```
 !TweakEngine.F95  Version 1.0
!2018.10.08.0945cdt JMS- Applied numerical partial differentiation.

!  Jeffrey M. Setterholm, 8095 230th St. E., Lakeville, Minnesota 55044, USA
!   I have authored the four Fortran *.f95 source code files listed below.
!                       I hereby place these four files:
!       Tweak-Begin.f95, Tweak-Engine.F95, Tweak-User.F95, & Tweak-Vis.f95
!               and the algorithms which are demonstrated therein,
!                     in the public domain (a.k.a.: "free").
!Disclaimer:
!       ***************************************************************
!       **********   Individual cognition is always flawed,  **********
!       **********         including yours and mine.         **********
!       **********                  - So: -                  **********
!       **********      Use this code at your own risk.      **********
!       ***************************************************************

!Table of Contents:
!Program Tweak
!Subroutine EvalFit(RSSL,iP)
!Subroutine PrintIter(iP)
!Subroutine RSSPartials(jpUL)
!Subroutine DatapointPartials
!Subroutine Invert(N,A,ValMin,iRank,DetN,iUsed,iP)
!Subroutine PrintA(N,A,Noise,iRank,DetN,iRu,jCu,iP)
!Subroutine SelectStepMult(iP)
!Function om(Value1,Value0,iP)
!Subroutine FloatWrite(R16In,a40out)
!Subroutine FDate23(DaTime)
!Subroutine Beamer(n,nTot)
!----------------------------------------------------------------------------7-9

Program Tweak
!2018.10.09.0820cdt JMS- Empowers numerical partial differentiation to fit
!                          the parameters of your model(s) to your datae
!                                   with extreme accuracy.
!                         "Tweaks" refers to the small deltas of parameter
!                                values used to compute the numerical partials.
!                      - Traveler2/Athlon64/WinXPPro-32/APF9.0-32
!---                                                                  globals
!Defined in Tweak-Begin.f95; all named here, as an overview:
 use Tweakrec,only: jPhase,jMode  ,cVersion,cDateTime & !Tweak's FYI
                   ,jBOn,jUnClamp,jStepMult,jPrev     & !Solution strategy
                   ,jpU10,jpU,jpD           ,cFloat40 & !Printout-Alphanumeric
                   ,jpU3d,TLrec,TL,TLprev,TL2,TLsave  & !Printout-3D
                   ,jTLmorph,TLiter,omjSave           & !         "
                   ,jItertot,jIter,jDone              & !Iteration control
                   ,RSS,RSSbase,Weight,Delta,offon    & !Tweaking & errors
                   ,AbsDet,NoiseFloor,iRank,kPChanged & !Inverter outputs
                   ,jMintot,jMin,StepMult,omj         & !Minimization passes !Minim
                   ,B,BtB,BtZ                         & !Allocated matrices
                   ,What,Why,How,Who,When,Where1      & !Project context
                   ,jUserPhase,jUserConfig,cjUserFile & !Use in YouTweak()
                   ,TweakNml                            !Runtime reconfig.
 Use KPrec,   only: Kptot,Kp,Kp2,  Pr   ,Pu,PstepMag    !Parameters -to fit to-
 Use LZrec,   only: Lztot,Lz,      Zr   ,Zu,Zu2,Z0      !Outputs    - of your -
 use MDrec,   only: Mdtot,Md,MdMax,Datae                !Dataset
 use NXrec,   only: Nxtot,Nx,      Xr                   !Independent variables
!                                  ^^:These are allocated records.
!                    ^^:At the outset you size these four values.
 !---
 implicit none                                                       !arguments
  !---                                                               !internals
  integer*4:: i,j
  integer*4:: nY
  integer*4:: jpUL       !Printer argument of Invert() and SelectStepMult()
  character:: A1*1
 !real*16  :: RSSL
  logical  ::LExists       !File existence flag

```

```
   !-----                                                            !end defs
  !call fdate(cDateTime)
   call FDate23(cDateTime)
!  --------------------------
   jPhase = "0: Allocates records based on Kptot,Lztot,Mdtot,& Nxtot"c

   call YouTweak(0) !You must size Kptot,Lztot,Mdtot, & Nxtot!

!  Modifying key variables at runtime using NameList:
   inquire(file="Tweak-user.nml",exist=LExists)
   if(LExists)then
     write(6,"('Tweak-user.nml variable changes included...')")
     open(unit=jpU10,file  ='Tweak-user.nml',action='read' &
         ,access='sequential'  ,status='old' ,err=11)
        read(jpU10,nml=TweakNml,err=11)
     close(jpU10); goto 12
 11  pause "Error reading Tweak-user.nml. Press enter to continue."
     close(jpU10)
    endif!LExists
 12 continue

   call TweakAllocateAll !Allocates all but the inverter's internal arrays.

   if(jpU < 6) jpD=0       !No output file: no printouts.
               jpUL = jpU
   if(jpD < 6) jpUL = 0

   if(Weight > 0._16) Zr.Wt    = Weight
                      Pr.offon = offon

!  Using previous values of the parameters:
   if(jPrev == 0) then
 20  write(6,"(1x,'Use the last run`s Pr.Pbase values? (0 or 1)'\)");
     read(5,*,err=20) jPrev
   endif !jPrev = 0
   if(jPrev.eq.1) then  !Import the previous values from "Tweak-params.txt":
     inquire(file="Tweak-params.txt",exist=LExists)
     if(LExists)then
       open(unit=jpU10,file  ='Tweak-params.txt',action='read' &
         ,access='sequential'  ,status='old' ,err=21)
         read(jpU10,*,err=21) Kptot,Lztot
         do Kp=1,Kptot; read(jpU10,*) Pr(Kp).P ;enddo!Kp
       close(jpU10); goto 22
 21      pause "Error reading Tweak-params.txt. Press enter to continue."
         close(jpU10)
      else
       pause "jPrev==1, but Tweak-params.txt not found. 'Enter` to continue."
