=record_status) cdata
1166     xdata = trim(cdata)
1167
1168 101 continue
1169
1170 return
1171 end
1172
1173 subroutine get_integer (xdata,icol,csv_record,csv_value,csv_len,record_status)
1174 ! Subroutine to extract integer data type from csv record
1175 ! icol is location in record, from 1 to value count as per parameter file input
1176 ! Written by Richard K. Preece
1177 ! August 19, 2019
1178
```

```
1179
1180        integer (kind = 4) xdata
1181        character csv_char
1182        character csv_char_old
1183        integer ( kind = 4) csv_len
1184        integer ( kind = 4) csv_loc
1185        character ( len = *) csv_record
1186        integer ( kind = 4) record_status
1187        integer ( kind = 4) value_count
1188        integer ( kind = 4) csv_value
1189        integer ( kind = 4) word_length
1190
1191    ! Initiate counters
1192        value_count = 0
1193        record_status = 0
1194        word_length = 0
1195        csv_char_old = ','
1196
1197    ! Check to determine if variable is missing from the input file
1198        if (icol .eq. 0) then
1199            xdata = -99
1200            return
1201        end if
1202
1203        do csv_loc = 1, csv_len
1204            csv_char = csv_record(csv_loc:csv_loc)
1205            !
1206            !  Check for start of new value field
1207            if (csv_char_old == ',') then
1208                value_count = value_count + 1
1209                word_length = 1
1210            else
1211                 word_length = word_length + 1
1212            end if
1213                if (csv_char == ',') then
1214                    if (value_count == icol) go to 100
1215                end if
1216            csv_char_old = csv_char
1217        end do
1218
1219        if (csv_loc == csv_len) then
1220            ! data column not found, set data to -99 and return with error code
1221            record_status = 1
1222            xdata = -99
1223            return
1224        else
1225            !Unknown reason for early termination, set value to -99 and return with error code
1226            record_status = 9
1227            xdata = -99
1228            go to 101
1229        end if
1230
1231        ! Data column found. Calculate column location, read real value, and return
```

```
1232 100 continue
1233     mloc = csv_loc
1234     nloc = mloc - (word_length-1)
1235     read (csv_record(nloc:mloc), *,err=101,iostat=record_status) zdata
1236     xdata = int(zdata)
1237
1238 101 continue
1239
1240 return
1241 end
1242
1243 subroutine get_real (xdata,icol,csv_record,csv_value,csv_len,record_status)
1244 ! Subroutine to extract real data type from csv record
1245 ! icol is location in reoord, from 1 to value count as per parameter file input
1246 ! Written by Richard K. Preece
1247 ! August 16, 2019
1248
1249
1250     real (kind = 4) xdata
1251     character csv_char
1252     character csv_char_old
1253     integer ( kind = 4) csv_len
1254     integer ( kind = 4) csv_loc
1255     character ( len = *) csv_record
1256     integer ( kind = 4) record_status
1257     integer ( kind = 4) value_count
1258     integer ( kind = 4) csv_value
1259     integer ( kind = 4) word_length
1260
1261 ! Initiate counters
1262     value_count = 0
1263     record_status = 0
1264     word_length = 0
1265     csv_char_old = ','
1266
1267 ! Check to determine if variable is missing from the input file
1268     if (icol .eq. 0) then
1269         xdata = -99.0
1270         return
1271     end if
1272
1273     do csv_loc = 1, csv_len
1274         csv_char = csv_record(csv_loc:csv_loc)
1275         !
1276         !  Check for start of new value field
1277         if (csv_char_old == ',') then
1278             value_count = value_count + 1
1279             word_length = 1
1280         else
1281             word_length = word_length + 1
1282         end if
1283             if (csv_char == ',') then
1284                 if (value_count == icol) go to 100
1285             end if
```

