```
!SA2-hVecMath16Mod.f95
!2025.05.24.2030cdt- Dodecahedron Analysis & other real(16) subroutines.

!            Author- Jeffrey M. Setterholm, Lakeville,MN 55044 USA
!         IP Status- Free source code (e.g.: post copyright)
!
!          Computer- "T3"/Dell Precision T3500/Intel i5 E5520/Win10Pro-21H2
!                     ^name ^Mfgr.Id           ^chipset       ^OS
!                    /Absoft Pro Fortran 21.0.2/GeForce GTX 1050/f90gl~Glut3.7
!                     ^compiler ~Fortran 95     ^graphics card   ^graphics

!    f90gl bindings- public domain; see "https://math.nist.gov/f90gl/"

!Disclaimer:
! ****************************************************************
! **********   Individual cognition is always flawed,  **********
! **********         including yours and mine.         **********
! **********                  - So: -                  **********
! **********       Use this code at your own risk.     **********
! ****************************************************************

!Table of Contents: ...use to search...
!-----
!Module hVecMath16Mod
! Module DodecIO
! contains
!  Function Sqrj(Valin) Result(ValSqrout)                          2025.04.21.1545
!End Module DodecIO
!Subroutine DodecModel16(iP)                                      2013.10.23.0840
!Subroutine SinCos16(AngleD,SinAngle,CosAngle)                    2013.10.22.0940
!Subroutine ASinD16(Angle,SineAngle,iP)                           2013.10.20.0810
!Subroutine ACosD16(Angle,CosineAngle,iP)                         2013.10.20.0810
!Subroutine hPointPolar16(Ph,aRpyh,PDmag)                         2013.10.17.2210
!Subroutine hVnorm16(Vinh,Vouth,Vmag)                             2013.10.17.2210
!Subroutine hVecMirror2D16(Vinh,Vmirrorh,Vouth)                   2013.10.09.0800
!Subroutine hVecMirror3D16(Vinh,Vmirror1h,Vmirror2h,Vouth)        2013.10.09.0800
!Subroutine hCross16(Voh,Vi1h,Vi2h)                               2013.10.09.1005
!Subroutine MantExp16(ValIn16,rMant16,iExp,iP)                    2025.04.27.1110
!Subroutine Sqrt16a(ValIn,ValSqrOut,iMZP,iP)      Rev.A           2025.04.27.1110
!Function Factrl(N)                                               2025.04.21.1510
!-------------------------------------------------------------------------------
!-------------------------------------------------------------------------------7-9
!-------------------------------------------------------------------------------

Module DodecIO    !This is the user interface to the Dodecahedron model.
!2013.10.23.0840cdt JMS- Dodecahedron Input/Output Variables
!                      - DodecIO: Now using "type(DodecRec),save::Dd"
!use DodecIO
!use DodecIO, only:vNormal,Vertex,iFace,iEdge,DdWicki,iWikiToSett

 type::  DodecRec  ;sequence                                        !2013.10.16
  integer(4)::InitDodec
  !Wikipedia's Parameters - Theoretical values:
  real(16)  ::EdgeL          !Edge length   =(sqrt(5)-1)/sqrt(3) exactly
                            !~= 0.71364417954617986383939686092_16
  !For an edge of length 1   Radii:
  real(16)  ::ru             !outer          = (1 +  sqrt(5))  *sqrt(3) /4
  real(16)  ::rm             !mid-edge       = (     sqrt(5)      +3   )/4
  real(16)  ::ri             !inner          = (sqrt(sqrt(5)*11/10+5/2))/2
  real(16)  ::P1             !of face vertices=  sqrt(ru*ru-ri*ri)
  !Wikipedia's vertices are on a sphere with radius=sqrt(3):
  real(16)  ::Phi            ! =(1+sqrt(5))/2 exactly = the "Golden Ratio"
```

```
63                                ! = 1.6180339887498948482045868343656 4_16
64     real(16)  ::DPhi            ! ="one divided by Phi"
65                               !~= 0.6180339887498948482045868343656 4_16
66     real(16)  ::DdWicki(4,20)
67     !...the resulting coordinates must be divided by sqrt(3)
68     !...and pitched down:
69     real(16)  ::WickiPitch     !=90-(acosd(rm/ru)+acosd(ri/ru)
70                               !=-31.71747441146100532421390313982794_16 degrees
71                               !                                ...982700 ...playback
72     real(16)  ::EdgeLengthWiki(30) !Edge- lengths- ~= EdgeL ...Yes!
73     real(16)  ::EdgeLErrorRmsWiki  !           - RMS error
74     !...and renumbered as follows:
75     integer(4)::iWikiToSett(20)
76                      !=(/19,16,4,2,15,7,14,18,13,5,11,3,17,1,9,20,8,12,10,6/)
77     !... to match my model.
78
79     !So, my model parameters:
80                               !... are theoretically quantified by:
81     real(16)  :: sinV1Ang       != (ri-2*P1)/(ru*sqrt(5))
82     real(16)  ::cosV16Ang       !=       P1 / ru
83     real(16)  :: v1Ang          !Vertex -  #1- Pitch Angle w.r.t. the X-Y plane.
84       !V1Ang    =-10.8123169635717062912849447744 7353_16 !...Theoretical
85       !V1Ang    =-10.8123169635717062912849447744 7422_16 !...best fit to date.
86     real(16)  ::v16Ang          !       - #16-
87       !V16Ang   =-52.6226318593503043571428615051 0032_16 !...Theoretical
88       !V16Ang   =-52.6226318593503043571428615050 1010_16 !...best fit to date.
89     !Indices:
90     integer(4)::iFace(5,12)     !Face   - vertex  # list (CCW looking out)
91     integer(4)::iEdge(2,30)     !Edge   - vertex  # list
92     integer(4)::iVertex(3,20)   !vertex - vNormal # list (CCW looking out)
93     integer(4)::iEdgevN(2,30)   !edge   - vNormal # list
94     !Computed Vertices & Normals:
95     real(16)  ::Vertex(4,20)    !   Vertex   vector- (X,Y,Z,1.)
96     real(16)  ::vNormal(4,12)   !Face normal vector- (X,Y,Z,1.)
97     real(16)  ::EdgeLength(30)  !Edge- lengths- ~= EdgeL ...Yes!
98     real(16)  ::EdgeVnDist(30)  !Edge- Distance between vNormal vectors
99     real(16)  ::EdgeLErrorRms   !             - RMS error
100    !Nominal number of significant digits for real(16) variables herein:
101    integer(4)::iQuadDigits     !=31 for AbsoftProFortran vsn 9.0 (used here).
102    !Graphics drawing flags:
103    integer(4)::  iFaceSee(0:12)=1 !if(iFaceSee(0)=0: set desired face flags=1
104                               !=(/0,1,0,0,0,0,1,0,0,0,0,1,1/) for #1,#6,#11,&#12
105    integer(4)::iVertexSee(0:20)=0 !computed- based on iFaceSee(1:12).
106    integer(4)::  iEdgeSee(0:30)=0 !        -
107                         ! ^      (0)=-1: disable labels, (0)=-2: disable lines
108  end type DodecRec ;type(DodecRec),save::Dd !<-for "Dodecahedron"
109 !-------------------------------------------------------------------------------7 9
110  Contains
111 !-------------------------------------------------------------------------------7 9
112
113 Function Sqrj(Valin) Result(ValSqrout) !<-real(16)
114 !2025.04.21.1545cdt JMS- more accurate than intrinsic function sqrt(Valin).
115 !                       - ~ 30-place accuracy.
