```
  1     !PolyGen.f95
  2     !2021.04.06.0610cdt JMS- Traveler2/Athlon64/WinXPPro/APF9.0/Ogl1.2.1
  3     !
  4     !              Up to 8th order and seven variables this program
  5     !                generates N-Variable polynomial coefficients
  6     !                spanning a [-1.,1.]**#ofVariables hyperspace
  7     !             and computes scaled exact integer matrix coefficients
  8     !
  9     !    The user specifies the number & order of each variable in `PolyGen.ini`
 10
 11    !Table of Contents:
 12    !Module PolyGenDef
 13    !Program Main
 14    ! Subroutine ReadPolyGenIni(iP)
 15    ! Subroutine Allocator(iP)
 16    ! Subroutine PowersMatrixGen(iP)
 17    ! Subroutine ComputeCoe(iP)
 18    ! Subroutine MatrixExportPrep( nTot,A,ADiv,AMin,AMax,AMaxScaled &
 19    !                             ,iAout,iADigits,iADeciPl,iP)
 20    ! Subroutine ExportBinaryData(iP)
 21    ! Subroutine Deallocator(iP)
 22    ! Subroutine Invertr16(N,A,ValMin,iRank,DetN,Vused,iP)
 23    ! Subroutine Printr16(N,A,Noise,iRank,DetN,iRu,jCu,iP)
 24    ! Subroutine DisclaimerEtc(iP)
 25    ! Subroutine CoefficientCount(iP)
 26    !-------------------------------------------------------------------------7 9
 27
 28    Module PolyGenDef
 29    !use PolyGenDef                                                  !JMS 2021.03.31
 30    ! -- Polynomial generator data structures
 31    ! -- Specify the multi-variable polynomial:
 32      integer*4,public            ::nVT       !Variables- total number [1,2,...,10]
 33      integer*4,public            ::nOrd(10)  !Order of polynomial of each variable
 34    !                                          1's are forced to be 2's.
 35    !                                          for symmetric & on-center relevance.
 36      integer*4,public            ::nOrdMax   !Largest of the nOrd()'s
 37      integer*4,public            ::nCT       !Coefficients total number
 38      integer*4,public            ::MemoryEst !Real*16 memory estimate:
 39    ! -- Specify the polynomial's domain & output characteristics:
 40      character,public            ::cFileOut*79!Filename of the binary matrices
 41      character*1    ::AugHex(0:20)=(/"0","1","2","3","4","5","6","7","8","9","a" &
 42                                     ,"b","c","d","e","f","g","h","i","j","k"/)
 43
 44      integer*4,public            ::iFileBytes !Binary file size in bytes
 45
 46      integer*4,public            ::iOutFiles  &!Output type:
 47                                               =1!:"Comma separated values" (.csv)
 48    !                                          =2 : Binary                  (.bin)
 49    !                                          =3 : Both .csv & .bin
 50      integer*4,public            ::iPolyClass &!Coefficients type:
 51    !                                          =1 :Exact
 52                                               =2!:Unscaled
 53      integer*4,public            ::iDomain    &
 54                                               =1!:[ -1.  , 1. ]
 55    !                                          =2 :[-N/2. ,N/2.]
 56      integer*4,public            ::iReInSm    &!Real or integer or small arrays
 57                                               =1!:real
```

```
  58    !                                          =2 :integer
  59    !                                          =3 :most compact representation
  60      integer*4,public            ::iOutCoes   &!Binary data type to use
  61    !                                          =0 :By default- none
  62    !                                          =1 :i* 1-   8 bits ~  2 digits
  63    !                                          =2 :i* 2-  16  "   ~  4   "
  64    !                                          =3 :r* 4-  32  "   ~  6   "
  65    !                                          =4 :i* 4-  32  "   ~  9   "
  66    !                                          =5 :r* 8-  64  "   ~ 15   "
  67    !                                          =6 :i* 8-  64  "   ~ 18   "
  68                                               =7!:r*16- 128  "   ~ 31   "
  69      integer*4,public            ::iDecPlaces!Number of decimal places in .csv
  70    !                                          =[0,1,2,3,4,...,31,32]
  71    ! -- Refine the runtime estimate:
  72      real*16                     ::TEC       &!Time Estimate Coefficient
  73                                               = 208408._16
  74    !                                           units: nCT's squared per minute
  75      real*16                     ::TimeEstMin!= nCT*nCT/iToc ~Runtime in minutes
  76    !                                             ...
  77      integer*4,public            ::iPolyAlloc =0!:unAllocated
  78    !                                          =1 :  Allocated
  79    ! --Data samplepoint values:
  80      real*16  ,public,allocatable::X( :,:)   !Coefficient values in [-1,1]
  81    !                           (0:nOrdMax,nVT)
  82
  83    !  --Sequence of matrix values for both rows and columns of both matrices:
  84      integer*4,public,allocatable::xRC(:,:)  !(_,Repeats & Cycles) counts
  85    !                                 (2,nVT)   = product(nOrd(i)+1),i=1,nVT)
  86    !                                          This bypasses the need for recursion!
  87      integer*4,public,allocatable::nVnC(:,:) !Indexes & powers lookup table
  88    !                                  (nVT,nCT)
  89    ! --Dataset powers matrix:
  90      real*16  ,public,allocatable:: Pow(:,:) !The matrix to be inverted
  91    !                                  (nCT,nCT)
  92      real*16  ,public            :: PowDiv   !Divisor of Coe()- real
  93      real*16  ,public            :: PowMin   !MinAbs Pow() coefficient >1.e-20
  94      real*16  ,public            :: PowMax   !MaxAbs coefficient-
  95      real*16  ,public            :: PowMaxScaled !              - scaled
  96      integer*4,public            ::iPowOut   !Binary reporting Compression:
  97    !                                          =0: By default- none
  98    !                                          =1: i* 1-   8 bits ~  2 digits
  99    !                                          =2: i* 2-  16  "   ~  4   "
 100    !                                          =3: r* 4-  32  "   ~  6   "
 101    !                                          =4: i* 4-  32  "   ~  9   "
 102    !                                          =5: r* 8-  64  "   ~ 15   "
 103    !                                          =6: i* 8-  64  "   ~ 18   "
 104    !                                          =7: r*16- 128  "   ~ 31   "
 105      integer*4,public            ::iPowDigits!Number of digits for exact values
 106      integer*4,public            ::iPowDeciPl!Number of decimal places
 107
 108    ! --Polynomial coefficients matrix:
 109      real*16  ,public,allocatable:: Coe(:,:) !The matrix to be inverted
 110    !                                  (nCT,nCT)
 111      real*16  ,public            :: CoeMult  !Additional Coe() multiplier
 112      real*16  ,public            :: CoeDiv   !Divisor of Coe()- real
 113      real*16  ,public            :: CoeMin   !MinAbs Coe() coefficient >1.e-20
 114      real*16  ,public            :: CoeMax   !MaxAbs coefficient-
```

```
  real*16  ,public            :: CoeMaxScaled !              - scaled
  integer*4,public            ::iCoeOut   !Binary reporting Compression:
!                                          =0: By default- none
!                                          =1: i* 1-   8 bits ~  2 digits
!                                          =2: i* 2-  16  "   ~  4   "
!                                          =3: r* 4-  32  "   ~  6   "
!                                          =4: i* 4-  32  "   ~  9   "
!                                          =5: r* 8-  64  "   ~ 15   "
!                                          =6: i* 8-  64  "   ~ 18   "
!                                          =7: r*16- 128  "   ~ 31   "
  integer*4,public            ::iCoeDigits!Number of digits for exact values
  integer*4,public            ::iCoeDeciPl!Number of decimal places

  real*16  ,public,allocatable:: Vused(:) !Matrix inverter reporting.