```
1286            csv_char_old = csv_char
1287        end do
1288
1289        ! Check if item is in last value on line
1290        if (value_count == icol) then
1291            word_length = word_length + 1
1292            go to 100
1293        end if
1294
1295        if (csv_loc == csv_len) then
1296            ! data column not found, set data to -99 and return with error code
1297            record_status = 1
1298            xdata = -99.0
1299            return
1300        else
1301            !Unknown reason for early termination, set value to -99 and return with error code
1302            record_status = 9
1303            xdata = -99.0
1304            go to 101
1305        end if
1306
1307        ! Data column found. Calculate column location, read real value, and return
1308100 continue
1309        mloc = csv_loc
1310        nloc = mloc - (word_length-1)
1311        read (csv_record(nloc:mloc), *,err=101,iostat=record_status) xdata
1312
1313101 continue
1314
1315return
1316end
1317!-----------------------------------------------------------------------------
1318! Public domain subroutines to manage CSV-formatted input and output files
1319!-----------------------------------------------------------------------------
1320subroutine csv_value_count(csv_record, csv_record_status, value_count, csv_len)
1321
1322    !*****************************************************************************80
1323    !
1324    !! CSV_COUNT counts the number of values in a CSV record.
1325    !
1326    !  Licensing:
1327    !
1328    !    This code is distributed under the GNU LGPL license.
1329    !
1330    !  Modified:
1331    !
1332    !    28 November 2008
1333    !
1334    !  Author:
1335    !
1336    !    John Burkardt
1337    !
1338    !  Parameters:
```

```
1339    !
1340    implicit none
1341
1342    character csv_char
1343    character csv_char_old
1344    integer ( kind = 4) csv_len
1345    integer ( kind = 4) csv_loc
1346    character ( len = *) csv_record
1347    integer ( kind = 4) csv_record_status
1348    character :: TAB = achar(9)
1349    integer ( kind = 4) value_count
1350    integer ( kind = 4) word_length
1351    !
1352    !  No values so far.
1353    !
1354    value_count = 0
1355    !
1356    !  We begin in "unquoted" status.
1357    !
1358    csv_record_status = 0
1359    !
1360    !  How many characters in the record?
1361    !
1362    csv_len = len_trim(csv_record)
1363    !
1364    !  Count number of characters in each word.
1365    !
1366    word_length = 0
1367    !
1368    !  Consider each character.
1369    !
1370    csv_char_old = ','
1371
1372    do csv_loc = 1, csv_len
1373
1374        csv_char = csv_record(csv_loc:csv_loc)
1375        !
1376        !  Each comma divides one value from another.
1377        !
1378        if (csv_char_old == ',') then
1379
1380            value_count = value_count + 1
1381            word_length = 0
1382            !
1383            !  For quotes, try using CSV_RECORD_STATUS to count the number of
1384            !  quoted characters.
1385            !
1386        else if (csv_char == '"') then
1387
1388            if (0 < csv_record_status) then
1389            csv_record_status = 0
1390        else
1391            csv_record_status = csv_record_status + 1
1392            end if
```

```
1393            !
1394            !  Ignore blanks
1395            !
1396        else if (csv_char == ' ' .or. csv_char == TAB) then
1397            !
1398            !  Add character to length of word.
1399            !
1400        else
1401
1402            word_length = word_length + 1
1403
1404            if (value_count == 0) then
1405            value_count = 1
1406            end if
1407
1408        end if
1409
1410        csv_char_old = csv_char
1411
1412    end do
1413
1414    return
1415end
1416subroutine csv_file_close_read(csv_file_name, csv_file_unit)
1417
1418    !*******************************************************************************80
1419    !
1420    !! CSV_FILE_CLOSE_READ closes a CSV file for reading.
1421    !
1422    !  Licensing:
1423    !
1424    !    This code is distributed under the GNU LGPL license.
1425    !
1426    !  Modified:
1427    !
1428    !    29 November 2008
1429    !
1430    !  Author:
1431    !
1432    !    John Burkardt
1433    !
1434    !  Parameters:
1435    !
1436    !    Input, character ( len = * ) CSV_FILE_NAME, the name of the file.
1437    !
1438    !    Input, integer ( kind = 4 ) CSV_FILE_UNIT, the unit number
1439    !
1440    implicit none
1441
1442    character ( len = *) csv_file_name
1443    integer ( kind = 4) csv_file_unit
1444
1445    close ( unit = csv_file_unit)
```