116 implicit none                                                    !arguments
117   real(16)::ValIn  !Input number
118   real(16)::ValSqrout != square root of Valin
119  !--                                                            !internals
120   integer(4)::iP
121   integer(4) iMZP
122   !-----
123                                  iP = 0
124    call Sqrt16a(ValIn,ValSqrOut,iMZP,iP)
125 End Function Sqrj
```

```
End Function Sqrj
!-------------------------------------------------------------------------------7 9

End Module DodecIO
!-------------------------------------------------------------------------------
!-------------------------------------------------------------------------------7 9
!-------------------------------------------------------------------------------
!Module hVecMath16Mod
!use hVecMath16Mod                                                  !2011.04.25
!---
!Implicit none
!---
!public ::        & !****************************************
!    DodecModel16   & !(iP)                                      *2013.10.18.1910
! ,  hPointPolar16  & !(Ph,aRpyh,PDmag)                         *2013.10.17.2210
! ,  hVnorm16       & !(Vinh,Vouth,Vmag)                        *2013.10.17.2210
! ,  hVecMirror2D16 & !(Vinh,Vmirrorh,Vouth)                     2013.10.09.0800
! ,  hVecMirror3D16 & !(Vinh,Vmirror1h,Vmirror2h,Vouth)          2013.10.09.0800
! ,  SinCos16       & !(AngleD,SinAngle,CosAngle)               2013.10.14.1150
! ,  ASinD16        & !(Angle,SineAngle,iP)                     *2013.10.18.1315
! ,  ACosD16        & !(Angle,CosineAngle,iP)                   *2013.10.18.1315
! ,  hCross16       & !(Voh,Vi1h,Vi2h)                           2013.10.09.1005
! ,  Sqrt16           !(ValIn,ValSqrOut,iP)                     *2013.10.18.1000

!-------------------------------------------------------------------------------7-9

Subroutine DodecModel16(iP)
!2013.10.22.1950cdt JMS- Graphics drawing flags.
!2013.10.20.1405cdt JMS- Using best-fit value of V1Ang & V16Ang
!2013.10.16.1815cdt JMS- Dodecahedron - the 12-face Platonic Solid
!                        which fits inside a sphere of radius=1.0
!------                - Reference: Wikipedia's article on "Dodecahedron".
 use DodecIO, only:Dd,Sqrj !,Pi16
 implicit none
 !-----                                                              Arguments
  integer(4)::iP
 !-----                                                              Internals
  !-- Internal use variables
  integer(4)::i,iDel,iDel2,j,k,L
  real(16)  ::v0=0._16  !0.
  real(16)  ::v1=1._16  !1.
  real(16)  ::Angle     !Yaw angle-
  real(16)  ::Ver1(4)   !Vertex vector- # 1- used as a template
  real(16)  ::Ver16(4)  !              - #16-
  real(16)  ::sY,cY     !Yaw  - cosine, sine
  real(16)  ::sP,cP     !Pitch- cosine, sine
  real(16)  ::vTemph(4) !Temporary vector
  real(16)  ::DdWickiTemp(4,20)
  real(16)  ::Angle1,Angle2
  integer(4)::iReverse(20)
  integer(4)::iMZP

  real(16)::Vsqrt                                                   !2025.04.20
  real(16)::Pi                                                      !2025.04.20

   !----- Dodecahedron initilization - Jeff Setterholm's algorithm:
   if(Dd%InitDodec.eq.0) then !-------------------------------
     if(iP>5) write(iP, &
    "('Dodecahedron model initialization: SA2-hVecMath16Mod.f95 @L181+')")
     !---
     Dd%iQuadDigits=precision(Dd%EdgeL)
     !---
```

```
187       Dd%EdgeL  =(sqrj(5._16)-1._16)/sqrj(3._16)       !Theoretical value.
188        !=   0.71364417954617986388393968609217_16
189
190       if(O>1) then !---------- Bypasses when (O>1); enters using (1>O)
191         if(iP>5) then  !Testing Sqrt(), call sqrt16(), & sqrj()
192         write(iP,"(/,'Testing Absoft`s sqrt():')")
193         Vsqrt   = sqrt(5._16)
194         write(iP,"(e45.35,' :5.-sqrt(  5.)*sqrt(  5.)'  )") 5._16-Vsqrt*Vsqrt
195        !Testing Absoft`s sqrt():
196        !   -0.40245584642661924590356647968292230E-15 :5.-sqrt(  5.)*sqrt(  5.)
197         write(iP,"(  'Testing my sqrt16() subroutine:')")
198         call sqrt16a(5._16,Vsqrt,iMZP,0)
199         write(iP,"(e45.35,' :5.-sqrt16(5.)*sqrt16(5.)'  )") 5._16-Vsqrt*Vsqrt
200        !Testing my sqrt16() subroutine:
201        !    0.00000000000000000000000000000000000E+00 :5.-sqrt16(5.)*sqrt16(5.)
202         write(iP,"(  'Testing my sqrj() function:')")
203         Vsqrt = sqrj(5._16)
204         write(iP,"(e45.35,' :5.-sqrj(  5.)*sqrj(  5.)',/)") 5._16-Vsqrt*Vsqrt
205        !Testing my sqrj() function:
206        !    0.00000000000000000000000000000000000E+00 :5.-sqrj(  5.)*sqrj(  5.)
207         write(iP,"('Dd%EdgeL = '/f40.35)")  Dd%EdgeL
208         endif!(iP>5)
209       endif!(O>1) -------- end bypass ----------------------------------------
210
211
212       Dd%ru= (1._16+sqrj(5._16))*sqrj(3._16)/4._16
213        !=  1.4012585384440735446766779353220 7_16
214       Dd%rm= (sqrj(5._16)+3._16)/4._16
215        !=  1.3090169943749474241022934171828 2_16
216       Dd%ri= (sqrj(sqrj(5._16)*11._16/10._16+5._16/2._16))/2._16
217        !=  1.1135163644116067351943750394869 6_16
218       Dd%P1=  sqrj(Dd%ru*Dd%ru-Dd%ri*Dd%ri)
219        !=
220      !-- wikipedia's vertex coordinates:
221       Dd%Phi     = (1._16+sqrj(5._16))/2._16
222        !=  1.6180339887498948482045868343656 4_16
223       Dd%dPhi=  1._16/Dd%Phi
224        !=  0.6180339887498948482045868343656 4_16
225       call ACosD16(Angle1,(Dd%rm/Dd%ru),0)
226       call ACosD16(Angle2,(Dd%ri/Dd%ru),0)
227       Dd%WickiPitch=90._16-(Angle1+angle2)
228        !=-31.717474411461005324213903139827 94_16
229
230      !Dd%sinV1Ang=(   ri-2    *   P1)/(   ru*sqrt(5    ))
231       Dd%sinV1Ang=(Dd%ri-2._16*Dd%P1)/(Dd%ru*sqrj(5._16))
232                 != -0.1875924740850798986013934690761_16
233       call ASinD16(Dd%V1Ang,Dd%sinV1Ang,0)
234      !Dd%V1Ang   =-10.8123169635717062912849447744 7353_16 !=Pitch ang.v#1 theor.