!                                    (nCT)
  real*16  ,public            :: Frac     !The division fraction accumulator

  real*16  ,public            ::CoePowMax ! AbsMax value in [Coe*Pow-I]

  character,public            ::Chrono(0:9)*22!TimeStamps:
!                                   (0):               for differential use
!                                   (1):               Start
!                                   (2):Data entry    - done
!                                   (3):Powers matrix - done
!                                   (4):Powers matrix - inverted

!                                   (5):Coeffs matrix - done
!                                   (6):Binary file   - exported
!                                   (7):CSV file      - exported
!                                   (8):               Done
!                                   (9):Elapsed runtime

! --The external runtime environment variables:
  character::cIniFIle*79  ='PolyGen.ini'c
  character::cOutFile*79  ='PCoes.txt'c
  integer*4::uD  =14 !: Open(uD...) for reading & writing documentation
  integer*4::uP  =13 !: write(uP,...) Print to- file
  integer*4::uS  = 6 !: write(uS,...) Print to- screen
  integer*4::uT  = 0 ! [0,uP, or uS] temporarily requested outputs
!                     if(uT>5) write(uT,...)
  logical  ::LExists
  integer*4::iAlloc     !Allocation flag: used with allocatable arrays     flag
!Contains
End Module PolyGenDef
!-------------------------------------------------------------------------------7 9
Program Main                                                          !2021.03.31.1710
!2021.03.31.1710cdt JMS- Opens an output file & calls PolyGen()
 use PolyGenDef,only: &
   cIniFile,cOutFile,iAlloc,LExists,uP,uS,Chrono,TimeEstMin,nCT,iOutFiles,TEC &
  ,Coe,CoeDiv,CoeMin,CoeMax,CoeMaxScaled,iCoeOut,iCoeDigits,iCoeDeciPl        &
  ,Pow,PowDiv,PowMin,PowMax,PowMaxScaled,iPowOut,iPowDigits,iPowDeciPl
!--End Globals
 implicit none
 !--Arguments
 !--Internals
  character::cL*80
  !--EndDefs--------------------------
```

```
172      uP = 13  ! = 6 ...for screen only (e.g. for no writes to a hard drive.)
173                                                  call DateAndTime22(Chrono(1))
174      !--- Open output
175      if(uP.gt.12) then
176        open( uP,file=cOutFile,action='write',err=10)
177          cL='Opening output file: '//cOutFile    ;call TimeStamp(cL,chrono(1),uP)
178          goto 20
179      endif !uP>12
180   10     uP=6
181          cL='The computer monitor will be the output. 'c ;call WSF(cL,uP)
182          cL=''c                                          ;call WSF(cL,uP)
183   20 continue
184      cL='This is:    "PolyGen.exe",         By: Jeff Setterholm'c
185                                                          ;call WSF(cL,uP)
186      cL='         Version 1. 2021.04.06    Lakeville, MN 55044 USA'c
187                                                           call WSF(cL,uP)
188      cL=''c                                              ;call WSF(cL,uP)
189      cL=' Output: a large binary file of very accurate matrix coefficients.'c
190                                                           call WSF(cL,uP)
191      cL='       : Up to 8th order the nearest integers are scaled exact values'c
192                                                           call WSF(cL,uP)
193      cL='           & can be written to a .csv file by `Polyize.exe`.'c
194                                                           call WSF(cL,uP)
195     cL='       : 2000+ coefficient polynomials up to 20th order can be computed.'c
196                                                           call WSF(cL,uP)
197      cL=''c                                              ;call WSF(cL,uP)
198      cL='"IntroToPolyGen.txt" - has some background information.'c
199                                                           call WSF(cL,uP)
200      !--- Checking for PolyGen.ini
201      Inquire(file=cIniFIle,exist=LExists)
202        if(LExists)then
203          cL='       "PolyGen.ini" - has your program control inputs.'c
204                                                           call WSF(cL,uP)
205         else; cL='       "PolyGen.ini" - wasn`t found.'c          ;call WSF(cL,uP)
206               cL='          The file is self-documented.'c        ;call WSF(cL,uP)
207          pause"               Press return to exit PolyGen..."          ;goto 100
208        endif !LExists
209      !-- end Inquiry.
210
211   !  call CoefficientCount(uP)
212
213      call ReadPolyGenIni(uP) !/////////////////////
214      call DateAndTime22(Chrono(2)) !Data entry - done
215      write(cL,"('Data entry - done')");call TimeStamp(cL,Chrono(2),uP)
216      call ElapsedTime(Chrono(1),Chrono(2),Chrono(9),TimeEstMin,0)
217      call TimeStamp( & !                                  |
218      "                                    task`s time: "c       ,Chrono(9),uP)
219
220      call Allocator(uP) !//////////////////////////
221
222      call PowersMatrixGen(uP) !////////////////////
223      call MatrixExportPrep( nCT,Pow,PowDiv,PowMin,PowMax,PowmaxScaled &
224                            ,iPowout,iPowDigits,iPowDeciPl,uP) !////////
225
226      call DateAndTime22(Chrono(3)) !Powers matrix - done
227      write(cL,"('Powers matrix - done')")        ;call TimeStamp(cL,Chrono(3),uP)
228      call ElapsedTime(Chrono(2),Chrono(3),Chrono(9),TimeEstMin,0)
```

```
229      call TimeStamp( & !                                          |
230      "                                      task`s time: "c       ,Chrono(9),uP)
231
232      call ComputeCoe(uP) !////////////////////////
233      call DateAndTime22(Chrono(4)) !Powers matrix - inverted
234      write(cL,"('Powers matrix - inverted')")    ;call TimeStamp(cL,Chrono(4),uP)
235      call ElapsedTime(Chrono(3),Chrono(4),Chrono(9),TimeEstMin,0)
236      call TimeStamp( & !                                          |
237      "                                      task`s time: "c       ,Chrono(9),uP)
238
239
240      call MatrixExportPrep( nCT,Coe,CoeDiv,CoeMin,CoeMax,CoeMaxScaled &
241                            ,iCoeout,iCoeDigits,iCoeDeciPl,uP) !///////
242
243      call DateAndTime22(Chrono(5)) !Coeffs matrix - done
244      write(cL,"('Coeffs matrix - done')")        ;call TimeStamp(cL,Chrono(5),uP)
245      call ElapsedTime(Chrono(4),Chrono(5),Chrono(9),TimeEstMin,0)
246      call TimeStamp( & !                                          |
247      "                                      task`s time: "c       ,Chrono(9),uP)
248
249      Chrono(6)=Chrono(5)
250      call ExportBinaryData(uP) !//////////////////
251      call DateAndTime22(Chrono(6)) !Binary file   - exported
252      write(cL,"('Binary file - exported')")    ;call TimeStamp(cL,Chrono(6),uP)
253      call ElapsedTime(Chrono(5),Chrono(6),Chrono(9),TimeEstMin,0)
254      call TimeStamp( & !                                          |
255      "                                      task`s time: "c     ,Chrono(9),uP)