```
1446
1447      return
1448  end
1449  subroutine csv_file_close_write(csv_file_name, csv_file_unit)
1450
1451      !*****************************************************************************80
1452      !
1453      !! CSV_FILE_CLOSE_WRITE closes a CSV file for writing.
1454      !
1455      !  Licensing:
1456      !
1457      !    This code is distributed under the GNU LGPL license.
1458      !
1459      !  Modified:
1460      !
1461      !    22 November 2008
1462      !
1463      !  Author:
1464      !
1465      !    John Burkardt
1466      !
1467      !  Parameters:
1468      !
1469      !    Input, character ( len = * ) CSV_FILE_NAME, the name of the file.
1470      !
1471      !    Input, integer ( kind = 4 ) CSV_FILE_UNIT, the unit number
1472      !
1473      implicit none
1474
1475      character ( len = *) csv_file_name
1476      integer ( kind = 4) csv_file_unit
1477
1478      close ( unit = csv_file_unit)
1479
1480      return
1481  end
1482  subroutine csv_file_header_write(csv_file_name, csv_file_unit, header)
1483
1484      !*****************************************************************************80
1485      !
1486      !! CSV_FILE_HEADER_WRITE writes a header to a CSV file.
1487      !
1488      !  Licensing:
1489      !
1490      !    This code is distributed under the GNU LGPL license.
1491      !
1492      !  Modified:
1493      !
1494      !    22 November 2008
1495      !
1496      !  Author:
1497      !
1498      !    John Burkardt
1499      !
```

```
1500      !  Parameters:
1501      !
1502      !    Input, character ( len = * ) CSV_FILE_NAME, the name of the file.
1503      !
1504      !    Input, integer ( kind = 4 ) CSV_FILE_UNIT, the unit number
1505      !
1506      !    Input, character ( len = * ) HEADER, the header.
1507      !
1508      implicit none
1509
1510      character ( len = *) csv_file_name
1511      integer ( kind = 4) csv_file_unit
1512      character ( len = *) header
1513
1514      write ( csv_file_unit, '(a)') trim(header)
1515
1516      return
1517  end
1518  subroutine csv_file_line_count(csv_file_name, line_num)
1519
1520      !*****************************************************************************80
1521      !
1522      !! CSV_FILE_LINE_COUNT counts the number of lines in a CSV file.
1523      !
1524      !  Discussion:
1525      !
1526      !    This routine does not try to distinguish the possible header line,
1527      !    blank lines, or cases where a single CSV record extends over multiple
1528      !    lines.  It simply counts the number of lines.
1529      !
1530      !  Licensing:
1531      !
1532      !    This code is distributed under the GNU LGPL license.
1533      !
1534      !  Modified:
1535      !
1536      !    22 November 2008
1537      !
1538      !  Author:
1539      !
1540      !    John Burkardt
1541      !
1542      !  Parameters:
1543      !
1544      !    Input, character ( len = * ) CSV_FILE_NAME, the name of the file.
1545      !
1546      !    Output, integer ( kind = 4 ) LINE_NUM, the number of lines.
1547      !
1548      implicit none
1549
1550      character ( len = *) csv_file_name
1551      integer ( kind = 4) ierror
1552      integer ( kind = 4) input_status
1553      integer ( kind = 4) input_unit
```

```
1553     integer ( kind = 4) input_unit
1554     character ( len = 1023) line
1555     integer ( kind = 4) line_num
1556
1557     line_num = -1
1558
1559     call get_unit(input_unit)
1560
1561     open ( unit = input_unit, file = csv_file_name, status = 'old', &
1562     iostat = input_status)
1563
1564     if (input_status /= 0) then
1565         write ( *, '(a)') ' '
1566         write ( *, '(a)') 'CSV_FILE_LINE_COUNT - Fatal error!'
1567         write ( *, '(a,i8)') '  Could not open "' // trim(csv_file_name) // '".'
1568         stop
1569     end if
1570
1571     line_num = 0
1572
1573     do
1574
1575         read ( input_unit, '(a)', iostat = input_status) line
1576
1577         if (input_status /= 0) then
1578             ierror = line_num
1579             exit
1580         end if
1581
1582         line_num = line_num + 1
1583
1584     end do
1585
1586     close ( unit = input_unit)
1587
1588     return
1589 end
1590 subroutine csv_file_record_write(csv_file_name, csv_file_unit, record)
1591
1592     !*****************************************************************************80
1593     !
1594     !! CSV_FILE_RECORD_WRITE writes a record to a CSV file.
1595     !
1596     !  Licensing:
1597     !
1598     !    This code is distributed under the GNU LGPL license.
1599     !
1600     !  Modified:
1601     !
1602     !    22 November 2008
1603     !
1604     !  Author:
1605     !
1606     !    John Burkardt
```