235       Dd%V1Ang   =-10.8123169635717062912849447744 7422_16 !...best fit to date.
236      !            -10.812316963571706291284944774 7412    !...as reported back
237      !                                             ^ 32nd place
238       Dd%cosV16Ang=Dd%P1/Dd%ru
239       call ACosD16(Dd%V16Ang,Dd%cosV16Ang,0) ;
240                    Dd%V16Ang=-Dd%v16Ang
241      !Dd%V16Ang  =-52.6226318593503043571428615051003 2_16 !=Pitch ang.v#16 theor
242       Dd%V16Ang  =-52.6226318593503043571428615050101 0_16 !...best fit to date.
243      !            -52.6226318593503043571428615050076 3_16 !...as reported back
244      !                                             ^ 32nd place
245
246      !-- Index faces vertices, edge vertices, vertex faces, & edge faces:
247       iDel =5 ;iDel2=0
248       do i=1,5 !Index faces(10 of 12 sets) {-CCW looking out} & all edges(30):
249          Dd%iFace(1:5,i    )=(/i    ,i+5,i+10,i+ 9+iDel,i+4+iDel /) !( 1  5]
```

```
249           Dd%iFace(1:5,i    )=(/i  ,i+5,i+10,i+ 9+iDel ,i+4+iDel /) ![ 1, 5]
250           Dd%iFace(1:5,i  +5)=(/i+5,i  ,i+15,i+16-iDel2,i+1-iDel2/) ![ 6,10]
251           Dd%iEdge(1:2,i*6-5)=  Dd%iFace(1:2,i)                      !1, 7,13,19,25
252           Dd%iEdge(1:2,i*6-4)=  Dd%iFace(2:3,i)                      !2, 8,14,20,26
253           Dd%iEdge(1:2,i*6-3)=  Dd%iFace(3:4,i)                      !3, 9,15,21,27
254           Dd%iEdge(1:2,i*6-2)=(/Dd%iFace(1,i+5),Dd%iFace(5,i+5)/) !4,10,16,22,28
255           Dd%iEdge(1:2,i*6-1)=(/Dd%iFace(5,i+5),Dd%iFace(4,i+5)/) !5,11,17,23,29
256           Dd%iEdge(1:2,i*6  )=(/Dd%iFace(4,i+5),Dd%iFace(3,i+5)/) !6,12,18,24,30
257         Dd%iVertex(1:3,i    )=(/i  ,i+4+iDel ,i+5/)                  ![ 1, 5]
258         Dd%iVertex(1:3,i  +5)=(/i+5,i+1-iDel2,i  /)                  ![ 6,10]
259         Dd%iVertex(1:3,i +10)=(/i  ,i+1-iDel2, 11/)                  ![11,15]
260         Dd%iVertex(1:3,i +15)=(/i+5,i+4+iDel , 12/)                  ![16,20]
261
262         Dd%iEdgevN(1:2,i*6-5)=(/i        ,i+ 5      /)              !1, 7,13,19,25
263         Dd%iEdgevN(1:2,i*6-4)=(/i        ,i+ 1-iDel2/)              !2, 8,14,20,26
264         Dd%iEdgevN(1:2,i*6-3)=(/i        ,  11      /)              !3, 9,15,21,27
265         Dd%iEdgevN(1:2,i*6-2)=(/i+1-iDel2, i+5      /)              !4,10,16,22,28
266         Dd%iEdgevN(1:2,i*6-1)=(/i+5      , i+6-iDel2/)              !5,11,17,23,29
267         Dd%iEdgevN(1:2,i*6  )=(/i+5      ,  12      /)              !6,12,18,24,30
268         if(i.eq.1) iDel =0
269         if(i.eq.4) iDel2=5
270       enddo!i
271           Dd%iFace(  1:5,11)=(/13,14,15,11,12/)                !Face #11-Top
272           Dd%iFace(  1:5,12)=(/19,18,17,16,20/)                !Face #12-Bottom
273
274       !-- Wikipedia model: ---------------------------------
275       Dd%iWikiToSett=(/19,16,4,2,15,7,14,18,13,5,11,3,17,1,9,20,8,12,10,6/)
276                                                    L=0
277       do i=-1,1,2 ;do j=-1,1,2 ;do k=-1,1,2       ;L=L+1
278       Dd%DdWicki(1:4,L)=(/i*1._16  ,j*1._16  ,k*1._16  ,1._16/)
279       enddo       ;enddo       ;enddo
280       do j=-1,1,2 ;do k=-1,1,2                    ;L=L+1
281       Dd%DdWicki(1:4,L)=(/  0._16  ,j*Dd%DPhi,k*Dd%Phi ,1._16/)
282       enddo       ;enddo
283       do i=-1,1,2 ;do j=-1,1,2                    ;L=L+1
284       Dd%DdWicki(1:4,L)=(/i*Dd%DPhi,j*Dd%Phi ,0._16    ,1._16/)
285       enddo       ;enddo
286       do i=-1,1,2              ;do k=-1,1,2       ;L=L+1
287       Dd%DdWicki(1:4,L)=(/i*Dd%Phi ,  0._16  ,k*Dd%DPhi,1._16/)
288       enddo                    ;enddo
289       !Normalize the vertex vectors:
290       do L=1,20 ;Dd%DdWicki(1:3,L)=Dd%DdWicki(1:3,L)/sqrj(3._16) ;enddo!L
291
292     !-- Compute the Edge Lengths for Wikipedia's model:
293
294       !Rotate the vertex vectors to the "Setterholm Model" orientation:
295       call SinCos16(Dd%WickiPitch,sp,Cp)
296       do j=1,20 ;L=Dd%iWikiToSett(j) ; iReverse(L)=j
297         vTemph(1)= Dd%DdWicki(1,L)*cP+Dd%DdWicki(3,L)*sP
298         vTemph(2)= Dd%DdWicki(2,L)
299         vTemph(3)=-Dd%DdWicki(1,L)*sP+Dd%DdWicki(3,L)*cP
300         DdWickiTemp(1:3,j)=vTemph(1:3)
301       enddo!L
302       !Remap the Wickimodel vector indices.
303       Dd%DdWicki=DdWickiTemp
304
305      !if(iP>5) write(iP,"('Edge length errors @L303:')")
306
307       Dd%EdgeLErrorRmsWiki=0._16
308       do i=1,30
309         vTemph= Dd%DdWicki(1:4,Dd%iEdge(2,i))  &
310                -Dd%DdWicki(1:4,Dd%iEdge(1,i))
```

```
311         Dd%EdgeLengthWiki(i)= ( vTemph(1)*vTemph(1) &
312                                +vTemph(2)*vTemph(2) &
313                                +vTemph(3)*vTemph(3)   )
314         Dd%EdgeLengthWiki(i)=sqrj(Dd%EdgeLengthWiki(i))-Dd%EdgeL
315         Dd%EdgeLErrorRmsWiki=Dd%EdgeLErrorRmsWiki+ Dd%EdgeLengthWiki(i)  &
316                                                   *Dd%EdgeLengthWiki(i)
317
318        !write(iP,"(i3,f40.35)") i,Dd%EdgeLErrorRmsWiki !/i
319       enddo!i
320       Dd%EdgeLErrorRmsWiki=sqrj(Dd%EdgeLErrorRmsWiki/30._16)
321      !write(iP,"(/3x,f40.35,' Dd%EdgeLErrorRmsWiki')")  Dd%EdgeLErrorRmsWiki
322
323
324       Dd%InitDodec=1 ;write(6,"(a1)") char(7)                              !Beep
325     endif !InitDodec=0 ------------------------------------------------------------
326
327     !-- Update the Graphics drawing flags:
328    !Dd%iFaceSee(0:12)=(/0,1,0,0,0,0,1,0,0,0,0,1,1/)
329     Dd%iVertexSee(1:20)=0 ;Dd%iEdgeSee(1:30)=0
330     if(Dd%iFaceSee(0).eq.1) Dd%iFaceSee=1                                 !Reset
331     do i=1,12 ;if(Dd%iFaceSee(i).eq.0) cycle
332       do j=1,5 ;k=Dd%iFace(j,i) ;Dd%iVertexSee(k)=1 ;enddo!j
333     enddo!i
334     do i=1,30 ;if(Dd%iVertexSee(Dd%iEdge(1,i)).eq.0) cycle
335                if(Dd%iVertexSee(Dd%iEdge(2,i)).eq.0) cycle
336       Dd%iEdgeSee(i)=1
337     enddo!i
338
339     !----- From here on: V1Ang & V16Ang quantify the model...