256
257      Chrono(7)=Chrono(6) !No .csv export.
258
259      call Deallocator(uP) !////////////////////////
260      call DateAndTime22(Chrono(8)) !Done
261        write(cL,"('Done.')")                       ;call TimeStamp(cL,Chrono(7),uP)
262        call ElapsedTime(Chrono(7),Chrono(8),Chrono(9),TimeEstMin,0)
263        call TimeStamp( & !                                        |
264      "                                        task`s time: "c     ,Chrono(9),uP)
265      write(cL,"('')")                                ;call WSF(cL,uP)
266      call ElapsedTime(Chrono(1),Chrono(8),Chrono(9),TimeEstMin,uP)
267
268      TEC = nCT*nCT/TimeEstMin
269      write(cL,"('')")                                                ;call WSF(cL,uP)
270      write(cL,"(f15.3,',:TEC- measured matrix coefficients per minute')") TEC
271                                                                       call WSF(cL,uP)
272      write(cL,"('')")                                                ;call WSF(cL,uP)
273      call DisclaimerEtc(uP) !/////////////////////
274      cL=''c                                                          ;call WSF(cL,uP)
275    100 cL='Ending "PolyGen.exe"'c
276      call DateAndTime22(Chrono(9))                  ;call TimeStamp(cL,Chrono,uP)
277      if(uP.gt.12) close(uP)
278      pause 'Press enter to continue - thus closing this output screen & exiting.'
279    End Program Main
280    !-------------------------------------------------------------------------------7 9
281
282    Subroutine ReadPolyGenIni(iP)
283    !2021.03.31.1710cdt JMS- Print out the run setup
284     use PolyGenDef
285    !--End Globals
```

```
  implicit none
  !--Arguments
   integer*4::iP           !Write enable>5: write(iP,...)
!--Internals
   integer*4::N,nV,i,nCh
   character::cL*80,cString*80
   real*16  ::TECin
   !--EndDefs---------------------------
!  cL=''c                                                  ;call WSF(cL,iP)
!  cL='Subroutine ReadPolyGenIni():'c                      ;call WSF(cL,iP)

!  --Default values:
   nVT         = 4             !#1
   nOrd        = 0
   nOrd(1:2)   = (/2,2/)       !#2
   iPolyClass  = 1             !#3 exact
   TEC         = 208408._16    !#4 #of matrix coefficients computed per minute
   iOutFiles   = 2             !#5 .bin
   iDecPlaces  = 0             !#8 no trailing zeros after the decimal point
!  --Read the PolyGen.ini:
   cL='                       ... opening it now...'c          ;call WSF(cL,uP)
   Open(unit=uD,file=cIniFIle,action='read')
10   read(uD,"(a80)") cString
     read(cString(1:1),"(i1)") i
     select case(i)
      case(1); read(cString(3:),*) nVT                          !nVT
                                if(nVT< 1) nVT  =  1
                                if(nVT>10) nVT  = 10
      case(2);                           nOrd =  0            !nOrd(1:nVT)
               read(cString(3:),*) nOrd(1:nVT)

       do nV = 1,nVT ;if(mod(nOrd(nV),2)== 1) nOrd(nV) = nOrd(nV)- 1
                      if(    nOrd(nV)   == 0) nOrd(nV) =           2
                      if(    nOrd(nV)    >20) nOrd(nV) =          20
       enddo!nV
      case(3); read(cString(3:),*) iPolyClass                   !iPolyClass
                                if(iPolyClass.ne.2) iPolyClass = 1
      case(4); read(cString(3:),*,err=10) TECin                 !TEC
                                if(TECin<  1000._16) TEC = 200000._16
                                if(TECin>= 1000._16) Tec = TecIn
      case(0);  goto 20!                        ///Ends the .ini file read///
     end select!(i)
     goto 10
20 close(uD)

!  --Find the max order, count coefficients, & precompute the exact divisor:
   nOrdMax = maxval(nOrd); if(nOrdMax>8) iPolyClass = 2 !Unscaled
                  PowDiv = 1._16;           CoeDiv = 1._16;  nCt =  1
   do nV = 1,nVT;                                        nCT = nCT*(nOrd(nV)+1)
     if(iPolyClass==1) then
       select case(nOrd(nV))
        case(2); PowDiv = PowDiv*    1._16; CoeDiv = CoeDiv*  2._16
        case(4); PowDiv = PowDiv*   16._16; CoeDiv = CoeDiv*  6._16
        case(6); PowDiv = PowDiv*  729._16; CoeDiv = CoeDiv* 80._16
        case(8); PowDiv = PowDiv*65536._16; CoeDiv = CoeDiv*630._16
       end select!nOrd(nV)
     endif!(iPolyClass==1)
```

```
343      enddo!nV
344
345   !  --Name the binary output file:
346      cL=char(0)
347      if(iPolyClass==1) then; write(cL(1:7),"('PCoesE',i1,\)")nVT; nCh = 7
348                        else; write(cL(1:7),"('PCoesU',i1,\)")nVT; nCh = 7
349      endif!(iPolyClass==1)
350        do nV=nVT,1,-1;write(cL(nCh+1:nCh+1),"(a1,\)")AugHex(nOrd(nV))
351                                                          nCh = nCh+1;enddo!nV
352                       write(cL(nCh+1:nCh+4),"('.bin')")
353                  cFileOut = cL(    1:   79)
354
355   !  --Report the run configuration:
356      cL='!--- Run configuration:'                                ;call WSF(cL,iP)
357      cL='Values  #'c                                             ;call WSF(cL,iP)
358      write(cL,"( i6,',:1 nVT             -independent variables')") nVT
359                                                                   call WSF(cL,iP)
360      write(cL,"('      ,:2`order` =power of each variable- evens only')")
361                                                                   call WSF(cL,iP)
362      do N = 1,nVT; write(cL,"(i6,',\)") nOrd(n); call WSF(cL,iP)        ;enddo!N
363   !    cL=''                                                     ;call WSF(cL,iP)
364      write(cL,"( i6,',:3 iPolyClass       - 1:exact, 2:unscaled'   )") iPolyClass
365                                                                   call WSF(cL,iP)
366      write(cL,"('      ,:4 TEC             - Time Estimate Coefficient')")
367                                                                   call WSF(cL,iP)
368      write(cL,"('=',f13.1)")TEC                                  ;call WSF(cL,iP)
369      cL='!--- Assigned internally:'c                             ;call WSF(cL,iP)
370      write(cL,"( i6,',:5 iOutFiles Output- 1:.csv, 2:.bin, 3:both')")iOutFiles
371                                                                   call WSF(cL,iP)
372      write(cL,"( 6x,',:6 cFileOut        - `PolyGen`+hex+`.bin`'\)")