```
1607    !
1608    !  Parameters:
1609    !
1610    !    Input, character ( len = * ) CSV_FILE_NAME, the name of the file.
1611    !
1612    !    Input, integer ( kind = 4 ) CSV_FILE_UNIT, the unit number
1613    !
1614    !    Input, character ( len = * ) RECORD, the record.
1615    !
1616    implicit none
1617
1618    character ( len = *) csv_file_name
1619    integer ( kind = 4) csv_file_unit
1620    character ( len = *) record
1621
1622    write ( csv_file_unit, '(a)') trim(record)
1623
1624    return
1625end
1626subroutine csv_file_open_read(csv_file_name, csv_file_unit)
1627
1628    !*****************************************************************************80
1629    !
1630    !! CSV_FILE_OPEN_READ opens a CSV file for reading.
1631    !
1632    !  Licensing:
1633    !
1634    !    This code is distributed under the GNU LGPL license.
1635    !
1636    !  Modified:
1637    !
1638    !    29 November 2008
1639    !
1640    !  Author:
1641    !
1642    !    John Burkardt
1643    !
1644    !  Parameters:
1645    !
1646    !    Input, character ( len = * ) CSV_FILE_NAME, the name of the file.
1647    !
1648    !    Output, integer ( kind = 4 ) CSV_FILE_UNIT, the unit number
1649    !
1650    implicit none
1651
1652    character ( len = *) csv_file_name
1653    integer ( kind = 4) csv_file_status
1654    integer ( kind = 4) csv_file_unit
1655
1656    call get_unit(csv_file_unit)
1657
1658    open ( unit = csv_file_unit, file = csv_file_name, status = 'old', &
1659    iostat = csv_file_status)
1660
```

```
1661      if (csv_file_status /= 0) then
1662          write ( *, '(a)') ' '
1663          write ( *, '(a)') 'CSV_FILE_OPEN_READ - Fatal error!'
1664          write ( *, '(a,i8)') '  Could not open "' // trim(csv_file_name) // '".'
1665          csv_file_unit = -1
1666          stop
1667      end if
1668
1669      return
1670  end
1671  subroutine csv_file_open_write(csv_file_name, csv_file_unit)
1672
1673      !*******************************************************************************80
1674      !
1675      !! CSV_FILE_OPEN_WRITE opens a CSV file for writing.
1676      !
1677      !  Licensing:
1678      !
1679      !    This code is distributed under the GNU LGPL license.
1680      !
1681      !  Modified:
1682      !
1683      !    22 November 2008
1684      !
1685      !  Author:
1686      !
1687      !    John Burkardt
1688      !
1689      !  Parameters:
1690      !
1691      !    Input, character ( len = * ) CSV_FILE_NAME, the name of the file.
1692      !
1693      !    Output, integer ( kind = 4 ) CSV_FILE_UNIT, the unit number
1694      !
1695      implicit none
1696
1697      character ( len = *) csv_file_name
1698      integer ( kind = 4) csv_file_status
1699      integer ( kind = 4) csv_file_unit
1700
1701      call get_unit(csv_file_unit)
1702
1703      open ( unit = csv_file_unit, file = csv_file_name, status = 'replace', &
1704      iostat = csv_file_status)
1705
1706      if (csv_file_status /= 0) then
1707          write ( *, '(a)') ' '
1708          write ( *, '(a)') 'CSV_FILE_OPEN_WRITE - Fatal error!'
1709          write ( *, '(a,i8)') '  Could not open "' // trim(csv_file_name) // '".'
1710          stop
1711      end if
1712
1713      return
```

```
1714 end
1715 subroutine csv_record_append_i4(i4, record)
1716
1717     !*****************************************************************************80
1718     !
1719     !! CSV_RECORD_APPEND_I4 appends an I4 to a CSV record.
1720     !
1721     !  Licensing:
1722     !
1723     !    This code is distributed under the GNU LGPL license.
1724     !
1725     !  Modified:
1726     !
1727     !    24 November 2008
1728     !
1729     !  Author:
1730     !
1731     !    John Burkardt
1732     !
1733     !  Parameters:
1734     !
1735     !    Input, integer ( kind = 4 ) I4, the integer to be appended
1736     !
1737     !    Input/output, character ( len = * ) RECORD, the CSV record.
1738     !
1739     implicit none
1740
1741     character ( len = 5) fmat
1742     integer ( kind = 4) i
1743     integer ( kind = 4) i4
1744     integer ( kind = 4) i4_len
1745     integer ( kind = 4) i4_width
1746     character ( len = *) record
1747     !
1748     !  Locate last used location in RECORD.
1749     !
1750     i = len_trim(record)
1751     !
1752     !  Append comma.
1753     !
1754     if (0 < i) then
1755         i = i + 1
1756         record(i:i) = ','
1757     end if
1758     !
1759     !  Determine "width" of I4.
1760     !
1761     i4_len = i4_width(i4)
1762     !
1763     !  Create format for I4.
1764     !
1765     write ( fmat, '(a,i2,a)') '(i', i4_len, ')'
1766     !
1767     !  Write I4 to RECORD.
```