340     !-- Setup the Vertex templates:
341     call SinCos16(Dd%v1Ang ,sP,cP) ;Ver1 =(/cP,v0,-sP,v1/) != vertex#1
342     call SinCos16(Dd%v16Ang,sP,cP) ;Ver16=(/cP,v0,-sP,v1/) != vertex#16
343     do i=1,5  !Spin the vertex templates to define all the vertices:
344       Angle=(i-1)*72._16       ;call SinCos16(Angle,sY,cY)
345       Dd%Vertex( 1:4,i  )=(/ Ver1(1)*cY, Ver1(1)*sY,  Ver1(3),v1/)! 1- 5
346       Dd%Vertex(1:4,i+15)=(/Ver16(1)*cY,Ver16(1)*sY, Ver16(3),v1/)!16-20
347       Angle=(i-1)*72._16+36._16 ;call SinCos16(Angle,sY,cY)
348       Dd%Vertex( 1:4,i+5)=(/ Ver1(1)*cY, Ver1(1)*sY, -Ver1(3),v1/)! 6-10
349       Dd%Vertex(1:4,i+10)=(/Ver16(1)*cY,Ver16(1)*sY,-Ver16(3),v1/)!11-15
350     enddo!i
351
352     !-- Compute each Face Normal vector by averaging its five vertices:
353     do i=1,12 ;Dd%vNormal(1:4,i)=0._16
354       do j=1,5
355         Dd%vNormal(1:4,i)=Dd%vNormal(1:4,i)+Dd%Vertex(1:4,Dd%iFace(j,i))
356       enddo!j
357       Dd%vNormal(1:4,i)=Dd%vNormal(1:4,i)/5._16
358     enddo!i
359
360     !-- Compute the resulting Edge Lengths:
361     Dd%EdgeLErrorRms=0._16
362     do i=1,30
363       vTemph=Dd%Vertex(1:4,Dd%iEdge(2,i))-Dd%Vertex(1:4,Dd%iEdge(1,i))
364       Dd%EdgeLength(i)= ( vTemph(1)*vTemph(1) &
365                          +vTemph(2)*vTemph(2) &
366                          +vTemph(3)*vTemph(3)   )
367       call Sqrt16a(Dd%EdgeLength(i),Dd%EdgeLength(i),iMZP,0)
368       Dd%EdgeLErrorRms=Dd%EdgeLErrorRms+ (Dd%EdgeLength(i)-Dd%EdgeL)  &
369                                         *(Dd%EdgeLength(i)-Dd%EdgeL)
370     enddo!i
371     Dd%EdgeLErrorRms=sqrj(Dd%EdgeLErrorRms/30._16)
372
373     !----- Report the results:
```

```
373     !----- Report the results.
374     if(iP.gt.5) then
375       write(iP,"(/79('/'))")
376       write(iP,"( 'Begin DodecModel16 printout @L374+' )")
377       write(iP,"(/'  real(16) precision =',i3,' significant digits')")  &
378                                          Dd%iQuadDigits
379       write(iP,"( '     & tentatively: printouts truncate at 34 places max.')")
380       write(iP,"(/'ru            =',f40.32)")  Dd%ru
381       write(iP,"( 'rm            =',f40.32)")  Dd%rm
382       write(iP,"( 'ri            =',f40.32)")  Dd%ri
383       write(iP,"( 'P1            =',f40.32)")  Dd%P1
384       write(iP,"(/' sinV1Ang     =',f40.32,' -theoretical'    )")    Dd%sinV1Ang
385       write(iP,"( ' V#1 pitch Ang=',f40.32,' -best-fit value used' )")  Dd%V1Ang
386       write(iP,"( 'cosV16Ang     =',f40.32,' -theoretical'    )")   Dd%cosV16Ang
387       write(iP,"( 'V#16 Pitch Ang=',f40.32,' -best-fit value used'/)") Dd%V16Ang
388       write(iP,"( 'Vertices:')")
389         write(iP,"(i2,' =(/',2(f35.31,','),f35.31,'/)')")   1, ver1( 1:3)
390         write(iP,"(i2,' =(/',2(f35.31,','),f35.31,'/)'/)") 16, ver16(1:3)
391       do i=1,20
392         write(iP,"(i2,' =(/',2(f35.31,','),f35.31,'/)')") i,Dd%Vertex( 1:3,i)
393       enddo!i
394       write(iP,"(/'Face normals:')")
395       do i=1,12
396         write(iP,"(i2,' =(/',2(f35.31,','),f35.31,'/)')") i,Dd%vNormal( 1:3,i)
397       enddo!i
398       !-- Indices & Lengths:
399       write(iP,"(/'       CCW looking out    CCW looking out' )")
400       write(iP,"( 'Face:    VertexIndex   Vertex:  FaceIndex ',\)")
401       write(iP,"( ' Edge:   vNIndex VertexIndex &   ComputedEdgeLength')")
402       do i=1,30
403         if(i.le.12) then
404           write(iP,"(' #',i2,'=( ',4(i2,','),i2,'/)',\)") i,Dd%iFace(1:5,i)
405          else
406           write(iP,"('                          ',\)")
407         endif !i<=12
408         if(i.le.20) then
409           write(iP,"('   #',i2,'=( ',2(i2,','),i2,'/)',\)") i,Dd%iVertex(1:3,i)
410          else
411           write(iP,"('                     ',\)")
412         endif !i<=20
413         write(iP,"('   #',i2,' =( ',1(i2,','),i2,'/)',\)") i,Dd%iEdgeVn(1:2,i)
414         write(iP,"(       ' =( ',1(i2,','),i2,'/)',\)")   Dd%iEdge(  1:2,i)
415         write(iP,"(' :',f35.32)") Dd%EdgeLength(i)
416       enddo!i
417         write(iP,"(/48x,'Theoretical Edge Length=',f37.34)") Dd%EdgeL
418         write(iP,"( 84x,'1         2         3')")
419         write(iP,"( 75x,'123456789012345678901234567890123456789012')")