373                                                                   call WSF(cL,iP)
374                                                                   call WSF(cL,iP)
375      write(cL,"('=',28x,79(a1,:))") (cFileOut(i:i),i=1,len_trim(cFileOut))
376                                                                   call WSF(cL,iP)
377      write(cL, &
378      "(22x,'order indices ^ are augmented hexadecimals:{ 0,15}->{0:f}')")
379                                                                   call WSF(cL,iP)
380      write(cL,"(39x,'-&- 16->g 17->h 18->i 19->j   20 ->   k')")
381                                                                   call WSF(cL,iP)
382      cL='!--- Overview:'c                                        ;call WSF(cL,iP)
383
384      write(cL,"(i13,', :Highest `order`' )") nOrdMax             ;call WSF(cL,iP)
385      write(cL,"(i13,', :# of variables')") nVT                   ;call WSF(cL,iP)
386      do nV=1,nVT; write(cL,"(i13,', :Order of X(',i2,')')") nOrd(nV),nV
387                                                          call WSF(cL,iP) ;enddo!nV
388      write(cL,"(i13,', :# of  Poly  Coeff`s')") nCT              ;call WSF(cL,iP)
389      write(cL,"(i13,', :# of matrix Coeff`s')") nCT*nCT          ;call WSF(cL,iP)
390      write(cL,"(f17.3,',  estimated runtime (minutes)')") nCT*nCT/TEC
391                                                                   call WSF(cL,iP)
392   !  pause"in ReadPolyGenIni"
393                                                                        return
394   End Subroutine ReadPolyGenIni
395
396   !-------------------------------------------------------------------------7 9
397
398   Subroutine Allocator(iP)
399   !2021.03.24.0815cdt JMS- Deallocates PolyGen arrays.
```

```
!                      - Traveler2/Athlon64/WinXPPro/APF9.0/Ogl1.2.1
 use PolyGenDef
!--End Globals
 implicit none
 !--Arguments
  integer*4::iP           !Write enable>5: write(iP,...)
 !--Internals
  character::cL*80
  !--EndDefs-----------------
!  --Deallocate any allocated arrays:
   if(iPolyAlloc.ne.0) call Deallocator(iP)
!  --Allocate the arrays:
   allocate(X(0:nOrdMax,nVT),stat=iAlloc);         if(iAlloc /= 0) then
       pause 'X(0:nOrdMax,nVT) alloc. error. `Enter` to exit.'    ;stop; endif
   X        = 0._16                                  ;iPolyAlloc = iPolyAlloc+1
   allocate(xRC(2,nVT),stat=iAlloc);               if(iAlloc /= 0) then
       pause 'XRC(2,nVT) alloc. error. `Enter` to exit.'          ;stop; endif
   XRC      = 0                                      ;iPolyAlloc = iPolyAlloc+1
   allocate(nVnC(nVT,nCT),stat=iAlloc);            if(iAlloc /= 0) then
       pause 'nVnC(nVT,nCT) alloc. error. `Enter` to exit.'       ;stop; endif
   nVnC      = 0                                    ;iPolyAlloc = iPolyAlloc+1
   allocate(Pow(nCT,nCT),stat=iAlloc);             if(iAlloc /= 0) then
       pause 'Pow(nCT,nCT) alloc. error. `Enter` to exit.'        ;stop; endif
   Pow       = 1._16                                ;iPolyAlloc = iPolyAlloc+1
   allocate(Coe(nCT,nCT),stat=iAlloc);             if(iAlloc /= 0) then
       pause 'Coe(nCT,nCT) alloc. error. `Enter` to exit.'        ;stop; endif
   Coe      = 1._16                                  ;iPolyAlloc = iPolyAlloc+1
   allocate(Vused(nCT),stat=iAlloc);               if(iAlloc /= 0) then
       pause 'Vused(nCT) alloc. error. `Enter` to exit.'   ;stop; endif
   Vused    = 0._16                                  ;iPolyAlloc = iPolyAlloc+1
   write(cL,"('All arrays needed have been successfully allocated')")
                                                        call WSF(cL,iP)
   return
End Subroutine Allocator
!-------------------------------------------------------------------------------7 9

Subroutine PowersMatrixGen(iP)
!2021.03.24.0815cdt JMS- Populates PolyGenDef
 use PolyGenDef
!--End Globals
 implicit none
 !--Arguments
  integer*4::iP           !Write enable>5: write(iP,...)
!--Internals
  integer*4::nV,nC,nCTlocal,N,nCyc,nRep,nRow,nCol
  real*16  ::n16,PowUse
  character::cL*80
  !--EndDefs---------------------------
     write(cL,"('')")                              ;call WSF(cL,iP)
     write(cL,"('Subroutine PowersMatrixGen():')")
                                                        call WSF(cL,iP)
!  --Compute the sample points of the variables:
   do nV  = 1,nVT;                    n16 = quad(nOrd(nV))
     do N = 0,nOrd(nV)
       if(iDomain==1) &
       X(N,nV) = ((2._16*quad(N) -n16)/n16  )  ![ -1. ,+1. ] //////////
       if(iDomain==2) &
```

```
457         X(N,nV) = ((2._16*quad(N) -n16)/2._16)  ![ -N/2,+N/2] //////////
458       enddo!N
459     enddo!nV
460                   nCTlocal = 1
461     do nV = 1,nVT
462       xRC(1,nV) = nCTlocal                          !# of repeats
463       nCTlocal  = nCTlocal*(nOrd(nV)+1)
464     enddo!nV
465     do nV = 1,nVT
466       xRC(2,nV) = nCTlocal/(xRC(1,nV)*(nOrd(nV)+1)) !# of cycles of the repeat
467     enddo!nV
468
469   !  --Generate the indices & powers lookup table:
470     do nV               = 1,nVT       ; nC = 0
471       do nCyc           = 1,xRC(2,nV)              !Number of repeats
472         do N            = 0,nOrd(nV)
473           do nRep       = 1,xRC(1,nV) ; nC = nC +1 !Number of repetitions
474             nVnC(nV,nC) = N                        !Indices = Powers
475           enddo!nRep
476         enddo!N
477       enddo!nCyc
478     enddo!nV
479
480   !  --Compute the Powers Matrix:
481                          call Beamer( 0  ,nCT,6)
482     do nRow   = 1,nCT   ;call Beamer(nRow,nCT,6)
483       do nCol = 1,nCT
484         do nV=1,nVT
485           PowUse=X(nVnC(nV,nRow),nV)**nVnC(nV,nCol)
486           if(nVnC(nV,nCol)==0) PowUse = 1._16
487           Pow(nrow,nCol)=Pow(nrow,nCol)*PowUse
488         enddo!nV
489       enddo!nCol
490     enddo!nRow
491                                                                          return
492   End Subroutine PowersMatrixGen
493   !-------------------------------------------------------------------------7 9
494   Subroutine ComputeCoe(iP)
495   !2021.03.24.0815cdt JMS- Populates PolyGenDef
496    use PolyGenDef
497   !--End Globals
498    implicit none
499    !--Arguments
500     integer*4::iP           !Write enable>5: write(iP,...)