```
1768     !
1769     write ( record(i + 1:i + i4_len), fmat) i4
1770
1771     return
1772 end
1773 subroutine csv_record_append_r4(r4, record)
1774
1775     !*****************************************************************************80
1776     !
1777     !! CSV_RECORD_APPEND_R4 appends an R4 to a CSV record.
1778     !
1779     !  Licensing:
1780     !
1781     !    This code is distributed under the GNU LGPL license.
1782     !
1783     !  Modified:
1784     !
1785     !    29 November 2008
1786     !
1787     !  Author:
1788     !
1789     !    John Burkardt, modified by R.K. Preece 19 Aug 2019
1790     !
1791     !  Parameters:
1792     !
1793     !    Input, real ( kind = 8 ) R4, the value to be appended
1794     !
1795     !    Input/output, character ( len = * ) RECORD, the CSV record.
1796     !
1797     implicit none
1798
1799     character ( len = 5) fmat
1800     character ( len = 7) fmat2
1801     integer ( kind = 4) i
1802     integer ( kind = 4) i4
1803     integer ( kind = 4) i4_len
1804     integer ( kind = 4) i4_width
1805     real ( kind = 4) r4
1806     character ( len = *) record
1807     !
1808     !  Locate last used location in RECORD.
1809     !
1810     i = len_trim(record)
1811     !
1812     !  Append comma.
1813     !
1814     if (0 < i) then
1815         i = i + 1
1816         record(i:i) = ','
1817     end if
1818
1819     if (r4 == 0.0) then
1820         i = i + 1
```

```
1821            record(i:i) = '0'
1822        else if (r4 == real ( int ( r4), kind = 4)) then
1823            i4 = int ( r4)
1824            i4_len = i4_width(i4)
1825            write ( fmat, '(a,i2,a)') '(i', i4_len, ')'
1826            write ( record(i + 1:i + i4_len), fmat) i4
1827        else
1828    ! This branch for writing R4 variable modified by RK Preece, 19 August 2019
1829    ! Changed from g18.7 format to variable width f*.3 format
1830            i4 = int (r4*1000)
1831            i4_len = i4_width(i4) + 2
1832            if (i4_len .lt. 5) then
1833                if (r4 .lt. 0) then
1834                    i4_len = 6
1835                else
1836                    i4_len = 5
1837                end if
1838            end if
1839            write ( fmat2, '(a,i2,a)') '(f', i4_len, '.3)'
1840            write ( record(i + 1:i + i4_len), fmat2) r4
1841    !        write ( record(i + 1:i + 8), '(f8.3)') r4
1842        end if
1843
1844        return
1845    end
1846    subroutine csv_record_append_r8(r8, record)
1847
1848        !*****************************************************************************80
1849        !
1850        !! CSV_RECORD_APPEND_R8 appends an R8 to a CSV record.
1851        !
1852        !  Licensing:
1853        !
1854        !    This code is distributed under the GNU LGPL license.
1855        !
1856        !  Modified:
1857        !
1858        !    29 November 2008
1859        !
1860        !  Author:
1861        !
1862        !    John Burkardt
1863        !
1864        !  Parameters:
1865        !
1866        !    Input, real ( kind = 8 ) R8, the value to be appended
1867        !
1868        !    Input/output, character ( len = * ) RECORD, the CSV record.
1869        !
1870        implicit none
1871
1872        character ( len = 5) fmat
1873        integer ( kind = 4) i
1874        integer ( kind = 4) i4
```