420         write(iP,"( 79x,'RMS Error=',e13.6,' : ^')") Dd%EdgeLErrorRms
421
422       Dd%EdgeL  =(sqrj(5._16)-1._16)/sqrj(3._16)       !Theoretical value.
423        !=   0.71364417954617986388393968609217_16
424
425       write(iP,"( 'Wikipedia Model:')")
426       write(iP,"( 'Phi           =',f40.37)")  Dd%Phi
427       !=      1.61803398874989484820458683436564_16
428       write(iP,"( 'dPhi          =',f40.37)")  Dd%dPhi
429       !=      0.61803398874989484820458683436564_16
430       write(iP,"( '...vertex algebraic summary:')")
431       write(iP,"( 'W#:  S#:     Numeric values:')")
432                                                  L=0
433       do i=-1,1,2 ;do j=-1,1,2 ;do k=-1,1,2      ;L=L+1
434         write(iP,"(i3,' =',i3,' =(/',i2,'     ,',i2,'     ,',i2,'     /)')")  &
```

```
435         L,iReverse(L)        ,i          ,j          ,k
436       enddo       ;enddo       ;enddo
437       do j=-1,1,2 ;do k=-1,1,2                    ;L=L+1
438         write(iP,"(i3,' =',i3,' =(/',i2,'     ,',i2,'*dPhi,',i2,'*phi /)')")  &
439         L,iReverse(L)        ,0          ,j          ,k
440       enddo       ;enddo
441       do i=-1,1,2 ;do j=-1,1,2                    ;L=L+1
442         write(iP,"(i3,' =',i3,' =(/',i2,'*dPhi,',i2,'*Phi ,',i2,'     /)')")  &
443         L,iReverse(L)        ,i          ,j          ,0
444       enddo       ;enddo
445       do i=-1,1,2             ;do k=-1,1,2      ;L=L+1
446         write(iP,"(i3,' =',i3,' =(/',i2,'*Phi ,',i2,'     ,',i2,'*dPhi/)')")  &
447         L,iReverse(L)        ,i          ,0          ,k
448       enddo                   ;enddo
449        !write(iP,"( 37x,'RMS Edge Length Errors before rotation=',d13.6)") &
450        !                  EdgeLErrorRmsWiki
451       write(iP,"(/'...scale and rotate the model:')")
452      !write(iP,"( 'WickiPitch    =',f40.37)")  WickiPitch
453       !=    -31.71747441146100532421390313982794_16
454       write(iP,"( 'Divide by sqrj(3)=',f40.32)")  sqrj(3._16)
455       !=    -31.71747441146100532421390313982794_16
456       write(iP,"( 'and pitch down by ',f40.32,' degrees.')")  Dd%WickiPitch
457       !=    -31.71747441146100532421390313982794_16
458
459       write(iP,"(/'... resulting Wicki-Vertices in Setterholm Model order:')")
460       write(iP,"( 'W#:')")
461       do i=1,20 ;L=Dd%iWikiToSett(i) !=iReverse(i)
462         write(iP,"(i2,' =(/',2(f35.31,','),f35.31,'/)')") L,Dd%DdWicki( 1:3,i)
463       enddo!i
464       write(iP,"( '       Edge#   W1 W2       Length',29x,'Length error')")
465       do i=1,30
466         write(iP,"(9x,'#',i2,'=(/',i2,',',i2,'/)=',f38.34,d13.4)") &
467         i,Dd%iWikiToSett(Dd%iEdge(1:2,i)),Dd%EdgeLengthWiki(i)+Dd%EdgeL &
468                                          ,Dd%EdgeLengthWiki(i)
469       enddo!i
470         write(iP,"(     'Theoretical Edge Length=',f37.34)") Dd%EdgeL
471         write(iP,"( 36x,'1         2         3')")
472         write(iP,"( 27x,'123456789012345678901234567890123456789012345678901234')")
473         write(iP,"( 33x,'RMS Error=',e13.6,' : ^')") Dd%EdgeLErrorRmsWiki
474
475       write(iP,"(/'Graphics drawing flags:' )")
476       write(iP,"( i2,1x,12i1,18x,' :  iFaceSee')")   Dd%iFaceSee
477       write(iP,"( i2,1x,20i1,10x,' :iVertexSee')") Dd%iVertexSee
478       write(iP,"( i2,1x,30i1    ,' :  iEdgeSee')")   Dd%iEdgeSee
479       write(iP,"( ' ^         1         2         3')")
480       write(iP,"( ' 0|12345678901234567890123456789012345678901234567890')")
481       write(iP,"(/'End of DodecModel16 printout. @L479')")
482       write(iP,"( 79('/')/)")
483     endif !iP>5
484     return
485 End Subroutine DodecModel16
486 !-------------------------------------------------------------------------------7 9
487 Subroutine SinCos16(AngleD,SinAngle,CosAngle)
488 !2013.10.22.0940cdt JMS- Using 41! & 40!
489 !2013.10.14.1150cdt JMS- "Improved" DtoR value confirmed!
490 !                      - Tested vs. absoft's sinD & cosD in [-720.36,720.36]deg
491 !                        in .0036 degree steps. Disagreements all below 1.d-18
492 !2013.10.13.0950cdt JMS- Computes the sine & cosine of an angle
493 !                      - result based on QPClosure's testing:   2013.10.13.0950
494 !-----          From the web:   Pi=3.14159265358979323846264338327950288419 71...
495 !   Mine: 29 places:              3.14159265358979323846264338323464000 :Pi
496 !                     Hence: DtoR=0.01745329251994329576923690768490 49
497 !               Confirmed: SinCos16(30  16 sX cX)
```

```
!              Confirmed: SinCos16(30._16,sY,cY)
!              sY= 0.5000000000000000000000000000000456   Accurate to 30 places!
!                        SinCos16(60._16,sY,cY)
!              cY= 0.4999999999999999999999999999999088   Accurate to 30 places!
!-----
 implicit none                                                          !arguments
  real(16)  ::AngleD   !Input - angle (degrees)
  real(16)  ::SinAngle !Output-   sine(AngleD)
  real(16)  ::CosAngle !      - cosine(AngleD)
  !                                                                     !internals
  real(16)  ::AngleDL
  real(16)  ::DtoR
  real(16)  ::AngleR
  real(16)  ::SumNum
  integer(4)::i,iQ,iQpm,iQuadrant,iQuadrant2
  real(16)  ::Xsq
  real(16)  ::SinAngleL
  !-----
   !-- Internalize the angle:
   AngleDL=AngleD
   !-- Bound to within [-180.,180.] degrees:
   if(AngleDL.gt. 360._16) AngleDL=AngleDL-int( AngleDL/360._16)*360._16
   if(AngleDL.le.-360._16) AngleDL=AngleDL+int(-AngleDL/360._16)*360._16
   if(AngleDL.gt. 180._16) AngleDL=AngleDL-360._16
   if(AngleDL.le.-180._16) AngleDL=AngleDL+360._16
   !-- Determine the quadrant
   iQuadrant=floor(AngleDL/90._16)
   select case (iQuadrant)
    case(-2,2       )  ![-180.,-90.)
     iQuadrant2=-1 ;AngleDL=AngleDL+180._16
    case(-4,-1,0,3,4)  ![-90., +90.]
     iQuadrant2= 0
    case(-3,1       )  !(+90.,+180.]
     iQuadrant2=+1 ;AngleDL=AngleDL-180._16
   end select !iQuadrant
   !-- Convert degrees to radians
  !DtoR  =  0.017453292519943295769236907684905_16   !<- Math Handbook@page 12 to 22places
  !DtoR  =  0.017453292519943295769236907685_16   !JMS previous best
   DtoR  =  0.017453292519943295769236907684905_16  !using Pi{40}/180.

   AngleR=AngleDL*DtoR !Using 28! & 29! ...can the value really be that good?
!                      !Using 36! & 37!        the results were the same.
   Xsq=AngleR*AngleR
   SumNum=0._16    ;iQpm=-1._16  ;               !  41! ~= 3.3452d49
   do i=41,3,-2  ;iQ=i ;iQpm=-iQpm              !  37! ~= 1.3764d43
     SumNum=(SumNum+iQpm)*Xsq/(iQ*(iQ-1._16))   !  29! ~= 8.8417d30
   enddo !i
   SinAngleL=(SumNum+1._16)*AngleR
  !-----
   SumNum=0._16    ;iQpm=-1._16                 !  40! ~= 8.1591d47
   do i=40,2,-2  ;iQ=i ;iQpm=-iQpm              !  36! ~= 3.7199d41
     SumNum=(SumNum+iQpm)*Xsq/(iQ*(iQ-1._16))   !  28! ~= 3.0488d29
   enddo !i
   CosAngle=SumNum+1._16
  !-----
   SinAngle=SinAngleL
   if(iQuadrant2.ne.0) then; Sinangle=-sinangle; CosAngle=-CosAngle ;endif
   return
End Subroutine SinCos16
!-------------------------------------------------------------------------------7 9
Subroutine ASinD16(Angle,SineAngle,iP)
!2013.10.20.0810cdt JMS- Computes the asind(SineAngle) to ~31 signif. places.