     endif!Lexists
   endif !jPrev==0
 22 continue

!  Open the output file:
   if((jpU >10).and.(jpD > 0)) &
     open(unit=jpU,file='Tweak-out.txt',action='write')
   if(jpD>0) then;   if(jpU>6) write(  6,"(a60/)") cVersion
                               write(jpU,"(a60)") cVersion
                               write(jpU,"(/'Project Context:')")
     j=len_trim(What)  ;if(J>0)write(jpU,"(60a1)") (What(i:i)  ,i=1,j)
     j=len_trim(Why)   ;if(J>0)write(jpU,"(60a1)") (Why(i:i)   ,i=1,j)
     j=len_trim(How)   ;if(J>0)write(jpU,"(60a1)") (How(i:i)   ,i=1,j)
     j=len_trim(Who)   ;if(J>0)write(jpU,"(60a1)") (Who(i:i)   ,i=1,j)
     j=len_trim(When)  ;if(J>0)write(jpU,"(60a1)") (When(i:i),  i=1,j)
     j=len_trim(Where1);if(J>0)write(jpU,"(60a1)") (Where1(i:i),i=1,j)
   endif!(jpD>0)
!  --------------------------
   jPhase = "1: You quantify your system model in the allocated arrays"c

   call YouTweak(1)
!  Key Tweakrec values:
   if(jpD >= 1) then
```

```
     write(jpU,"(/'Tweak`s initial values, after calling YouTweak(1):')")
     write(jpU,"( ' jBOn     =',i6,6x,' = [     1: ]' )") jBOn
     write(jpU,"( ' jUnClamp =',i6,6x,' = [     1: ]' )") jUnClamp
     write(jpU,"( ' jStepMult=',i6,6x,' = [     1,2]' )") jStepMult
     write(jpU,"( ' jPrev    =',i6,6x,' = [-1,0,1  ]' )") jPrev

     write(jpU,"(/' jpU      =',i6,6x,' = [   0,6,11]')") jpU
     write(jpU,"( ' jpD      =',i6,6x,' = [   0-8   ]')") jpD    !Details
     write(jpU,"( ' jpU10    =',i6,6x,' = [       10]')") jpU10
     write(jpU,"( ' jpU3D    =',i6,6x,' = [   0  ,13]')") jpU3D

     write(jpU,"(/' jItertot =',i6,6x,' = [   0:    ]')") jItertot

     write(jpU,"(/' Weight   =',e12.3,' = [ nom: 1.] ')") Weight
     write(jpU,"( ' Delta    =',e12.3,' = [small>0.] ')") Delta
     write(jpU,"( ' offon    =',e12.3,' = [  0., 1.] ')") offon

     write(jpU,"(/' jMintot  =',i6,6x,' = [   1:    ]')") jMintot

     write(jpU,"(/' Kptot    =',i6,6x,' = [   1:    ]')") Kptot
     write(jpU,"( ' Lztot    =',i6,6x,' = [   1:    ]')") Lztot
     write(jpU,"( ' Mdtot    =',i6,6x,' = [   1:    ]')") Mdtot
     write(jpU,"( ' Nxtot    =',i6,6x,' = [   1:    ]')") Nxtot
   endif !jpD>=1

!  --------------------------
   jPhase = "2: Top-of-each iteration: establish baseline residual errors"c

   do jIter = 0,jItertot !Iterate the solution:
     if((jpU > 5).and.(jpD >=4)) write(jpU,"(/'Iteration:',i3)") jIter

     Zr.RMS   = 0._16
     Zr.Zmax  = 0._16
     Zr.Mdmax = 0
        Md    = 0
     call YouTweak(2)  !(Re-)Initialize your dataset
    !Pr.Pbase - holds the baseline parameter values for each iteration.
     Pr.P = Pr.Pbase

     if(Delta  > 0._16) Pr(1:Kptot).Delta=Delta  !Numerical tweak deltas
     if((jpD >= 5).and.(Delta > 0._16)) write(jpU,"('Delta=',e12.6)") Delta
     if(Weight > 0._16) Zr(1:Lztot).Wt =Weight !Output weights

!    Establish the base values at the start of each iteration:
     call EvalFit(RSS, 0)
     if((jIter == 0).and.(jpD >=4)) call PrintIter(jpU)
     RSSbase=RSS; Zr.Zbase=Zr.Z; Zr.RMSbase=Zr.RMS     ;if(jIter == 0) goto 700

!    jIter>0:

!    Compute the numerical partial derivatives two ways:
!  --------------------------

     call RSSPartials(jpUL) !Computes Pr(*).Pstep(1,1)'s
!  --------------------------

     call DatapointPartials !Computes Pr(*).Pstep(2,1)'s
!      Inside the call:
       jPhase = "3: Populate BtB(Kptot,Kptot) and BtZ(Kptot)"c
       jPhase = "4: Normalize and invert BtB(Kptot,Kptot)"c
       jPhase = "5: Inverters stepsize = BtBinv(denormalized)*BtZ"c
!  --------------------------

!    By each of the paths above - Pr(1:Kptot).Pstep's have been computed.
!    The .Pstep's are the Tweak-Engine's guess(es) about the vector direction
!    in which to change your parameter values so that your mathematical model
!    better-fits your dataset. When using jBon=1 on a linear system model
!    the length of the vector is just about right on.

!    My experience has been that non-linear models are locally linear around
```

```fortran
!    their solution point; hence they routinely converge very rapidly
!    in the last few iterations of a highly-accurate fit. In calibrating
!    sensors, model fits with less than .01% error are the norm, rather than
!    the exception.

!    The computations in SelectStepMult() search for a value of StepMult that
!    significantly reduces the fit error for non-linear problems.

!    For non-linear models the opening .Pistep vector may be huge.
!    Restricting the magnitude of parameter changes may prevent having
!    your model blow up before nears the stable region where it fits your data:

     do Kp2=1,2
       if(jpD >= 5) then
         if(Kp2 == 1) write(jpU,"(/'Pr.Pstep(1,1) RSS values:'\)")
         if(Kp2 == 2) write(jpU,"(/'Pr.Pstep(2,1) Inv values:'\)")
       endif!jpD>=5
       Pr.Pstep(Kp2,2) = Pr.Pstep(Kp2,1)
       if(jIter < jUnClamp) then
                        PstepMag = 0._16
         do Kp=1,Kptot; PstepMag = PstepMag +Pr(Kp).Pstep(Kp2,1)**2; enddo!Kp
         if(PstepMag  > 1._16) then !Shorten Pr.P.Step to magnitude = 1.
           Pr.Pstep(Kp2,2)=Pr.Pstep(Kp2,1)/sqrt(PstepMag )
           if(jpD >= 5) write(jpU,"(' Clamped:'\)")
         endif!(PstepMag  > 1._16)
       endif!(jIter<=__)
       if(jpD >= 5) then
         write(jpu,*)
         do Kp = 1,Kptot
           call FloatWrite( Pr(Kp).Pstep(Kp2,2),cFloat40)
           write(jpU,"(a40,2x,16a1)") cFloat40,(Pr(Kp).Pname(i:i) &
                                  ,i=1,len_trim(Pr(Kp).Pname)      )
         enddo!Kp
       endif!jpD>5
     enddo!Kp2

!    Set flag jBOn=_ to:
     if(jIter <  jBOn) Pr.Pstep(4,1) =  Pr.Pstep(1,2) !Use RSS Partials
     if(jIter >= jBOn) Pr.Pstep(4,1) =  Pr.Pstep(2,2) !Use Inverter's Partials

!    --------------------------
     jPhase = "6: Find an error-reducing actual stepsize"c

!    Select/Compute StepMult:
     select case(jStepMult)
      case(1) !Use inverter's result; excellent for linear models.
                            StepMult = 1._16
       Pr.Pnew = Pr.Pbase + StepMult * Pr.PStep(4,2)

      case(2) !Use SelectStepMult(), esp. when using jBPath=2
       call SelectStepMult(jpUL)       !(0): no detail

      case default; pause"jStepMult /= [1,2]; press enter to halt."; stop

     end select!(jStepMult)
                                                   kPChanged = 0
     do Kp=1,kptot;
       if(Pr(Kp).Pnew /= Pr(Kp).Pbase) kPChanged = kPChanged + 1
     enddo!Kp
     if((kPChanged == 0).and.(jIter > 5)) jDone=1

     Pr.P = Pr.Pnew; call EvalFit(RSS, 0)
     if(jpD >= 4) call PrintIter(jpU) !Print the results.
     if(jIter < 101) omjSave(jIter) = omj
     Pr.Pbase = Pr.Pnew !Use the new baseline parameters.