```
1875     integer ( kind = 4) i4_len
1876     integer ( kind = 4) i4_width
1877     real ( kind = 8) r8
1878     character ( len = *) record
1879     !
1880     !  Locate last used location in RECORD.
1881     !
1882     i = len_trim(record)
1883     !
1884     !  Append comma.
1885     !
1886     if (0 < i) then
1887         i = i + 1
1888         record(i:i) = ','
1889     end if
1890
1891     if (r8 == 0.0D+00) then
1892         i = i + 1
1893         record(i:i) = '0'
1894     else if (r8 == real ( int ( r8), kind = 8)) then
1895         i4 = int ( r8)
1896         i4_len = i4_width(i4)
1897         write ( fmat, '(a,i2,a)') '(i', i4_len, ')'
1898         write ( record(i + 1:i + i4_len), fmat) i4
1899     else
1900         write ( record(i + 1:i + 14), '(g14.6)') r8
1901     end if
1902
1903     return
1904 end
1905 subroutine csv_record_append_s(s, record)
1906
1907     !*****************************************************************************80
1908     !
1909     !! CSV_RECORD_APPEND_S appends a string to a CSV record.
1910     !
1911     !  Licensing:
1912     !
1913     !    This code is distributed under the GNU LGPL license.
1914     !
1915     !  Modified:
1916     !
1917     !    26 November 2008
1918     !
1919     !  Author:
1920     !
1921     !    John Burkardt
1922     !
1923     !  Parameters:
1924     !
1925     !    Input, character ( len = * ) S, the string to be appended
1926     !
1927     !    Input/output, character ( len = * ) RECORD, the CSV record.
```

```
1928    !
1929    implicit none
1930
1931    integer ( kind = 4) i
1932    character ( len = *) record
1933    character ( len = *) s
1934    integer ( kind = 4) s_len
1935    !
1936    !  Locate last used location in RECORD.
1937    !
1938    i = len_trim(record)
1939    !
1940    !  Append a comma.
1941    !
1942    if (0 < i) then
1943        i = i + 1
1944        record(i:i) = ','
1945    end if
1946    !
1947    !  Prepend a quote.
1948    !
1949    !i = i + 1
1950    !record(I:i) = '"'
1951    !
1952    !  Write S to RECORD.
1953    !
1954    s_len = len_trim(s)
1955    record(i + 1:i + s_len) = s(1:s_len)
1956    i = i + s_len
1957    !
1958    !  Postpend a quote
1959    !
1960    !i = i + 1
1961    !record(i:i) = '"'
1962
1963    return
1964end
1965subroutine get_unit(iunit)
1966
1967    !*****************************************************************************80
1968    !
1969    !! GET_UNIT returns a free FORTRAN unit number.
1970    !
1971    !  Discussion:
1972    !
1973    !    A "free" FORTRAN unit number is an integer between 1 and 99 which
1974    !    is not currently associated with an I/O device.  A free FORTRAN unit
1975    !    number is needed in order to open a file with the OPEN command.
1976    !
1977    !    If IUNIT = 0, then no free FORTRAN unit could be found, although
1978    !    all 99 units were checked (except for units 5, 6 and 9, which
1979    !    are commonly reserved for console I/O).
1980    !
1981    !    Otherwise, IUNIT is an integer between 1 and 99, representing a
```

```
1982    !    free FORTRAN unit.  Note that GET_UNIT assumes that units 5 and 6
1983    !    are special, and will never return those values.
1984    !
1985    !  Licensing:
1986    !
1987    !    This code is distributed under the GNU LGPL license.
1988    !
1989    !  Modified:
1990    !
1991    !    26 October 2008
1992    !
1993    !  Author:
1994    !
1995    !    John Burkardt
1996    !
1997    !  Parameters:
1998    !
1999    !    Output, integer ( kind = 4 ) IUNIT, the free unit number.
2000    !
2001    implicit none
2002
2003    integer ( kind = 4) i
2004    integer ( kind = 4) ios
2005    integer ( kind = 4) iunit
2006    logical lopen
2007
2008    iunit = 0
2009
2010    do i = 1, 99
2011
2012        if (i /= 5 .and. i /= 6 .and. i /= 9) then
2013
2014            inquire ( unit = i, opened = lopen, iostat = ios)
2015
2016            if (ios == 0) then
2017                if (.not.lopen) then
2018                    iunit = i
2019                    return
2020                end if
2021            end if
2022
2023        end if
2024
2025    end do
2026
2027    return
2028end
2029function i4_log_10(i)
2030
2031    !*****************************************************************************80
2032    !
2033    !! I4_LOG_10 returns the integer part of the logarithm base 10 of an I4.
2034    !
```