501   !--Internals
502     integer*4::nC,nRow,nCol
503   ! Inverter arguments:
504     real*16  :: ValMin   = 0.d0
505     integer*4:: iRank    = 0
506     real*16  :: DetN    !Abs(determinant product) for each iteration
507     real*16  :: CoePow
508     character::cL*80
509
510     !--EndDefs---------------------------
511   !  --Computing the coefficient matrix by inversion of Pow():
512      Coe=Pow
513
```

```
514      write(cL,"('')")                                  ;call WSF(cL,iP)
515      write(cL,"('Coe[,] = inverse(Pow[,]), inversion progress bar...')")
516                                                        call WSF(cL,iP)
517
518      call Invertr16(nCT,Coe,ValMin,iRank,DetN,Vused,0)  !iP!/////////////////////
519      write(cL,"(e54.30,', :Determinant' )") DetN     ;call WSF(cL,iP)
520      write(cL,"(f50.30,','              )") DetN     ;call WSF(cL,iP)
521
522      if(iRank.ne.nCT) &
523      write(cL,"('Pseudoinverse result. Missed ',i6,' dimensions')") nCT-iRank
524      if(iRank.ne.nCT)                                  call WSF(cL,iP)
525
526      call DateAndTime22(Chrono(9)) !Inversion - done
527      write(cL,"('')")                                 ;call TimeStamp(cL,Chrono(9),uP)
528
529   !  --Confirm inversion accuracy:
530      write(cL,"('Evaluating the inversion accuracy...')")
531                                                        call WSF(cL,iP)
532      CoePowMax=0._16                                  ;call Beamer(0   ,nCT,6)
533      do nRow=1,nCT                                    ;call Beamer(nRow,nCT,6)
534        do nCol=1,nCT; CoePow=0._16
535          do nC=1,nCT; CoePow=CoePow+Coe(nRow,nC)*Pow(nC,nCol)         ;enddo!nC
536        if(nRow==nCol) CoePow=CoePow-1._16
537        if(abs(CoePow)>CoePowMax) CoePowMax=abs(CoePow)          ;enddo;enddo!nRow
538      write(cL,"(e12.3,', :AbsMax r*16 inversion error in [Coe*Pow-I]')") &
539                                  CoePowMax             ;call WSF(cL,iP)
540                                                                      return
541   End Subroutine ComputeCoe
542   !-------------------------------------------------------------------------------7 9
543
544   Subroutine MatrixExportPrep( nTot,A,ADiv,AMin,AMax,AMaxScaled &
545                               ,iAout,iADigits,iADeciPl,iP)
546   !2021.04.01.0835cdt JMS- Preparing the Coe() matrix for export
547    use PolyGenDef     , only:iPolyClass,iDecPlaces
548   !--End Globals
549    implicit none
550    !--Arguments
551     integer*4::nTot
552     real*16  ::A(nTot,nTot) !Pow(,) or Coe(,)
553     real*16  ::ADiv
554     real*16  ::AMin
555     real*16  ::AMax         !MaxAbs coefficient-
556     real*16  ::AMaxScaled   !              - scaled
557     integer*4::iAOut        !Binary reporting Compression: Not yet implemented.
558   !                         =0: By default- none
559   !                         =1: i* 1-   8 bits ~  2 digits
560   !                         =2: i* 2-  16  "   ~  4   "
561   !                         =3: r* 4-  32  "   ~  6   "
562   !                         =4: i* 4-  32  "   ~  9   "
563   !                         =5: r* 8-  64  "   ~ 15   "
564   !                         =6: i* 8-  64  "   ~ 18   "
565   !                         =7: r*16- 128  "   ~ 31   "
566     integer*4::iADigits     !Number of digits for exact values
567     integer*4::iADeciPl     !Number of decimal places
568     integer*4::iP           !Write enable>5: write(iP,...)
569   !--Internals
570     integer*4::nRow,nCol
```

```
  real*16  ::AUse
  character::cL*80
  !--EndDefs---------------------------
   write(cL,"('!--- MatrixExportPrep():')");       ;call WSF(cL,iP)

   write(cL,"(e54.30,', Division factor')") ADiv   ;call WSF(cL,iP)
   write(cL,"(f50.30,','                )") ADiv   ;call WSF(cL,iP)
   write(cL,"('')")                                ;call WSF(cL,iP)

   AMax = maxval(abs(A))
   write(cL,"(e54.30,', matrix MaxAbs')") AMax     ;call WSF(cL,iP)
   write(cL,"(f50.30,','               )") AMax     ;call WSF(cL,iP)
                     AMin = 1._16
   do nRow   = 1,nTot
     do nCol = 1,nTot
       AUse = abs(A(nRow,nCol))
       if(AUse<1.e-20_16) cycle
       if(AUse<AMin) AMin=AUse
     enddo!nCol
   enddo!nRow
   write(cL,"('')")                                ;call WSF(cL,iP)
   write(cL,"(e54.30,', matrix MinAbs')") AMin     ;call WSF(cL,iP)
   write(cL,"(f50.30,','               )") AMin     ;call WSF(cL,iP)
   write(cL,"('')")                                ;call WSF(cL,iP)

!  --Exporting the coefficient matrix:
   iADeciPl=0
   if(iPolyClass==2) then
     write(cL,"('Unscaled matrix coefficients are needed.')")
                                                          call WSF(cL,iP)
     if(iDecPlaces>0) then; iADeciPl = iDecPlaces
      else
       write(cL,"('  0 decimal place results were also requested.')")