!    --------------------------
     jPhase = "7: Iterative pass done. Your intervention opportunity:"c
!                                      E.g.: to adjust Delta & Pr().onOff's
 700 call YouTweak(5)
                     ;if(jDone == 0) cycle
     NY=2
     if(jpD >= 2) then
       call PrintIter(6)
 710   WRITE(6,"(1X,'Want to quit (N/n or Y/y)?'\)")
       A1=char(0)
       read(5,*) A1
       select case(ichar(A1))
        case(78,110);  NY=1   !Nn
        case(89,121);  NY=2   !Yy
        case default; goto 710
       end select !ichar(A1)
     endif !(jpD>=2)
                                                               if(NY.EQ.2) exit
   enddo!jIter
!    --------------------------
   jPhase = "8: Tweak is done interating. Export the results & close"c
   Pr.P=Pr.Pbase
   if(jpD == 3) call PrintIter(jpU)
   if(jpD >= 3) write(jpU,"(/'EvalFit final printout:')")
   call EvalFit(RSS,jpU)
   call YouTweak(6) !"Tweak-is-done": your completion-printout opportunity.

   if(jpD > 0) write(jpU,"('Tweak: saving results & deallocating arrays...')")

   if(jpU>10) close(jpU)

!  Save final values, e.g. to use as 'previous' parameters:
   call FDate23(cDateTime)
  !call fdate(cDateTime)
   open(unit=jpU10,file='Tweak-params.txt',action='write',access='sequential')
                write(jpU10,*) Kptot,Lztot
     do Kp=1,Kptot; write(jpU10,"(e41.32,i5,'  '\)") Pr(Kp).Pbase,Kp
       j=len_trim( Pr(Kp).Pname)
       if(J>0)write(jpU10,"(16a1)") (Pr(Kp).Pname(i:i),i=1,j)
     enddo!Kp
     write(jpU10,"(3x,'^These are the last-iteration`s values'\)")
     write(jpU10,"(   ' at full precision.'/)")

     do Kp=1,Kptot;  call FloatWrite(Pr(Kp).Pbase,cFloat40)
                write(jpU10,"(1x,i4,' ',a40)") Kp,cFloat40
     enddo!Kp
     write(jpU10,"(/13x,'.12345678901234567890123456789012345')")
     write(jpU10,"( 23x,'1         2         3'\)")
     write(jpU10,"(  6x,'-Floats decimal place counter')")

     Pr.P=Pr.Pbase; call EvalFit(RSS,jpU10)

     write(jpU10,"(/'Project Context:',40x,a23)") cDateTime
     j=len_trim(What)  ;if(J>0)write(jpU10,"(60a1)") (What(i:i)  ,i=1,j)
     j=len_trim(Why)   ;if(J>0)write(jpU10,"(60a1)") (Why(i:i)   ,i=1,j)
     j=len_trim(How)   ;if(J>0)write(jpU10,"(60a1)") (How(i:i)   ,i=1,j)
     j=len_trim(Who)   ;if(J>0)write(jpU10,"(60a1)") (Who(i:i)   ,i=1,j)
     j=len_trim(When)  ;if(J>0)write(jpU10,"(60a1)") (When(i:i),  i=1,j)
     j=len_trim(Where1);if(J>0)write(jpU10,"(60a1)") (Where1(i:i),i=1,j)
     write(jpU10,"(/a60)") cVersion
   close(jpU10)

   deallocate(Pr,Zr,Datae,Xr)
   deallocate(B,BtB,BtZ)
   if(jpU > 10) write(6,"('`Tweak-out.txt` has your run summary.' )")
                write(6,"('`Tweak-params.txt` has the parametric values.' )")
   pause 'Press `Enter` to exit.'
End Program Tweak
!---------------------------------------------------------------------------7-9
Subroutine EvalFit(RSSL,iP)
!2018.09.02.1220cdt JMS- Evaluates the residual error(s) of the dataset.
!                      - Traveler2/Athlon64/WinXPPro-32/APF9.0-32
 !---                                                                  globals
 use Tweakrec,only: jItertot,jIter                     & !Iteration control
                   ,jpU,jpD                 ,cFloat40   !Printout
```

```
Use KPrec,   only:               Pr                   !Parameters -to fit to-
Use LZrec,   only: Lztot,Lz,      Zr                   !Outputs     - of your -
use MDrec,   only: Mdtot,Md,MdMax,Datae                !Dataset
use NXrec,   only: Nxtot,Nx,      Xr                   !Independent variables
!---
implicit none                                                       !arguments
 real*16  ::RSSL
 integer*4::iP
 !---                                                               !internals
 real*16  ::temp
 integer*4::i
  !-----                                                            !end defs
  if(iP > 5) write(iP,"(/'EvalErr:')")
  RSSL=0._16
  Md = 0
  call YouTweak(2)

  Zr.Zmax  = 0._16
  Zr.Mdmax = 0
  Zr.RMS   = 0._16
     RSSL  = 0._16
  if(iP > 5) &
    write(iP,"('Data# Out#',9x,'Error  : results comparison',11x,'dataID')")

  do Md = 1,Mdtot
    call YouTweak(3)
    call YouTweak(4)
    do Lz=1,Lztot
                               temp = Zr(Lz).Z
      Zr(Lz).RMS  = Zr(Lz).RMS+temp*temp
     !if(jIter.le.jItertot) cycle
      if(Md == 1) then
             Zr(Lz).Zmax      = temp; Zr(Lz).Mdmax=1; endif!Md=1
      if(abs(Zr(Lz).Zmax) < abs(temp)) then
             Zr(Lz).Zmax      = temp; Zr(Lz).Mdmax=Md;endif
    enddo!Lz
  enddo!Md

  do Lz=1,Lztot
     Zr(Lz).RMS =sqrt(Zr(Lz).RMS/Mdtot)
     RSSL=RSSL + Zr(Lz).RMS*Zr(Lz).RMS
  enddo!Lz
  RSSL=sqrt(RSSL)

! Md = 0
! call YouTweak(2)
                                                  !if(jIter < jItertot) return
                                                   if(iP    <    6    ) return
  write(iP,"('EvalFit Summary:')")
                           call FloatWrite(RSSL,cFloat40)
  write(iP,"(  'Mdtot=',i7,a40,' SRSS')") Mdtot,cFloat40
  do Lz=1,Lztot;                                   Mdmax =Zr(Lz).Mdmax
                                   call FloatWrite(Zr(Lz).RMS ,cFloat40)
    write(iP,  "('  Lz= ',i7,a40,' RMS')")     Lz,             cFloat40
    if(MdMax > 0) then
                                   call FloatWrite(Zr(Lz).Zmax,cFloat40)
      write(iP,"(' @Md= ',i7,a40,' Max(abs)')")    Mdmax,      cFloat40
         Nx=0
        write(iP,"(i13,f10.0,31x,16a1)")Nx,Datae(Nx    ,Mdmax) &
                               ,(Xr(Nx).Xname(i:i),i=1,len_trim(Xr(Nx).Xname))
      do Nx=1,Nxtot
             call FloatWrite(Datae(Nx      ,Mdmax),cFloat40)
        write(iP,"(i13,a40,1x,16a1)")   Nx,          cFloat40   &
                               ,(Xr(Nx).Xname(i:i),i=1,len_trim(Xr(Nx).Xname))
      enddo!Nx
             call FloatWrite(Datae(NxTot+LZ,Mdmax),cFloat40)
      write(  iP,"(13x,a40,1x,16a1)")                cFloat40   &
                               ,(Zr(Lz).Zname(i:i),i=1,len_trim(Zr(Lz).Zname))
    endif !(McMax>0)
  enddo!Lz
```

```
   write(iP,"(49x,'^^^^: exponents, if any.')")