```

```
559  use Dodecio, only:sqrj
560  implicit none                                                    !arguments
561   real(16)  ::Angle    !Output- angle                                 degrees
562   real(16)  ::SineAngle!Input - sine of the angle                        "
563   integer(4)::iP
564   !                                                                !internals
565   real(16)  ::Atemp,sinAtemp,sA,cA
566   integer(4)::i,iRev,iSignval
567   real(8)   ::DtoR=0.01745329251994329576923690768490 5_16
568   !-----
569    sinAtemp=SineAngle    ;iSignVal= 1
570    if(sinAtemp.lt.0._16)  iSignVal=-1
571    sinAtemp=SineAngle   *iSignVal   ;iRev=0
572    if(sinAtemp.gt..7071_16) then    ;iRev=1
573      sinAtemp=sqrj(1._16-sinAtemp*sinAtemp)
574    endif !
575    Atemp=dasind(sinAtemp) +.001d0     !getting into the ballpark
576    iRev=0
577       i=0
578   !do i=1,10
579 10    i=i+1
580      ATemp=Atemp-(sA-sinAtemp)/DtoR
581      call SinCos16(ATemp,sA,cA)
582      if(iP.gt.5) write(iP,"(i2,2f40.33       )")  i,ATemp,(sA-sinAtemp)/DtoR
583
584      if(i.gt.200) then                                            !2024.04.21
585        write( 6,"(i3,2f40.33       )")  i,ATemp,(sA-sinAtemp)/DtoR
586        pause &
587       'pause: ASinD16 @L534 didn`t converge <1.d-30 error in 200 iterations'
588        goto 20
589      endif!(i.gt.200)
590
591      if(abs(sA-sinAtemp).gt.1.d-30) goto 10
592     !if(abs(sA-sinAtemp).gt.1.d-33) goto 10
593   !enddo!i
594 20 continue
595    select case(iRev)
596     case(0) ;Angle=        ATemp *iSignVal
597     case(1) ;Angle=(90._16-ATemp)*iSignVal
598    end select !iRev
599    return
600   !Test case:
601   !     V1Ang  = -10.812316963571706291284944774472 - previous result
602   !sind(V1Ang) =  -0.187592474085079899860139346908 - theoretical
603   !     V1Ang  = -10.812316963571706291284944774474 - this algorithm
604 End Subroutine ASinD16
605 !-------------------------------------------------------------------------------7 9
606 Subroutine ACosD16(Angle,CosineAngle,iP)
607 !2013.10.20.0810cdt JMS- Computes the acosd(coainwAngle) to ~31 signif. places.
608 !                       an Angle in the interval: [0.,+180.] is returned.
609  implicit none                                                    !arguments
610   real(16)  ::Angle    !Output- angle                                 degrees
611   real(16)  ::CosineAngle!Input - cosine of the angle                    "
612   integer(4)::iP
613   !                                                                !internals
614   real(16)  ::Atemp,cosAtemp
615   integer(4)::iSignval
616   !-----
617    cosAtemp=CosineAngle ;iSignval= 1
618    if(cosAtemp.lt.0.)    iSignval=-1
619    call ASinD16(      Atemp,cosAtemp*iSignVal,iP)
620                       Atemp= 90._16-Atemp
621    if(iSignval.eq.-1) Atemp=180._16-Atemp
```

```
621     if(iSignVal.eq.-1) Atemp=180._16-Atemp
622                        Angle=Atemp
623     if(iP.gt.5) write(iP,"(    f40.33)") Angle
624     return
625    !Test case:
626    !     V16Ang = -52.622631859350304357142861505007 - previous result
627    !cosd(V16Ang)=   0.607061998206686223095391581420 - theoretical
628    !     V16Ang =  52.622631859350304357142861505100 - this algorithm
629    !cosd(V16Ang)=   0.607061998206686223095391581419   ... and its cosine
630  End Subroutine AcosD16
631  !-----------------------------------------------------------------------7 9
632  Subroutine hPointPolar16(Ph,aRpyh,PDmag)
633  !2013.10.17.2210cdt JMS- Quad precision version
634  !2013.10.01.2155cdt JMS- Expresses a 3D point in FS ~polar coordinates.
635   !-----
636   Implicit none                                                  !arguments
637    real(16)  ::Ph(4)     !Input - Point#1
638    real(16)  ::aRpyh(4) !Output- att.(Roll,Pitch,Yaw)-             [degrees]
639    real(16)  ::PDmag     !      - Distance to point
640    !---                                                          !internals
641    real(16)  ::Ptemph(4)
642    !-----
643     Ptemph=Ph
644     call hVnorm16(Ph,Ptemph,PDmag)
645     aRpyh=(/0._16,0._16,0._16,1._16/)
646
647     aRpyh(2)=-dasind(Ptemph(3))          !getting into the ballpar          !?
648    !call ASinD16(aRpyh(2),Ptemph(3),0)                          !2025.04.20
649    !aRpyh(2) =aRpyh(2)
650
651     if((Ptemph(1).ne.0._16).or.(Ptemph(2).ne.0._16))  &
652     aRpyh(3)=datan2d(Ptemph(2),Ptemph(1))
653     return
654  End Subroutine hPointPolar16
655  !-----------------------------------------------------------------------7 9
656  Subroutine hVnorm16(Vinh,Vouth,Vmag)
657  !2013.10.17.2210cdt JMS- Quad precision version
658  !2013.10.03.2315cdt JMS- Normalizes a homogeneous vector: Vouth=norm(Vinh)
659   use Dodecio, only:sqrj
660   implicit none                                                  !arguments
661    real(16)  ::Vinh(4)     !Vector- input- #1
662    real(16)  ::Vouth(4)    !              - #2
663    real(16)  ::Vmag        !      - Dot product = V1h .dot. V2h
664    !---                                                          !internals
665    real(16)  ::Temph(4)    !Isolates DotP from V1h() & V2h().
666    real(16)  ::TempMag
667    !-----
668                             Temph=Vinh
669    if(Vinh(4).ne.0._16) Temph=Temph/Vinh(4)
670    TempMag=Temph(1)*Temph(1)+Temph(2)*Temph(2)+Temph(3)*Temph(3)
671    if(TempMag.gt.0._16) then
672      Vouth(1:3)=Temph(1:3)/sqrj(TempMag) ;Vouth(4)=1._16 ;Vmag=sqrj(TempMag)
673     else
674      Vouth=0._16                  ;Vmag=0._16
675    endif !TempMag>0
676    return
677  End Subroutine hVnorm16
678  !-----------------------------------------------------------------------7 9
679  Subroutine hVecMirror2D16(Vinh,Vmirrorh,Vouth)
680  !2013.10.09.0800cdt JMS- Mirrors one vector across another
681  !                        without using trig functions.
682   implicit none                                                  !arguments
```

```
683    real(16)  ::Vouth(4)   !Output- vector  (X,Y,Z)
684    real(16)  ::Vmirrorh(4)!Input - mirror vector
685    real(16)  ::Vinh(4)    !      - vector
686    !                                                                !internals
687    real(16)  ::DotIn
688    real(16)  ::VmirrorMagSq
689    real(16)  ::ScaleFactor
690    real(16)  ::vTemph(4)
691    !-----
692     DotIn=            Vinh(1)*Vmirrorh(1)  &
693                      +Vinh(2)*Vmirrorh(2)  &
694                      +Vinh(3)*Vmirrorh(3)
695     VmirrorMagSq= Vmirrorh(1)*Vmirrorh(1)  &
696                  +Vmirrorh(2)*Vmirrorh(2)  &
697                  +Vmirrorh(3)*Vmirrorh(3)
698                               ScaleFactor=2._16*DotIn/VmirrorMagSq
699     vTemph(1:3)=Vmirrorh(1:3)*ScaleFactor-Vinh(1:3)
700     vTemph( 4 )=                          Vinh( 4 )
701     Vouth      =Vtemph
702     return
703  End Subroutine hVecMirror2D16
704  !-------------------------------------------------------------------------------7 9
705  Subroutine hVecMirror3D16(Vinh,Vmirror1h,Vmirror2h,Vouth)
706  !2013.10.14.0930cdt JMS- Remirroring back errors are tiny... ~< 1.d-29
707  !2013.10.09.0800cdt JMS- Mirrors one vector across two other vectors
708  !                        without using trig functions.