```
2035    !  Discussion:
2036    !
2037    !    I4_LOG_10 ( I ) + 1 is the number of decimal digits in I.
2038    !
2039    !    An I4 is an integer ( kind = 4 ) value.
2040    !
2041    !  Example:
2042    !
2043    !        I  I4_LOG_10
2044    !    -----  --------
2045    !        0    0
2046    !        1    0
2047    !        2    0
2048    !        9    0
2049    !       10    1
2050    !       11    1
2051    !       99    1
2052    !      100    2
2053    !      101    2
2054    !      999    2
2055    !     1000    3
2056    !     1001    3
2057    !     9999    3
2058    !    10000    4
2059    !
2060    !  Licensing:
2061    !
2062    !    This code is distributed under the GNU LGPL license.
2063    !
2064    !  Modified:
2065    !
2066    !    08 June 2003
2067    !
2068    !  Author:
2069    !
2070    !    John Burkardt
2071    !
2072    !  Parameters:
2073    !
2074    !    Input, integer ( kind = 4 ) I, the number whose logarithm base 10
2075    !    is desired.
2076    !
2077    !    Output, integer ( kind = 4 ) I4_LOG_10, the integer part of the
2078    !    logarithm base 10 of the absolute value of X.
2079    !
2080    implicit none
2081
2082    integer ( kind = 4) i
2083    integer ( kind = 4) i_abs
2084    integer ( kind = 4) i4_log_10
2085    integer ( kind = 4) ten_pow
2086
2087    if (i == 0) then
2088
```

```
2089        i4_log_10 = 0
2090
2091    else
2092
2093        i4_log_10 = 0
2094        ten_pow = 10
2095
2096        i_abs = abs(i)
2097
2098        do while (ten_pow <= i_abs)
2099            i4_log_10 = i4_log_10 + 1
2100            ten_pow = ten_pow * 10
2101        end do
2102
2103    end if
2104
2105    return
2106end
2107function i4_width(i)
2108
2109    !*****************************************************************************80
2110    !
2111    !! I4_WIDTH returns the "width" of an I4.
2112    !
2113    !  Discussion:
2114    !
2115    !    The width of an integer is the number of characters necessary to print it.
2116    !
2117    !    The width of an integer can be useful when setting the appropriate output
2118    !    format for a vector or array of values.
2119    !
2120    !    An I4 is an integer ( kind = 4 ) value.
2121    !
2122    !  Example:
2123    !
2124    !        I  I4_WIDTH
2125    !    -----  -------
2126    !    -1234    5
2127    !     -123    4
2128    !      -12    3
2129    !       -1    2
2130    !        0    1
2131    !        1    1
2132    !       12    2
2133    !      123    3
2134    !     1234    4
2135    !    12345    5
2136    !
2137    !  Licensing:
2138    !
2139    !    This code is distributed under the GNU LGPL license.
2140    !
2141    !  Modified:
```

```
2142    !
2143    !    04 December 2004
2144    !
2145    !  Author:
2146    !
2147    !    John Burkardt
2148    !
2149    !  Parameters:
2150    !
2151    !    Input, integer ( kind = 4 ) I, the number whose width is desired.
2152    !
2153    !    Output, integer ( kind = 4 ) I4_WIDTH, the number of characters
2154    !    necessary to represent the integer in base 10, including a negative
2155    !    sign if necessary.
2156    !
2157    implicit none
2158
2159    integer ( kind = 4) i
2160    integer ( kind = 4) i4_log_10
2161    integer ( kind = 4) i4_width
2162
2163    if (0 < i) then
2164        i4_width = i4_log_10(i) + 1
2165    else if (i == 0) then
2166        i4_width = 1
2167    else if (i < 0) then
2168        i4_width = i4_log_10(i) + 2
2169    end if
2170
2171    return
2172end
2173subroutine timestamp()
2174
2175    !*******************************************************************************80
2176    !
2177    !! TIMESTAMP prints the current YMDHMS date as a time stamp.
2178    !
2179    !  Example:
2180    !
2181    !    31 May 2001   9:45:54.872 AM
2182    !
2183    !  Licensing:
2184    !
2185    !    This code is distributed under the GNU LGPL license.
2186    !
2187    !  Modified:
2188    !
2189    !    18 May 2013
2190    !
2191    !  Author:
2192    !
2193    !    John Burkardt
2194    !
2195    !  Parameters:
```