   write(iP,  "(i13,'  Iterations')") jIter;                              return
End Subroutine EvalFit
!----------------------------------------------------------------------------7-9

Subroutine PrintIter(iP)
!2018.09.26.1150cdt JMS- Using FloatWrite
!                      - Traveler2/Athlon64/WinXPPro-32/APF9.0-32
 !---                                                                  globals
 use Tweakrec,only:                         cDateTime & !Tweak's FYI
                   ,jBOn,jUnClamp,jStepMult           & !Solution strategy
                   ,jpU,jpD                 ,cFloat40 & !Printout
                   ,      jItertot,jIter,jDone        & !Iteration
                   ,RSS                               & !Tweaking & errors
                   ,jMintot,jMin,StepMult,omj           !Minimization passes
 Use KPrec,   only: Kptot,Kp,Kp2,  Pr                   !Parameters -to fit to-
 Use LZrec,   only: Lztot,Lz,      Zr                   !Outputs    - of your -
 use MDrec,   only: Mdtot,Md                            !Dataset
 !---
 implicit none                                                        !arguments
  integer*4::iP
  !---                                                                !internals
  real*16  ::change,delta
  integer*4::KpL
  character::C1*1
   !-----                                                             end defs
   if(jpD < 3) return
   call FDate23(cDateTime)
   if(jMin > jMintot) write(iP,"('Lower value of error not found.')")
   if(jIter == 0) write(iP,"(/35x,'om = an order-of-magnitude change scale.')")
   if(jIter == 0) write(iP,"(/'Stopwatch - 24-hour local time (msec val'\)")
   if(jIter == 0) write(iP,"( 'ues stick):     Year Mo D  Hr Mn Sec~ms')")
   write(iP,"(/   'Pass',i3,' of',i4,\)") jIter,jItertot
   if(jBOn  > jIter) write(iP,"('  RSS partials',\)")
   if(jBOn <= jIter) write(iP,"('  BtB & BtZ   ',\)")
   if(jUnClamp >  jIter) write(iP,"(' -clamped  ',\)")
   if(jUnClamp <= jIter) write(iP,"('           ',\)")
   write(iP,"('om = ',a9,3x,a23)") omj,cDateTime
                        call FloatWrite(RSS,cFloat40)
   write(iP,"('RSS = ',a40,e14.6,' /',i2)") cFloat40,RSS,jMin
   write(iP,"('Lz#   ',13x,'Residual Error',16x,'Change',8x,'weight')")
   do Lz=1,Lztot
     call FloatWrite(Zr(Lz).RMS,cFloat40)
     change=Zr(Lz).RMS-Zr(Lz).RMSbase
                         C1="*"
     if(abs(change) > 0) C1=" "
       write(iP,"(1x,i4,a1,a40,\)") Lz,C1,cFloat40
     if(abs(change) > .000001_16) then
       write(iP,"(1x,f17.8 ,2x,f13.9)") dble(change),dble(Zr(Lz).Wt)
      else;
       write(iP,"(1x,e17.10,2x,f13.9)") dble(change),dble(Zr(Lz).Wt)
     endif !(change not tiny)
   enddo!Lz
   write(iP,"('----- ----.12345678901234567890123456789012345')")
   write(iP,"(     '     *:unchanged    1         2         3'\)")
   write(iP,"(  9x,'-Floats decimal place counter'/)")

   write(iP,"('Kp#   ',17x,'Value',21x,'Change',9x,'delta')")
   do Kp=1,Kptot
     call FloatWrite(Pr(Kp).P,cFloat40)
     change = Pr(Kp).P-Pr(Kp).Pbase
     KpL    = Kp
     delta  = Pr(Kp).Delta

     if(abs(change) > 0) then
            write(iP,"(1x,i4,' ',a40,\)") KpL,cFloat40
      else; write(iP,"(1x,i4,'*',a40,\)") KpL,cFloat40
     endif!(|Change|>0)
     if(abs(change) > .000001_16) then
       write(iP,"(1x,f17.8 \)") dble(change)
      else;
       write(iP,"(1x,e17.10\)") dble(change)
     endif !(change not tiny)
     if(abs(delta) >= 1.e-9_16) then
       write(iP,"(2x,f13.9)") dble(delta)
      else;
       write(iP,"(2x,e13.7)") dble(delta)
     endif !(change not tiny)
   enddo!Kp
   write(iP,"('----- ----.12345678901234567890123456789012345')")
   write(iP,"(     '     *:unchanged    1         2         3'\)")
   write(iP,"(  9x,'-Floats decimal place counter'/)")
   write(iP,*)
   Md=0
   if((iP == 6).and.(jDone == 0)) pause "... by PrintIter"
                                                                         return
End Subroutine PrintIter
!---------------------------------------------------------------------------7-9

Subroutine RSSPartials(jpUL)
!2018.09.26.1150cdt JMS- Uses the partial derivatives of the entire dataset
!                          to compute a single gradient vector.