                                                          call WSF(cL,iP)
       write(cL,"('  Your .ini file input #8 can override this guess:')")
                                                          call WSF(cL,iP)

       iADeciPl = int(log10(1._16/AMin))+3
       if(iADeciPl>24) iADeciPl = 24
       write(cL,"( i19,', :decimal places used')") iADeciPl
                                                          call WSF(cL,iP)
     endif!(iDecPlaces>0)
   endif!(iPolyClass==2)

   AMaxScaled = ADiv*AMax
   write(cL,"(f36.3,', :Matrix - biggest integer- decimal')") AMaxScaled
                                                          call WSF(cL,iP)
   if(     AMaxScaled>100000000000000._16) then; iAOut = 16
    elseif(AMaxScaled>       1000000._16) then; iAOut =  8
    else                                      ; iAOut =  4
   endif!AMaxScaled>1.e15_16

   write(cL,"(f36.3,,', :storage space needed- i*1,i*2,i*4,i*8')") &
               AMaxScaled                                ;call WSF(cL,iP)
   write(cL,"('21098765432109876543210987654321, :digits index')")
                                                          call WSF(cL,iP)
   write(cL,"('  3         2         1         ')");call WSF(cL,iP)
```

```
628      iADigits = int(log10(AMaxScaled))+3
629      write(cL,"( i19,', : digits recommended w/`.`')") iADigits  ;call WSF(cL,iP)
630
631    !  if(iPolyClass==1) then  !Multiply A*ADiv during binary write
632    !   !Exactly rescale the Coe Matrix:
633    !    write(cL,"(8x,f50.30,',Matrix division factor'/)") ADiv   ;call WSF(cL,iP)
634    !    do nRow   = 1,nTot
635    !      do nCol = 1,nTot
636    !        AUse = A(nRow,nCol)*ADiv
637    !       !write(cL,"(2i4,f50.30)") nRow,nCol,AUse              ;call WSF(cL,iP)
638    !        if(AUse<1.e-5_16) then; A(nRow,nCol) = 0._16
639    !                          else; A(nRow,nCol) = floor(AUse+.5_16)
640    !        endif!(AUse<1.e-5_16)
641    !       !write(cL,"(8x,f50.30)")            AUse              ;call WSF(cL,iP)
642    !       !cL=""c                                               ;call WSF(cL,iP)
643    !      enddo!nCol
644    !    enddo!nRow
645    !  endif!iPolyClass==1)
646                                                                          return
647    End Subroutine MatrixExportPrep
648    !-------------------------------------------------------------------------------7 9
649
650    Subroutine ExportBinaryData(iP)
651    !2021.04.01.0725cdt JMS- Exports the Coe() matrix & supporting variables
652     use PolyGenDef
653    !--End Globals
654     implicit none
655     !--Arguments
656      integer*4::iP           !Write enable>5: write(iP,...)
657    !--Internals
658      integer*4::nB
659      character::cL*80
660
661      !--EndDefs---------------------------
662       write(cL,"('')")                                 ;call WSF(cL,iP)
663       write(cL,"('Subroutine ExportBinaryData():')")
664
665     !  --Export the Binary matrices
666       write(cL,*) cFileOut                                    ;call WSF(cL,iP)
667                                                                    write(cL, &
668       "('Binary file data write/read sequence- integer*4`s & real*16`s:')")
669                                                               call WSF(cL,iP)
670       nB = sizeof(nVT);     iFileBytes =            nB         ;write(  cL,  &
671       "(i12,', i* 4 nVT       -Number of independent variables  fixed size' )") nB
672                                                               call WSF(cL,iP)
673       nB = sizeof(nOrdMax); iFileBytes = iFileBytes+nB         ;write(  cL,  &
674       "(i12,', i* 4 nOrdMax   -Highest order variable             `    `'   )") nB
675                                                               call WSF(cL,iP)
676       nB = sizeof(nCT);     iFileBytes = iFileBytes+nB         ;write(  cL,  &
677       "(i12,', i* 4 nCT       -Number of polynomial coefficients  `    `'   )") nB
678                                                               call WSF(cL,iP)
679       nB = sizeof(nOrd);     iFileBytes = iFileBytes+nB        ;write(  cL,  &
680       "(i12,', i* 4 nOrd(20)  -Order of each variable             `    `'   )") nB
681                                                               call WSF(cL,iP)
682       nB = sizeof(iPolyClass);iFileBytes = iFileBytes+nB        ;write(  cL,  &
683       "(i12,', i* 4 iPolyClass-=1:exact, =2:unscaled              `    `'   )") nB
684                                                               call WSF(cL,iP)
```

```
685      nB = sizeof(iCoeDigits);iFileBytes = iFileBytes+nB       ;write(  cL,  &
686      "(i12,', i* 4 iCoeDigits- +***. <-count                     `    `'   )") nB
687                                                          call WSF(cL,iP)
688      nB = sizeof(iCoeDeciPl);iFileBytes = iFileBytes+nB       ;write(  cL,  &
689      "(i12,', i* 4 iCoeDeciPl-number of digits after .           `    `'   )") nB
690                                                          call WSF(cL,iP)
691      nB = sizeof(CoeDiv);   iFileBytes = iFileBytes+nB        ;write(  cL,  &
692      "(i12,', r*16 CoeDiv           -Coe[,] dividing factor      `    `'   )") nB
693                                                          call WSF(cL,iP)
694      nB = sizeof(CoeMaxScaled); iFileBytes = iFileBytes+nB    ;write(  cL,  &
695      "(i12,', r*16 CoeMaxScaled     -AbsMax Coe[,]               `    `'   )") nB
696                                                          call WSF(cL,iP)
697      nB = sizeof(iPowDigits);iFileBytes = iFileBytes+nB       ;write(  cL,  &
698      "(i12,', i* 4 iPowDigits- +***. <-count                     `    `'   )") nB
699                                                          call WSF(cL,iP)
700      nB = sizeof(iPowDeciPl);iFileBytes = iFileBytes+nB       ;write(  cL,  &
701      "(i12,', i* 4 iPowDeciPl-number of digits after .           `    `'   )") nB
702                                                          call WSF(cL,iP)
703      nB = sizeof(PowDiv);   iFileBytes = iFileBytes+nB        ;write(  cL,  &
704      "(i12,', r*16 PowDiv           -Pow[,] dividing factor      `    `'   )") nB
705                                                          call WSF(cL,iP)
706      nB = sizeof(PowMaxScaled); iFileBytes = iFileBytes+nB    ;write(  cL,  &
707      "(i12,', r*16 PowMaxScaled     -AbsMax Pow[,]               `    `'   )") nB
708                                                          call WSF(cL,iP)
709      nB = sizeof(X);        iFileBytes = iFileBytes+nB        ;write(  cL,  &
710      "(i12,', r*16 X[0:nOrdMax,nVT] -Variable values to use    allocated'  )") nB
711                                                          call WSF(cL,iP)
712      nB = sizeof(nVnC);     iFileBytes = iFileBytes+nB        ;write(  cL,  &
713      "(i12,', i* 4 nVnC[nVT   ,nCT] -Values combinations sequence  `'      )") nB
714                                                          call WSF(cL,iP)
715      nB = sizeof(Coe);      iFileBytes = iFileBytes+nB        ;write(  cL,  &
716      "(i12,', r*16 Coe[ nCT   ,nCT] -The coefficients solver       `'      )") nB
717                                                          call WSF(cL,iP)
718      nB = sizeof(Pow);      iFileBytes = iFileBytes+nB        ;write(  cL,  &
719      "(i12,', r*16 Pow[ nCT   ,nCT]      Coe[,] = inverse(Pow[,])  `'      )") nB
720                                                          call WSF(cL,iP)
721       write(  cL,"(i12,', Total Bytes')") iFileBytes;  call WSF(cL,iP)
722     write(cL,"(f15.3,12x,':write-time-minutes ballpark')")iFileBytes/1.e10_16
723                                                          call WSF(cL,iP)
724       open(unit=14,file=cFileOut,form='binary',action='write')
725         write(14)  nVT, nOrdMax, nCT,nOrd, iPolyClass            &
726                  , iCoeDigits, iCoeDeciPl,CoeDiv, CoeMaxScaled  &
727                  , iPowDigits, iPowDeciPl,PowDiv, PowMaxScaled  &
728                  , X, nVnC, (Coe*CoeDiv), (Pow*PowDiv)      !<- Verify this!