```
2196    !
2197    !    None
2198    !
2199    implicit none
2200
2201    character ( len = 8) ampm
2202    integer ( kind = 4) d
2203    integer ( kind = 4) h
2204    integer ( kind = 4) m
2205    integer ( kind = 4) mm
2206    character ( len = 9), parameter, dimension(12) :: month = (/ &
2207    'January  ', 'February ', 'March    ', 'April    ', &
2208    'May      ', 'June     ', 'July     ', 'August   ', &
2209    'September', 'October  ', 'November ', 'December ' /)
2210    integer ( kind = 4) n
2211    integer ( kind = 4) s
2212    integer ( kind = 4) values(8)
2213    integer ( kind = 4) y
2214
2215    call date_and_time(values = values)
2216
2217    y = values(1)
2218    m = values(2)
2219    d = values(3)
2220    h = values(5)
2221    n = values(6)
2222    s = values(7)
2223    mm = values(8)
2224
2225    if (h < 12) then
2226        ampm = 'AM'
2227    else if (h == 12) then
2228        if (n == 0 .and. s == 0) then
2229        ampm = 'Noon'
2230    else
2231        ampm = 'PM'
2232        end if
2233    else
2234        h = h - 12
2235        if (h < 12) then
2236        ampm = 'PM'
2237    else if (h == 12) then
2238        if (n == 0 .and. s == 0) then
2239        ampm = 'Midnight'
2240    else
2241        ampm = 'AM'
2242        end if
2243        end if
2244    end if
2245
2246    write ( *, '(i2,1x,a,1x,i4,2x,i2,a1,i2.2,a1,i2.2,a1,i3.3,1x,a)') &
2247    d, trim(month(m)), y, h, ':', n, ':', s, '.', mm, trim(ampm)
2248
2249    return
```

```
2249      return
2250    end
2251    !
2252    !-------------------------------------------------------------------------------
2253    ! Begin functions for finding the determinant of an algebraic matrix
2254    ! These are calls from within an equation of the MAIN program
2255    ! In both functions, the array is passed to function A and the calculated determinant
2256    ! is returned to the calling equation
2257    !-------------------------------------------------------------------------------
2258    FUNCTION M33DET (A) RESULT (DET)
2259    !*****************************************************************************************
2260    !  M33DET  -  Function to compute the determinant of a 3x3 matrix.
2261    !  Programmer:   David G. Simpson
2262    !                NASA Goddard Space Flight Center
2263    !                Greenbelt, Maryland  20771
2264    !  Date:         July 22, 2005
2265    !  Language:     Fortran-90
2266    !  Version:      1.00a
2267    !!****************************************************************************************
2268
2269          IMPLICIT NONE
2270          real (kind = 8), DIMENSION(3,3), INTENT(IN)  :: A
2271          real (kind = 8) :: DET
2272
2273
2274          DET =   A(1,1)*A(2,2)*A(3,3)  &
2275                - A(1,1)*A(2,3)*A(3,2)  &
2276                - A(1,2)*A(2,1)*A(3,3)  &
2277                + A(1,2)*A(2,3)*A(3,1)  &
2278                + A(1,3)*A(2,1)*A(3,2)  &
2279                - A(1,3)*A(2,2)*A(3,1)
2280
2281          RETURN
2282
2283    END FUNCTION M33DET
2284    FUNCTION M44DET (A) RESULT (DET)
2285    !*****************************************************************************************
2286    !  Function:     M44DET
2287    !  Programmer:   David G. Simpson
2288    !                NASA Goddard Space Flight Center
2289    !                Greenbelt, Maryland  20771
2290    !  Date:         February 6, 2009
2291    !  Language:     Fortran-90
2292    !  Version:      1.00a
2293    !  Description:  Computes the determinant a 4x4 matrix
2294    !*****************************************************************************************
2295
2296          IMPLICIT NONE
2297          real (kind = 8), DIMENSION(4,4), INTENT(IN)  :: A
2298          real (kind = 8) :: DET
2299
2300          DET =  A(1,1)*(A(2,2)*(A(3,3)*A(4,4)-A(3,4)*A(4,3))+A(2,3)*(A(3,4)*A(4,2)-A(3,2)*A(4,4)
2301                 A(3,3)*A(4,2)))-A(1,2)*(A(2,1)*(A(3,3)*A(4,4)-A(3,4)*A(4,3))+A(2,3)*(A(3,4)*A(4,
2302                 A(2,4)*(A(3,1)*A(4,3)-A(3,3)*A(4,1)))+A(1,3)*(A(2,1)*(A(3,2)*A(4,4)-A(3,4)*A(4,2
```

```
2303                A(3,1)*A(4,4))+A(2,4)*(A(3,1)*A(4,2)-A(3,2)*A(4,1)))-A(1,4)*(A(2,1)*(A(3,2)*A(4,
2304                A(2,2)*(A(3,3)*A(4,1)-A(3,1)*A(4,3))+A(2,3)*(A(3,1)*A(4,2)-A(3,2)*A(4,1)))
2305
2306          RETURN
2307
2308    END FUNCTION M44DET
2309
2310
2311
```