709   implicit none                                                   !arguments
710    real(16)  ::Vouth(4)   !Output- vector  (X,Y,Z)
711    real(16)  ::Vmirror1h(4)!Input - mirror vector- #1
712    real(16)  ::Vmirror2h(4)!                     - #2
713    real(16)  ::Vinh(4)    !      - vector
714    !                                                                !internals
715    real(16)  ::Vcrossh(4)
716    real(16)  ::Vinh2(4)
717    !-----
718     call hCross16(Vcrossh,Vmirror1h,Vmirror2h)
719     call hVecMirror2D16(Vinh,Vcrossh,Vinh2)
720     Vouth(1:3)=-Vinh2(1:3)
721     Vouth( 4 )= Vinh2( 4 )
722     return
723  End Subroutine hVecMirror3D16
724  !-------------------------------------------------------------------------------7 9
725  Subroutine hCross16(Voh,Vi1h,Vi2h)
726  !2013.10.09.1005cdt JMS- Cross product of V1h & V2h
727  implicit none                                                   !arguments
728    real(16)  ::Vi1h(4)     !Input Vector
729    real(16)  ::Vi2h(4)     !Input Vector
730    real(16)  ::Voh(4)      !Voh = Vi1h X Vi2h
731   !--                                                            !internals
732    real(16)  ::Temph(4)
733    !-----
734     Temph(1)=Vi1h(2)*Vi2h(3)-Vi1h(3)*Vi2h(2)
735     Temph(2)=Vi1h(3)*Vi2h(1)-Vi1h(1)*Vi2h(3)
736     Temph(3)=Vi1h(1)*Vi2h(2)-Vi1h(2)*Vi2h(1)
737     Temph(4)=Vi1h(4)*Vi2h(4)
738     Voh=Temph
739     return
740  End Subroutine hCross16
741  !-------------------------------------------------------------------------------7 9
742  Subroutine MantExp16(ValIn16,rMant16,iExp,iP)
743  !2025.04.27.1110cdt JMS- Mantissa and Exponent of a ValIn16
744  implicit none                                                   !arguments
```

```
745   real(16)  ::ValIn16     ! Input- value
746   real(16)  ::rMant16     !Mantissa
747   integer(4)::iExp        !Exponent
748   integer(4)::iP
749  !--                                                              !internals
750   real(16)     ::Valu
751   character(60)::c60
752   character(45)::c45
753   !-----
754    Valu = ValIn16
755    if(Valu>0._16) then
756      Write(c60,"(e45.35,    ' :ValTemp')"       ) Valu
757      write(c45,"(a39,'E+00'             )"       ) c60(1:39)
758      read( c45,"(e45.35                 )"       ) rMant16
759      read(c60(42:45),"(i4               )",err=25) iExp
760                                           goto 26
761 25   read(c60(43:45),"(i3                )"       ) iExp
762 26   continue
763     else
764      rMant16 = 0._16
765      iExp    = 0
766    endif!(Valu>0._16)
767
768    if(iP>5) then
769      Write( iP,"(e45.35  ,'  mantissa and exponent:')") Valu
770      write(iP ,"(e45.35  ,' :mantissa')"              ) rMant16
771      write(iP,"(41x,SP,i4,' :',13x,      'exponent')" ) iExp
772      write(iP ,"( '      12345678901234567890123456789012345')")
773      write(iP ,"( '               1         2         3')")
774    endif!(iP>5)
775     !-----
776    return
777 End Subroutine MantExp16
778 !         1         2         3         4         5         6
779 !12345678901234567890123456789012345678901234567890123456789O
780 !    0.25000000000000000000000000000000000E+01  mantissa and exponent:
781 !    0.25000000000000000000000000000000000E+00 :mantissa
782 !                                           +1 :             exponent
783 !-------------------------------------------------------------------------7 9
784
785 Subroutine Sqrt16a(ValIn,ValSqrOut,iMZP,iP) !Rev A - 4 arguments.
786 !2025.04.27.0830cdt JMS- ValSqrOut = sqrt(ValIn)    APF21.0.2: has errors.
787 !                                                  (Perhaps a hardware issue.)
788 !2013.10.22.0940cdt JMS- This routine isn't needed. APF9.0's : sqrt() is fine!
789 !2013.10.20.1655cdt JMS- Iteratively computes a real(16) square root.
790
791 !The operating domain of the mantissa algorithm is [.1 -to- 10.]
792 !Odd-order exponents are made even in Iteration i = 1 by:
793 !    1. multiplying the mantissa by 10._16, and then
794 !    2. subtracting one from the exponent.
795 !    3. Setting Y(1) = sqrt(10._16)  ... which brings
796 !         the iterative domain into [.1 -to- 1.] which is well-behaved.
797 !The even-order exponent is divided by 2.
798
799 implicit none                                                    !arguments
800   real(16)  ::ValIn       ! Input- value
801   real(16)  ::ValSqrOut   ! = sqrt(abs(ValIn))
802   integer(4)::iMZP        !"Minus,Zero, or Plus = {-1,0,+1} =-1:imaginary
803 !                         ! & ValIn=(ValSqrOut*ValSqrOut)*iMZP
804   integer(4)::iP
805  !--                                                              !internals
806   real(16)  ::Valu
```

```
807    real(16)  ::rMantissa
808    integer(4)::iExponent
809
810
811    real(16)  ::VL,VsqrL,VerrL
812    integer(4)::i
813
814    real(16)  ::X(0:9)
815    real(16)  ::Y(0:9),Yout
816    integer(4)::Init
817    character(45)::c45
818    real(16)  ::Sqrt10
819    !-----
820     if(iP>5) write(iP,"(/'Sqrt16(): Valin =')")   !/2x,e43.35)") ValIn
821     if(     ValIn< 0._16) then; Valu      = -Valin; iMZP = -1
822      elseif(ValIn==0._16) then; ValSqrOut =  0._16; iMZP =  0     ;return
823      else                       ; Valu      =  ValIn; iMZP = +1
824     endif! (ValIn< 0._16)
825     call MantExp16(Valu,rMantissa,iExponent,iP)
826    !--Mantissa computation:
827                       X(0) = rMantissa
828     if(iP.gt.5) write(iP,"( '           iMZP =',i3       )") iMZP
829     if(iP.gt.5) write(iP,"( 'Mantissa:')")
830     Yout = 1._16
831     do i = 1, 9;    X(i) = X(i-1)                                        !key ops !