729       close(14)
730                                                                      return
731    End Subroutine ExportBinaryData
732    !-------------------------------------------------------------------------7 9
733
734    Subroutine Invertr16(N,A,ValMin,iRank,DetN,Vused,iP)          !2021.03.22.0535
735    !2021.03.22.0535cdt JMS- Vused is an array returning the division sequence.
736    !2010.01.06.1205cst JMS- Traveler2/Athlon64/WinXPPro-32/APF9.0-32
737    !2006.05.25.0715cdt JMS- An instantiation of the "Setterholm Matrix Inverter"
738    !                        Expects,uses, & returns real*16's (~QUAD precision).
739    !---                                                             No globals
740     implicit none                                                  !arguments
741      integer*4:: N
```

```
  742     real*16  :: A(N,N)
  743     real*16  :: ValMin
  744     integer*4:: iRank
  745     real*16  :: DetN    !Abs(determinant product) for each iteration
  746   ! integer*4:: iUsed(N)
  747     real*16  :: Vused(N)                                          !2021.03.14
  748     integer*4:: iP
  749     !---                                                          !internals
  750     integer*4,allocatable::iRu(:)
  751     integer*4,allocatable::jCu(:)
  752     real*16  ,allocatable::Temp(:)
  753     integer*4::i,iu,iu2,j,ju,ju2,L
  754     real*16  ::Amult,ValMax
  755     integer*8::iBinSum
  756     integer*4::iAlloc
  757      !-----                                                        !end defs
  758      if(iP.gt.5) write(iP,"('Overwriting Matrix Inversion:')")
  759      DetN=0._16 ;Vused(1:N)=0._16 ;iRank=0
  760      !---
  761      allocate(iRu(N),stat=iAlloc)
  762      if(iAlloc.ne.0) stop 'Invert: iRu(N) allocation error. Halt.'
  763      iRu=0
  764      allocate(jCu(N),stat=iAlloc)
  765      if(iAlloc.ne.0) stop 'Invert: jCu(N) allocation error. Halt.'
  766      jCu=0
  767      allocate(Temp(N),stat=iAlloc)
  768      if(iAlloc.ne.0) stop 'Invert: Temp(N) allocation error. Halt.'
  769      Temp=0._16
  770      !---
  771        Temp=0._16; iRu   =0 ;jCu   =0 ;ValMin=0._16
  772      do i=1,N    ;iRu(i)=i ;jCu(i)=i                ;end do
  773              if(iP.gt.5) call Printr16(N,A,ValMin,iRank,DetN,iRu,jCu,iP) !*******
  774   !  Inversion Loop: ---------------------------------------
  775                                                call Beamer(0,N,6)
  776      do L=1,N ;iU=0     ;jU=0     ;ValMax =0._16 ;call Beamer(L,N,6)
  777        do i=1,N  ;if(iRu(i).lt.0) cycle
  778          do j=1,N ;if(jCu(j).lt.0) cycle
  779            if(ValMax.ge.abs(A(i,j))) cycle
  780              ValMax=abs(A(i,j)) ;iu=i ;ju=j
  781          enddo!j
  782        enddo!i
  783        if(L==1) ValMin=ValMax*1.e-20_16!Establishes the "noise floor".
  784        if(L==1) DetN=ValMax
  785        if(iU.eq.0) Exit
  786        if(ValMax.lt.ValMin) Exit
  787        iRank=iRank+1
  788        if(L>1) DetN=DetN*ValMax
  789        ValMax=A(iu,ju) ;iu2=jCu(ju) ;ju2=iRu(iu)
  790       !if(iP.gt.5) write(iP,"(3i3,f15.6)")  L,iu,ju,ValMax
  791        if(iP.gt.5) write(iP,"(1x,2i4,f15.9,'   :iu ju pivot, & value')") &
  792                                                 iu,ju,        ValMax
  793        Vused(L)=ValMax                                           !2021.03.14
  794
  795        Temp(1:N)=A(1:N,ju)                          ;i=iRu(iu)
  796                  A(1:N,ju)=A(1:N,ju2)               ;  iRu(iu)=iRu(iu2)
  797                            A(1:N,ju2)=Temp(1:N)     ;          iRu(iu2)=-abs(i)
  798        Temp(1:N)=A(iu,1:N)                          ;j=jCu(ju)
```

```
799                 A(iu,1:N)=A(iu2,1:N)                 ;  jCu(ju)=jCu(ju2)
800                           A(iu2,1:N)=Temp(1:N)/ValMax ;         jCu(ju2)=-abs(j)
801       do i=1,N ;if(i.eq.iu2) cycle ;Amult=A(i,ju2)
802         do j=1,N ;A(i,j)=A(i,j)-A(iu2,j)*Amult ;enddo!j
803         A(i,ju2)=-Amult/ValMax
804       enddo; A(iu2,ju2)=1._16/ValMax
805            if(iP.gt.5) call Printr16(N,A,ValMin,iRank,DetN,iRu,jCu,iP) !*******
806     enddo!L  Inversion loop- done. -------------------------
807
808     iBinSum=0
809     do i=1,n
810       if((iRu(i).lt.0).and.(i.lt.63)) iBinsum=iBinSum+2**(i-1)
811  !!!! if(iRu(i).lt.0) iUsed(i)=1                                    !2021.03.14
812  !    Zero linearly-dependent rows and columns, if any:
813       if(iRu(i).gt.0) A(i,1:N)=0._16
814       if(jCu(i).gt.0) A(1:N,i)=0._16
815     enddo!i
816     if((iP >5).and.(iu == 0)) then
817       write(iP,"('Linear dependency dealt with. Partial inverse results:')")
818       write(iP,"(b63.2,' -Kp unused flag')") iBinSum !For the first 62 Kp's
819                         call Printr16(N,A,ValMin,iRank,DetN,iRu,jCu,iP) !*******
820     endif!iP>5 & iu==0
821     if(iP.gt.5) write(iP,"('Overwriting Inverter- done.')")
822     deallocate(iRu)
823     deallocate(jCu)
824     deallocate(Temp)
825     return
826   End Subroutine Invertr16
827   !-------------------------------------------------------------------------------7-9
828
829   Subroutine Printr16(N,A,Noise,iRank,DetN,iRu,jCu,iP)
830   !2018.09.04.1340cdt JMS- Prints Invert's progress.
831   !                        Expects,uses, & returns real*16's (~QUAD precision).