!                      - Traveler2/Athlon64/WinXPPro-32/APF9.0-32
 !---                                                                 globals
 use Tweakrec,only:     jpD                 ,cFloat40  &!Printout
                   ,RSS,RSSbase                         !Tweaking & errors
!                  ,B,BtB,BtZ                           !Allocated matrices
 Use KPrec,   only: Kptot,Kp,Kp2,  Pr   ,Pu,PstepMag    !Parameters -to fit to-
!Use LZrec,   only: Lztot,Lz,      Zr                   !Outputs    - of your -
!use MDrec,   only: Mdtot,Md,MdMax,Datae                !Dataset
 !---
 implicit none                                                       !arguments
  integer*4::jpUL
  !---                                                               !internals
  real*16  ::RSSL
  integer*4::i
   !-----                                                            !end defs
!  Reset the point from which numerical partial differentiation will occur:
     Pr.P     = Pr.Pbase              ;call EvalFit(RSS,jPUL)
!  Compute the numerical partial derivatives of each parameter vs. RSS:
   PstepMag  = 0._16
   do Kp = 1,Kptot !Tweak each of the parameters in turn:
     Pr(Kp).P = Pr(Kp).P+Pr(Kp).Delta ;call EvalFit(RSSL,jPUL) !Delta parameter
                                      !     Evaluates the entire dataset
!    Insert the numerical partial derivative into Pr(Kp).Pstep(1,1):
     Pr(Kp).Pstep(1,1) = (RSS-RSSL)/Pr(Kp).Delta
     PstepMag        = PstepMag  + Pr(Kp).Pstep(1,1) * Pr(Kp).Pstep(1,1)
     Pr(Kp).P        = Pr(Kp).Pbase         !un-Delta parameter
   enddo!Kp
     Pr.P            = Pr.Pbase             ;call EvalFit(RSS,jpUL)
   if(abs(PstepMag )  > 1.e-30_16) then
     PstepMag        = RSS/sqrt(PstepMag )
     Pr.Pstep(1,1)   = PstepMag  * Pr.Pstep(1,1)
   endif!(PstepMag  not tiny)


   if(jpUL >= 5) then
               call FloatWrite( PstepMag   ,cFloat40)
     write(jpUL,"('RSSPartials:  PstepMag =',a40)") cFloat40
     do Kp = 1,KpTot
       call FloatWrite( Pr(Kp).Pstep(1,1),cFloat40)
       write(jpUL,"(a40,2x,16a1)") cFloat40,(Pr(Kp).Pname(i:i) &
                               ,i=1,len_trim(Pr(Kp).Pname)      )
     enddo!Kp
   endif!(jpUL>=5)
                                                                         return
End Subroutine RSSPartials
!---------------------------------------------------------------------------7-9

Subroutine DatapointPartials
```

```
!2018.09.09.0735cdt JMS- Using the partial derivatives of individual datapoints
!                          to compute B(:,:) & hence BtB() and BtZ
!                      - Traveler2/Athlon64/WinXPPro-32/APF9.0-32
 !---                                                                  globals
 use Tweakrec,only: jPhase                                  & !Tweak's FYI
                   ,jpU,jpD                 ,cFloat40       & !Printout
                   ,AbsDet,NoiseFloor,iRank                 & !Inverter outputs
                   ,B,BtB,BtZ                                 !Allocated matrices
 Use KPrec,   only: Kptot,Kp,Kp2,  Pr                         !Parameters -to fit to-
 Use LZrec,   only: Lztot,Lz,      Zr                         !Outputs    - of your -
 use MDrec,   only: Mdtot,Md,MdMax,Datae                      !Dataset
 !---
 implicit none                                                        !arguments
  !---                                                                !internals
  integer*4:: i
  integer*4:: jpUL        !Printer argument of Invert() and SelectStepMult()

   !-----                                                             !end defs
             jpUL = jpU
   if(jpD<6) jpUL=0

!  Zero the matrix accumulators:
   BtZ         = 0._16
   BtB         = 0._16
!  Populate the key matrixes using the full dataset:
           Md = 0
   call YouTweak(2)  !(Re-)Initialize your dataset
   do Md = 1,Mdtot !For each dataset:
               if(MdTot > 1000) &
     call beamer(Md,Mdtot) !DOS-screen progress bar - mitigates angst
                                               ! while huge datasets process.
     call YouTweak(3) !Pull in the next datapoint
     call YouTweak(4) !Compute outputs for the current datapoint
     Zr.Zbase = Zr.Z  !Compute each outputs un-tweaked value

!    Compute the numerical partial derivatives of each parameter:
     do Kp = 1,Kptot !Tweak each of the parameters in turn:
       Pr(Kp).P = Pr(Kp).P+Pr(Kp).Delta !   Delta parameter
       call YouTweak(4)                 !Evaluate the outputs

!      Insert the numerical partial derivative into B():
       B(1:Lztot,Kp)=((Zr.Zbase-Zr.Z)/Pr(Kp).Delta)*Pr(Kp).offon
       Pr(Kp).P = Pr(Kp).Pbase         !un-Delta parameter
     enddo!Kp

     if(jpD >= 5) then  !Print B()
       write(jpU,"(/'Md# = ',i16/'Lz# ='\)")Md
       do Lz=1,Lztot;   write(jpU,  &
                "(i5,4x,g12.6,' Zr( Lz        ).Zbase'/'  '\)") Lz,Zr(Lz).Zbase
         do Kp=1,Kptot; write(jpU,"(g12.6\)") B(Lz,Kp)              ;enddo!Kp
                        write(jpU,"(' B(  Lz,1:Kptot)'/'  '\)")
         do Kp=1,Kptot; write(jpU,"(g12.6\)") B(Lz,Kp)*Zr(Lz).Zbase ;enddo!kp
                        write(jpU,"(' BtZ(   1:Kptot)')")
       enddo!Lz
     endif!((jpU>5)&(jpD>6))

!    Accumulate B() into -BtZ() and BtB:
     do Lz=1,Lztot
       do Kp=1,Kptot      !Vector add:
         BtZ(Kp)=BtZ(Kp)+B(Lz,Kp)*Zr(Lz).Zbase
         do Kp2 = 1,Kptot !Outer product of B() with itself (~covariance):
           BtB(Kp2,Kp) = BtB(Kp2,Kp) +    B(Lz,Kp2)  *  B(Lz,Kp)
         enddo!Kp2
       enddo!Kp2
     enddo!Lz
   enddo!Md
!  BtB(), BtZ(), and BnZ() have been computed.

!  ------------------------

```

```fortran
   jPhase = "4: Normalize and invert BtB(Kptot,Kptot)"c

   if(jpD >= 6) then  !Print BtB() & BtZ()
     write(jpU,"(/'Kp#   BtB(Kptot,Kptot) | BtZ(Kptot)=  (normalized)')")
     do Kp=1,Kptot;    write(jpU,"(i4,'  ',\)") Kp
       do Kp2=1,Kptot; write(jpU,"(     g18.9\)") BtB(Kp,Kp2); enddo!Kp2
                       write(jpU,"(' |',g18.9 )") BtZ(Kp)
     enddo;            write(jpU,"('')")
   endif!((jpU>5)&(jpD>=6))

!  Pr.BtBnorm will be used to normalize BtB() with 1.'s on the diagonal:
   do Kp=1,Kptot; Pr(Kp).BtBnorm = abs(BtB(Kp,Kp))
               if(Pr(Kp).BtBnorm > 1.e-20_16) then !This threshold is a wag,
!                                                             ...explore it.
                  Pr(Kp).BtBnorm = sqrt(Pr(Kp).BtBnorm)
                else
                  Pr(Kp).BtBnorm = 1._16
               endif!Pr(Kp).BtBnorm not tiny
     if((jpU > 5).and.(jpD >= 6)) &
       write(jpU,"('Pr(',i4,').BtBnorm=',e13.6)") Kp,Pr(Kp).BtBnorm
   enddo !Kp

!  Pr.BtBnorm = 1._16 !<- Disables BtBnorm normalization.

!  Normalize BtB()
   do Kp=1,Kptot
     do Kp2=1,Kptot
       BtB(Kp,Kp2) = BtB(Kp,Kp2)/(Pr(Kp).BtBnorm*Pr(Kp2).BtBnorm)
     enddo!Kp2
   enddo!Kp

   if(jpD >= 5) then  !Print BtB() & BtZ()
     write(jpU,"(/'Kp#   BtB(Kptot,Kptot) | BtZ(Kptot)=')")
     do Kp=1,Kptot;    write(jpU,"(i4,'  ',\)") Kp
       do Kp2=1,Kptot; write(jpU,"(     g18.9\)") BtB(Kp,Kp2); enddo!Kp2
                       write(jpU,"(' |',g18.9 )") BtZ(Kp)
     enddo;            write(jpU,"('')")
   endif!((jpU>5)&(jpD>=6))

   !---
   call invert( Kptot        &  !Input : dimensions of BtB(Kptot,Kptot)
               , BtB         &  !In/Out: matrix to be inverted
               , NoiseFloor  &  !Output: noise floor of the inversion
               , iRank       &  !        Rank of BtBinverse <= Kptot
               , AbsDet      &  !        abs(Determinant) of BtB
               , Pr.Inverted &  !        =0. linearly   dependent param.