832           Y(i) = (((X(i)- 5._16)* X(i)+15._16)* X(i)+ 5._16)/16._16 ! "   "  !
833       if(i==1) then  !Accomodatint odd exponents:
834         if(mod(iExponent,2).ne.0) then
835           Sqrt10 = 3.16227766016837933199889354443 27_16
836           Y(i) = Sqrt10
837           iExponent = iExponent-1
838                       X(0) = X(0)*Sqrt10*Sqrt10
839         endif!(mod(iExponent,2).ne.0)
840       endif!(i==1)
841       if(iP>5) write(iP,"(i3,2f36.32, ' :i X(',i1,') -> Y(',i1,')')") &
842                     i,X(i-1),Y(i),i-1,i
843                      X(i) = X(i-1)/(Y(i)*Y(i))
844      Yout = Yout*Y(i)
845      if(X(0)-Yout*Yout == 0._16) exit
846      if(Y(i)           == 1._16) exit
847     enddo!i
848     if(iP>5) then
849                 write(iP, &
850        "(7x,'Computed Mantissa X value:',10x,'Computed Mantissa Y value:')")
851                 write(iP,"( i3,2f36.32, ' :i X  -> Y ')")-1,X(0),Yout
852                 write(iP,"( 3x, f36.32,37x,':  X(0)')") X(0)
853                 write(iP ,"( '       12345678901234567890123456789012')")
854                 write(iP ,"( '                  1         2         3')")
855                 write(iP,"( 9x,e30.20,7x, ':Mantissa internal X value discrepancy')")
856                           X(0)-Yout*Yout
857     endif!(iP>5)
858    !--Exponent computation:
859     iExponent = iExponent/2
860    !--Conversion to text:
861     if(abs(iExponent) <100) write(c45,"(f40.35,'E',SP,i3.2)") Yout,iExponent
862     if(abs(iExponent)>=100) write(c45,"(f40.35,    SP,i4.3)") Yout,iExponent
863    !--Conversion to real(16) result:
864     read(c45,"(e45.35)") ValSqrOut
865     if(iP>5) write(iP,"(e45.35,' :ValSqrOut')") ValSqrOut
866     if(iP>5) write(iP,"(i5,40x,' :iMZP  {Minus,Zero, or Plus}')") iMZP
867     if(iMZP==-1) write(iP, &
868        "('    ***The square root is an imaginary number.***')")
```

```
869     call SaveOutFile
870                                                                    return
871 End Subroutine Sqrt16a
872
873 !Polynomial  coefficients C :  3rd order             ...these are exact!
874 !1           0.31250000000000000000000000000000 *x^0
875 !2           0.93750000000000000000000000000000 *x^1
876 !3          -0.31250000000000000000000000000000 *x^2
877 !4           0.06250000000000000000000000000000 *x^3
878 !Y(0) =  (((X(0)-5._16)*x(0)+15._16)*x(0)+5._16)/16._16
879
880 ! real(16)  ::Xval(-9:18)  & !These are mantissa testcase values of ValIn.
881 !  =(/  .1_16, .2_16, .3_16, .4_16, .5_16, .6_16, .7_16, .8_16, .9_16 ,1._16  &
882 !     ,1.1_16,1.2_16,1.3_16,1.4_16,1.5_16,1.6_16,1.7_16,1.8_16,1.9_16         &
883 !           ,2.0_16,3.0_16,4.0_16,5.0_16,6.0_16,7.0_16,8.0_16,9.0_16, 10._16  /)
884 !Mantissa processing:
885 !  i      Y(1)*Y(1)-X(0)=one-iteration error    X(0) For:
886 ! -9      0.06256016015625000000000000000000     0.1   even order exponents:
887 ! -8      0.03814400000000000000000000000000     0.2
888 ! -7      0.02184347265625000000000000000000     0.3
889 ! -6      0.01152224999999999999999999999999     0.4
890 ! -5      0.00543212890625000000000000000000     0.5
891 ! -4      0.00217600000000000000000000000001     0.6
892 ! -3      0.00067362890625000000000000000001     0.7
893 ! -2      0.00013025000000000000000000000001     0.8
894 ! -1      0.00000797265624999999999999999999     0.9
895 !  0      0.00000000000000000000000000000000     1.0
896 !                  1         2         3
897 !            12345678901234567890123456789012
898 !                                             For:
899 !  1      0.00000766015624999999999999999999     1.1   odd order exponents:
900 !  2      0.00012024999999999999999999999999     1.2   forced to even order
901 !  3      0.00059769140625000000000000000001     1.3   in the 1st iteration
902 !  4      0.00185600000000000000000000000006     1.4   by Y(1) = sqrt(10)
903 !  5      0.00445556640625000000000000000000     1.5
904 !  6      0.00909224999999999999999999999996     1.6
905 !  7      0.01659128515624999999999999999999     1.7
906 !  8      0.02790399999999999999999999999996     1.8
907 !  9      0.04410734765625000000000000000001     1.9
908 ! 10      0.06640625000000000000000000000000     2.0
909
910 !           3rd-order correction errors for X(0)>2.  :
911 ! 11      1.00000000000000000000000000000000     3.0
912 ! 12      5.37890625000000000000000000000000     4.0
913 ! 13     20.00000000000000000000000000000000     5.0
914 ! 14     61.03515625000000000000000000000000     6.0
915 ! 15    162.00000000000000000000000000000030     7.0
916 ! 16    384.53515625000000000000000000000110     8.0
917 ! 17    831.99999999999999999999999999999940     9.0
918 ! 18   1665.87890625000000000000000000000000    10.0
919 !-----
920
921 !Polynomial  coefficients C :  5th order
922 !1           0.24609375000000000000000000000001 *x^0
923 !2           1.23046875000000000000000000000000 *x^1
924 !3          -0.82031250000000000000000000000000 *x^2
925 !4           0.49218750000000000000000000000000 *x^3
926 !5          -0.17578125000000000000000000000000 *x^4
927 !6           0.02734375000000000000000000000000 *x^5
928
929 !Y =  ((((( +0.0273437500_16  *X &
930 !            -0.1757812500_16) *X &
```

```
!           +0.4921875000_16) *x &
!           -0.8203125000_16) *x &
!           +1.2304687500_16) *x &
!           +0.2460937500_16)

!  i         Y(1)*Y(1)-X(0)=one-iteration error    X(0)
! -9         0.042550491714477539062500000000000     0.1
!convergence is rapid at 3rd order - hence little improvement:
!-----
!Polynomial  coefficients C :  6th order
!1           0.225585937499999999999999999999999 *x^0
!2           1.353515625000000000000000000000001 *x^1
!3          -1.127929687500000000000000000000000 *x^2
!4           0.902343750000000000000000000000000 *x^3
!5          -0.483398437500000000000000000000000 *x^4
!6           0.150390625000000000000000000000000 *x^5
!7          -0.020507812500000000000000000000000 *x^6

!  i         Y(1)*Y(1)-X(0)=one-iteration error    X(0)
! -9         0.022859847178638802528838134765624     0.1
!convergence is rapid at 3rd order - hence little improvement:
!-------------------------------------------------------------------------------7 9

Function Factrl(N) Result(Out)  !<-integer(8)
!2024.07.30.1135cdt JMS- Factorial Computation using a function.
!                        13!=6,227,020,800  > integer(4)
!                        20!=2.432902008e18 < integer(8) is the upper limit.
implicit none                                                          !arguments
  integer(4)::N
  integer(8)::Out
 !--                                                                   !internals
  integer(4)::i
  !-----

   if(N.ge.0) goto 5
     stop 'Factorl: N < 0 '
 5 Out=1_8; if(N.lt.2) return
   do i=2,N; Out=Out*i; end do; return
End Function Factrl
!Used in a print statement:
! integer(8)::Factrl
!   !Test of function Factrl():  2024.07.30:
!    if(iP>5) write(iP,"('Factrl(13) =',i20)") Factrl(13)  <--
!                         Factrl(13) =          6227020800

!2025.04.21.1450cdt JMS- Fortran real(16)'s have ~ a 102-bit mantissa;
!                       hence 2^102 = 5.070602401e30 < 29!
!                       would be the upper limit of mantissa-exact factorials
!                       without more clever coding.
!                       69! = 1.711224524e98
!-------------------------------------------------------------------------------7 9

!End Module hVecMath16Mod
!-------------------------------------------------------------------------------7 9

```