832   !                     - Traveler2/Athlon64/WinXPPro-32/APF9.0-32
833    !---                                                          No globals
834    implicit none                                               !arguments
835     integer*4:: N
836     real*16  :: A(N,N)
837     real*16  :: Noise
838     integer*4:: iRank
839     real*16  :: DetN
840     integer*4:: iRu(N)
841     integer*4:: jCu(N)
842     integer*4:: iP
843     !---                                                       !internals
844     integer*4::i,j
845      !-----                                                    !end defs
846                                                           if(iP < 6) return
847     write(iP,"('Inverter: @ Rank =',i5,8x,'abs(Det)=',e39.30/2x\)") iRank,DetN
848     do j=1,N;   write(iP,"(i18,\)") jCu(j) ; enddo; write(iP,"('')")
849     do i=1,N;   write(iP,"(i5,' ' ,\)") iRu(i)
850       do j=1,N; write(iP,"(sp,f18.9\)") A(i,j); enddo; write(iP,"('')")
851     enddo!i
852     write(iP,"('')");                                                return
853   End Subroutine Printr16
854   !-------------------------------------------------------------------------------7-9
855
```

```
  Subroutine Deallocator(iP)
  !2021.03.24.0815cdt JMS- Deallocates PolyGen arrays.
  !                      - Traveler2/Athlon64/WinXPPro/APF9.0/Ogl1.2.1
   use PolyGenDef
  !--End Globals
   implicit none
   !--Arguments
    integer*4::iP           !Write enable>5: write(iP,...)
   !--Internals
    !--EndDefs-----------------
     if(iPolyAlloc>0) then
       deallocate(X)
       deallocate(xRC)
       deallocate(nVnC)
       deallocate(Pow)
       deallocate(Coe)
       deallocate(Vused)
       iPolyAlloc = 0
       if(iP>5) write(iP,"(1x,'Arrays deallocated')")
     endif!iPolyAlloc>0
     return
   End Subroutine Deallocator
  !-------------------------------------------------------------------------------7 9

  Subroutine DisclaimerEtc(iP)
  !2021.04.06.0610cdt JMS- Disclaimer, etc.
  !Elapsed is updated first,
  !       so using the same argument twice results in returning the DaTime value.
  !--Arguments
   implicit none
    integer*4::iP           !Write enable>5: write(iP,...)
   !--Internals
    character::cL*80
    !--EndDefs-----------------------------------------------
   cL=""                                                        ;call WSF(cL,iP)
   cL="   Link: http://ftp.setterholm.com/ExactInversePolynomials/PolyGen.zip"c
                                                                 call WSF(cL,iP)
   cL="Analyst: Jeff Setterholm  Lakeville MN 55044 USA    Tuesday 2020.04.06"c
                                                                 call WSF(cL,iP)
   cL=""                                                        ;call WSF(cL,iP)
   cL="  ********, PolyGen.exe is post-copyright i.e.:FREE ,********"c
                                                                 call WSF(cL,iP)
   cL=""                                                        ;call WSF(cL,iP)
   cL="  My legal disclaimer:"c                                 ;call WSF(cL,iP)
   cL="  ************************************************************"c
                                                                 call WSF(cL,iP)
   cL="  ********,  Individual cognition is always flawed  ,********"c
                                                                 call WSF(cL,iP)
   cL="  ********,          including yours & mine.        ,********"c
                                                                 call WSF(cL,iP)
   cL="  ********,                 - So: -                 ,********"c
                                                                 call WSF(cL,iP)
   cL="  ********,    Use these results at your own risk.  ,********"c
                                                                 call WSF(cL,iP)
   cL="  ************************************************************"c
                                                                 call WSF(cL,iP)
   cL="           Mitigate malfunctions. Nurture synergies."c
```

```
913                                                                     call WSF(cL,iP)
914                                                                          return
915   End Subroutine DisclaimerEtc
916   !-------------------------------------------------------------------------------7 9
917   Subroutine CoefficientCount(iP)
918   !2021.03.27.1035cdt JMS- # of coeff`s of even-order multi-variable Polynomials
919   !                                       up to 20th order.
920    implicit none
921    !--Arguments
922     integer*4 ::iP           !Write enable>5: write(iP,...)
923    !--Internals
924     integer*4 ::nMax=7,j,nOrder(10)=1,nCh,iUsed,nLine
925     integer*4 ::i1,i2,i3,i4,i5,i6,i7
926     integer*4 ::j1,j2,j3,j4,j5,j6,j7
927     character::cL*100
928     !--EndDefs-------------------------------------------------
929                                                                     if(iP<6) return
930      cL=''c ;write(iP,*) CL
931      cL= &
932      '    Number of polynomial coefficients <= 9999'// &
933      '  for 7 variables up to order 20'c
934      write(iP,*) CL
935      cL= &
936      '            Order of v1 = 0    2    4    6    8|'// &
937      '  10   12   14   16   18   20'c
938      write(iP,*) CL
939      cL= &
940      '                       ----------------- <-exact|unscaled->'// &
941      ' ------------------'c
942      write(iP,*) CL
943                                  nLine=1
944      do i1=0,20,2                 ; j1 = i1+1;
945        do i2=i1,20,2              ; j2 = i2+1;
946          do i3=i2,20,2            ; j3 = i3+1;
947            do i4=i3,20,2          ; j4 = i4+1;
948              do i5=i4,20,2        ; j5 = i5+1;
949                do i6=i5,20,2      ; j6 = i6+1;
950                                    j  = j1*j2*j3*j4*j5*j6
951                                  if(J>9999) exit
952                    cL=char(0);  iUsed = 0;           nCh = 22
953                    write(cL(1:22),"(i3,1x,6i3.0\)") nLine,i1,i2,i3,i4,i5,i6
954                                    if(nLine==  1) cL(17:22) = 'V2 = 0'
955                                    if(nLine== 11) cL(14:19) = 'V3 = 0'
956                                    if(nLine== 66) cL(11:16) = 'V4 = 0'
957                                    if(nLine==132) cL( 8:13) = 'V5 = 0'
958                                    if(nLine==161) cL( 5:10) = 'V6 = 0'
959                                    if(nLine==171) cL( 4: 7) = 'V7=0'
960                  do i7=2,i6,2;              write(cL(nCh+1:nCh+5),"('     '\)")
961                                                    nCh = nCh+5        ;enddo!i7
962                  do i7=i6,20,2;  j7 = i7+1;
963                                     j  = j1*j2*j3*j4*j5*j6*j7
964                                  if(J>9999) exit
965                                  iUsed = iUsed+1
966                                            write(cL(nCh+1:nCh+5),"(i5\)") j
967                                                    nCh = nCh+5
968                  enddo!i7
969                               if(iUsed==0) exit
```

```
970                                 nLine=nLine+1
971                                                   cL(48:48)='|'
972                                               write(iP,*) CL
973                 enddo!i6
974               enddo!i5
975             enddo!i4
976           enddo!i3
977         enddo!i2
978       enddo!i1
979       cL= &
980       '                       ----------------- <-exact|unscaled->'// &
981       ' ------------------'c
982       write(iP,*) CL
983       cL=''c ;write(iP,*) CL
984                                                                                 return
985     End Subroutine CoefficientCount
986     !-------------------------------------------------------------------------------7 9
987
```