                             &  !        =1. linearly independent param.
               , jpUL         ) !input : >5: Print unit number
                                  !        =0: No printout

!  --------------------------
   jPhase = "5: Inverters stepsize = BtBinv(denormalized)*BtZ"c

   do Kp=1,Kptot
     do Kp2=1,Kptot
       BtB(Kp,Kp2) =  BtB(Kp,Kp2)/( Pr(Kp).BtBnorm*Pr(Kp2).BtBnorm )
     enddo!Kp2
   enddo!Kp

   if(jpD >= 5) then  !Print BtBinv() & BtZ()
     write(jpU,"(/'Kp#   BtBinv(Kptot,Kptot) | BtZ(Kptot)=')")
     do Kp=1,Kptot;    write(jpU,"(i4,'  ',\)") Kp
       do Kp2=1,Kptot; write(jpU,"(     g18.9\)") BtB(Kp,Kp2); enddo!Kp2
                       write(jpU,"(' |',g18.9 )") BtZ(Kp)
     enddo;            write(jpU,"('')")
   endif!((jpU>5)&(jpD>=6))

   Pr.Pstep(2,1) = 0._16
   do Kp=1,Kptot     !Compute the step vector:
     do Kp2=1,Kptot
```

```
       Pr(Kp).Pstep(2,1) = Pr(Kp).Pstep(2,1) + BtB(Kp,Kp2) * BtZ(Kp2)
     enddo!Kp2
   enddo!Kp

   Pr.P = Pr.Pbase
   if(jpD >= 5) then
     write(jpU,"(/'Inverter`s full Pr(Kp).Pstep(2,1):')")
     do Kp = 1,KpTot
       call FloatWrite( Pr(Kp).Pstep(2,1),cFloat40)
       write(jpU,"(a40,2x,16a1)") cFloat40,(Pr(Kp).Pname(i:i), &
                               i=1,len_trim(Pr(Kp).Pname)        )
     enddo!Kp
   endif!jpD>=5
                                                                         return
End Subroutine DatapointPartials
!---------------------------------------------------------------------------7-9

Subroutine Invert(N,A,ValMin,iRank,DetN,iUsed,iP)
!2010.01.06.1205cst JMS- Traveler2/Athlon64/WinXPPro-32/APF9.0-32
!2006.05.25.0715cdt JMS- An instantiation of the "Setterholm Matrix Inverter"
!---                                                               No globals
 implicit none                                                       !arguments
  integer*4:: N
  real*16  :: A(N,N)
  real*16  :: ValMin
  integer*4:: iRank
  real*16  :: DetN    !Abs(determinant product) for each iteration
  integer*4:: iUsed(N)
  integer*4:: iP
  !---                                                               !internals
  integer*4,allocatable::iRu(:)
  integer*4,allocatable::jCu(:)
  real*16  ,allocatable::Temp(:)
  integer*4::i,iu,iu2,j,ju,ju2,L
  real*16  ::Amult,ValMax
  integer*8::iBinSum
  integer*4::iAlloc
   !-----                                                            !end defs
   if(iP.gt.5) write(iP,"('Overwriting Matrix Inversion:')")
   DetN=0._16 ;iUsed(1:N)=0 ;iRank=0
   !---
   allocate(iRu(N),stat=iAlloc)
   if(iAlloc.ne.0) stop 'Invert: iRu(N) allocation error. Halt.'
   iRu=0
   allocate(jCu(N),stat=iAlloc)
   if(iAlloc.ne.0) stop 'Invert: jCu(N) allocation error. Halt.'
   jCu=0
   allocate(Temp(N),stat=iAlloc)
   if(iAlloc.ne.0) stop 'Invert: Temp(N) allocation error. Halt.'
   Temp=0._16
   !---
     Temp=0._16; iRu   =0 ;jCu   =0 ;ValMin=0._16
   do i=1,N    ;iRu(i)=i ;jCu(i)=i                ;end do
           if(iP.gt.5) call PrintA(N,A,ValMin,iRank,DetN,iRu,jCu,iP) !*********
!  Inversion Loop:
   do L=1,N ;iU=0     ;jU=0     ;ValMax =0._16
     do i=1,N  ;if(iRu(i).lt.0) cycle
       do j=1,N ;if(jCu(j).lt.0) cycle
         if(ValMax.ge.abs(A(i,j))) cycle
           ValMax=abs(A(i,j)) ;iu=i ;ju=j
       enddo!j
     enddo!i
     if(L.eq.1) ValMin=ValMax*1.d-20  !Establishes the "noise floor".
     if(iU.eq.0) Exit
     if(ValMax.lt.ValMin) Exit
     iRank=iRank+1
     DetN=ValMax
     ValMax=A(iu,ju) ;iu2=jCu(ju) ;ju2=iRu(iu)
    !if(iP.gt.5) write(iP,"(3i3,f15.6)")  L,iu,ju,ValMax
     if(iP.gt.5) write(iP,"(1x,2i4,f15.9,'   :iu ju pivot, & value')") &
```

```
                                              iu,ju,        ValMax
     Temp(1:N)=A(1:N,ju)                            ;i=iRu(iu)
               A(1:N,ju)=A(1:N,ju2)                 ;  iRu(iu)=iRu(iu2)
                         A(1:N,ju2)=Temp(1:N)       ;          iRu(iu2)=-abs(i)
     Temp(1:N)=A(iu,1:N)                            ;j=jCu(ju)
               A(iu,1:N)=A(iu2,1:N)                 ;  jCu(ju)=jCu(ju2)
                         A(iu2,1:N)=Temp(1:N)/ValMax ;         jCu(ju2)=-abs(j)
     do i=1,N ;if(i.eq.iu2) cycle ;Amult=A(i,ju2)
       do j=1,N ;A(i,j)=A(i,j)-A(iu2,j)*Amult ;enddo!j
       A(i,ju2)=-Amult/ValMax
     enddo; A(iu2,ju2)=1._16/ValMax
           if(iP.gt.5) call PrintA(N,A,ValMin,iRank,DetN,iRu,jCu,iP) !*********
   enddo!L  Inversion loop- done.

   iBinSum=0
   do i=1,n
     if((iRu(i).lt.0).and.(i.lt.63)) iBinsum=iBinSum+2**(i-1)
      if(iRu(i).lt.0) iUsed(i)=1
!     Zero linearly-dependent rows and columns, if any:
      if(iRu(i).gt.0) A(i,1:N)=0._16
      if(jCu(i).gt.0) A(1:N,i)=0._16
   enddo!i
   if((iP >5).and.(iu == 0)) then
     write(iP,"('Linear dependency dealt with. Partial inverse results:')")
     write(iP,"(b63.2,' -Kp unused flag')") iBinSum !For the first 62 Kp's
                       call PrintA(N,A,ValMin,iRank,DetN,iRu,jCu,iP) !*********
   endif!iP>5 & iu==0
   if(iP.gt.5) write(iP,"('Overwriting Inverter- done.')")
   deallocate(iRu)
   deallocate(jCu)
   deallocate(Temp)
   return
End Subroutine Invert
!---------------------------------------------------------------------------7-9

Subroutine PrintA(N,A,Noise,iRank,DetN,iRu,jCu,iP)
!2018.09.04.1340cdt JMS- Prints Invert's progress.
!                      - Traveler2/Athlon64/WinXPPro-32/APF9.0-32
 !---                                                                No globals
 implicit none                                                       !arguments
  integer*4:: N
  real*16  :: A(N,N)
  real*16  :: Noise
  integer*4:: iRank
  real*16  :: DetN
  integer*4:: iRu(N)
  integer*4:: jCu(N)
  integer*4:: iP
  !---                                                                !internals
  integer*4::i,j
   !-----                                                             !end defs
                                                              if(iP < 6) return
   write(iP,"('Inverter: @ Rank =',i5,8x,'abs(Det)=',f39.30/2x\)") iRank,DetN
   do j=1,N;   write(iP,"(i18,\)") jCu(j) ; enddo; write(iP,"('')")
   do i=1,N;   write(iP,"(i5,' ' ,\)") iRu(i)
     do j=1,N; write(iP,"(f18.9\)") A(i,j); enddo; write(iP,"('')")
   enddo!i
   write(iP,"('')");                                                     return
End Subroutine PrintA
!---------------------------------------------------------------------------7-9

Subroutine SelectStepMult(iP)
!2018.10.03.1630cdt JMS- Rescales Pr.Pstep(4,1) to reduce Zr.Zbase error.
!                       & reports Pr.Pstep(4,2) = Pr.Pstep(4,1) * X(32)
!            Pr.Pnew = Pr.Pbase + Pr.Pstep(4,2)
!                      - X(32) is the multiplier with the smallest RSS Z error
!                      - Traveler2/Athlon64/WinXPPro-32/APF9.0-32
 !---                                                                  globals
 use Tweakrec,only:                         cDateTime & !Tweak's FYI
                             ,jpD           ,cFloat40 & !Printout-Alphanumeric
```

```
                                  ,jDone              & !Iteration control
                   ,RSS,RSSbase                       & !Tweaking & errors
                   ,jMintot,jMin,StepMult,omj           !Minimization passes
 Use KPrec,   only: Kptot,Kp,Kp2,  Pr      ,PstepMag    !Parameters -to fit to-
 Use LZrec,   only: Lztot,Lz,      Zr                   !Outputs    - of your -
 use MDrec,   only: Mdtot,Md                            !Dataset
 !---
 implicit none                                                       !arguments
  integer*4::iP
  !--                                                                 internals
  real*16,parameter:: Del =1.e-3_16 !Hard coded.
  real*16  :: X( 32)
  real*16  :: Z( 31)
  real*16  :: C(1:3,0:2)
  real*16  :: Delta
  real*16  :: Discrim
  integer*4:: i, iQ, k, nX, nX0, nXT
! real*16  :: Da
  real*16  :: B0,B1,S0,S1,S2,X0,X1,Z0,Z1
  character:: om*9
   !-----                                                             !end defs
   X( 1) =  -.1_16
   X( 2) =   .0_16
   X( 3) =         +Del
   X( 4) =         +Del*2._16
   X( 5) =   .1_16
   X( 6) =   .2_16
   X( 7) =   .3_16
   X( 8) =   .4_16
   X( 9) =   .5_16
   X(10) =   .6_16
   X(11) =   .7_16
   X(12) =   .8_16
   X(13) =   .9_16
   X(14) =  1.0_16
   X(15) =  1.0_16  +Del
   X(16) =  1.0_16  +Del*2._16
   X(17) =  1.1_16
   nX0    = 17 ! X( 1:17) are  canned  multipliers of Pr.Pstep(4,1)
   nXT    = 31 ! X(18:31) are computed multipliers of Pr.Pstep(4,1)
!              ! x( 32  ) is the multiplier used.

   do jMin = 1,jMintot
     Delta = 2._16**(jMin-1)
     if(iP.gt.0) write(iP,"(/'SelectStepMult minimization pass',i3,':')") jMin
     jDone = 0
     X(nX0+1:nXT+1) = 0._16
     Z              = 0._16
     C              = 0._16

     do i =1,nX0
       Pr.P = Pr.Pbase+X(i)*Pr.Pstep(4,1)/Delta; call EvalFit(Z(i), 0)
     enddo!i

!    The following two inverse quadratic matrix solvers are exact:
!    Quadratic at StepMult = 0._16, Del=.001_16,  AbsDet = 2.e-9 exactly.
     C(1,0) = + 500000._16*Z( 2) -1000000._16*Z( 3) +500000._16*Z( 4) !A0
     C(2,0) = -   1500._16*Z( 2) +   2000._16*Z( 3) -   500._16*Z( 4) !B0
     C(3,0) = +            Z( 2)                                      !C0
!    Quadratic at StepMult = 1._16, Del=.001_16,  AbsDet = 2.e-9 exactly.
     C(1,1) = + 500000._16*Z(14) -1000000._16*Z(15) +500000._16*Z(16) !A1
     C(2,1) = -1001500._16*Z(14) +2002000._16*Z(15)-1000500._16*Z(16) !B1
     C(3,1) = + 501501._16*Z(14) -1002000._16*Z(15) +500500._16*Z(16) !C1
!    Pass  1 of  40            om =  -26e.713           2018.10.03 14:25:27:750
!    RSS =    0.00000158425600535719021946731 72E-20 X(30) V intersect
!                        ^
!    In a linear testcase, less roundoff error did not result from using:
!    C(1,0) = (              Z( 2) -2.000_16*Z( 3)           +Z( 4) ) * 5.e5_16
!    C(2,0) = ( -3.000000_16*Z( 2) +4.000_16*Z( 3)           -Z( 4) ) * 5.e2_16
!    C(3,0) =                Z( 2)
!    C(1,1) = (   .500000_16*Z(14) -1.000_16*Z(15)  +.5000_16*Z(16) ) * 1.e6_16
!    C(2,1) = ( -1.001500_16*Z(14) +2.002_16*Z(15) -1.0005_16*Z(16) ) * 1.e6_16
!    C(3,1) = (  +.501501_16*Z(14) -1.002_16*Z(15)  +.5005_16*Z(16) ) * 1.e6_16
!    Pass  1 of  40            om =  -26e.713           2018.10.03 15:57:36:781
!    RSS =    0.0000015844125081052441839446760E-20 X(30) V intersect
!                         ^

!    Quadratic difference
     C(1:3,2) = C(1:3,1) - C(1:3,0)

     if(iP > 5) then
       write(iP,"(5x,'A',23x,'B',23x,'C')")
       do iQ=0,2
         write(iP,"(3e24.15\)") C(1:3,iQ)
         if(iQ == 0) write(iP,"(' @0.')")
         if(iQ == 1) write(iP,"(' @1.')")
         if(iQ == 2) write(iP,"(' dif')")
       enddo!iQ
     endif!iP>5

     do iQ=0,2; nX = nX0 + 4 * iQ
       if(abs(C(1,iQ)) > 1.e-28) then !Quadratic
         Discrim = C(2,iQ)*C(2,iQ) - 4._16*C(1,iQ)*C(3,iQ) != B*B - 4.*A*C
         if(Discrim >= 0._16) then
           X(nX+1)= (-C(2,iQ) -sqrt(Discrim)) / (2._16*C(1,iQ))
           X(nX+3)= (-C(2,iQ) +sqrt(Discrim)) / (2._16*C(1,iQ))
         endif!Discrim>=0.
         if(C(1,iQ) /= 0._16) &
           X(nX+2) =       -C(2,iQ)                 / (2._16*C(1,iQ))
        elseif(abs(C(2,iQ)) > 1.e-18) then
           X(nX+4) =       -C(3,iQ)                 / (2._16*C(2,iQ))
       endif!C(1,iQ)  not zero
     enddo!iQ

!    V Intersect: (Uses the slopes computed by the quadratic solvers.)
     X0 = X( 2); S0 =                C(2,0); Z0 = Z( 2); B0 = Z0-X0*S0
     X1 = X(14); S1 = 2._16*C(1,1) + C(2,1); Z1 = Z(14); B1 = Z1-X1*S1
                 S2 =                      Z1 - Z0
!    Error = 0. Intersect:
     if((S1-S0) /= 0._16) X(30) = (B0-B1)/(S1-S0)
     if(  S2    /= 0._16) X(31) =   -Z0  /  S2

     if(iP > 5)  &
       write(iP,"(' #',5x,'Xnom',3x,'X(#)used',11x,'om',8x,'Z(#)error')")

     do i = 1,nXT
       if(iP > 5) write(iP,"(i2,1x,f8.3\)") i,dble(X(i))
       if(X(i) < -2.0_16) X(i) = -2.0_16
       if(X(i) > +2.0_16) X(i) = +2.0_16
       if(iP > 5) write(iP,"(f19.15\)") dble(X(i))
       if((i > nX0) .and.(abs(X(i)) > 0._16)) then
         Pr.P = Pr.Pbase+X(i)*Pr.Pstep(4,1)*Delta; call EvalFit(Z(i), 0)
       endif !i>nX
       if(iP > 5) then
           write(iP,"(a9\)") om(Z(i),RSSbase,0)

           call FloatWrite(Z(i),cFloat40)
           write(iP,"(a40)")    cFloat40

         select case(i)
          case(17); write(iP,*) "M=0. Poly"
          case(21); write(iP,*) "M=1. Poly"
           case(25); write(iP,*) "Diff.Poly"
          case(29); write(iP,*) "V Intersect"
          case(30); write(iP,*) "0.Intersect"
          case(31); write(iP,*) "Using:"
         end select !(i)
       endif !iP>5
     enddo!i
                                                 K=2
     do i = 1,nXT; if(i == 2) cycle
       if( (X(i) /= 0._16) .and. (Z(i) < Z(K)) ) K=i
     enddo!i
     X(32)    = X(k)
     StepMult = X(k)
     if(K == 2) jDone=1
     Pr.P          = Pr.Pbase + StepMult * Pr.Pstep(4,1)
     Pr.Pstep(4,2) =          + StepMult * Pr.Pstep(4,1)

     if(iP > 5) then
                     call FloatWrite(Z(K),cFloat40)
       write(iP,"(i2,9x,f19.15,a9,a40)")  &
                   K,X(K),om(Z(k),RSSbase,0),cFloat40
     endif!(iP>5)

     call EvalFit(RSS, 0)
     omj     = om(Z(k),RSSbase,0)
     Pr.Pnew = Pr.P

     if(K == 3) jDone=1;                                      if(k.gt.1) return
   enddo!jMin
   if(iP > 5) write(iP,"(1x/1x,'lower value of error not found.')");     return
End Subroutine SelectStepMult
!---------------------------------------------------------------------------7-9
!Example X() values & om values solving Non-Linear Equation: Z=P1*X**P2
!                                   Set jpD=6 to printout this level of detail.
!            ... in this case V-intersect prevailed.

!SelectStepMult minimization pass  1:
!     A                      B                       C
!  -0.414516409594126E-01  -0.140501747823906E+01   0.140504987721336E+01 @0.
!   0.251661992428589E+01  -0.400476300816663E+01   0.195385476930220E+01 @1.
!   0.255807156524530E+01  -0.259974552992757E+01   0.548804892088839E+00 dif
! #     Xnom   X(#)used           om       Z(#)error
! 1   -0.100 -0.100000000000000  +1e.110  1.54514105931391141732820207662528000
! 2    0.000  0.000000000000000  +1e.1    1.40504987721336460651212773013003000
! 3    0.001  0.001000000000000  -0e.999  1.40364481828348458859671059585103000
! 4    0.002  0.002000000000000  -0e.998  1.40223967645032265185610978857108000
! 5    0.100  0.100000000000000  -0e.900  1.26414611570811272735039924459121100
! 6    0.200  0.200000000000000  -0e.799  1.12257883241864686388037827730726000
! 7    0.300  0.300000000000000  -0e.698  0.98084500816099882552929764258400300
! 8    0.400  0.400000000000000  -0e.598  0.84007207728405638394588855245573040
! 9    0.500  0.500000000000000  -0e.500  0.70262852517877158986036586780306200
!10    0.600  0.600000000000000  -0e.408  0.57351762090246646714465121384199720
!11    0.700  0.700000000000000  -0e.330  0.46350247302867487649693119255213000
!12    0.800  0.800000000000000  -0e.280  0.39401727368133816852907445587714210
!13    0.900  0.900000000000000  -0e.280  0.39293281899168267230187315871982200
!14    1.000  1.000000000000000  -0e.331  0.46571168542146961172869794433717400
!15    1.001  1.001000000000000  -0e.332  0.46674267888179905509175405851939340
!16    1.002  1.002000000000000  -0e.333  0.46777870558197707023862691272778000
!17    1.100  1.100000000000000  -0e.420  0.58966406517075711841952407757144900
! M=0. Poly
!18    0.972  0.972141384261836  -0e.313  0.43912669391825415641191665817935500
!19  -16.948 -2.000000000000000  +1e.322  4.52599394887466124352257057052209200
!20  -34.867 -2.000000000000000  +1e.322  4.52599394887466124352257057052209200
!21    0.000  0.000000000000000  - e.000  0.0000000000000000000000000000000E+00
! M=1. Poly
!22    0.000  0.000000000000000  - e.000  0.0000000000000000000000000000000E+00
!23    0.796  0.795663057722752  -0e.282  0.39578696731868254932219659803177730
!24    0.000  0.000000000000000  - e.000  0.0000000000000000000000000000000E+00
!25    0.000  0.000000000000000  - e.000  0.0000000000000000000000000000000E+00
! Diff.Poly
!26    0.299  0.299163738467439  -0e.699  0.98202848217066542510101889283740640
!27    0.508  0.508145582251970  -0e.492  0.69170390060797928150112681191491430
!28    0.717  0.717127426036501  -0e.319  0.44793855262103092187186490883597330
!29    0.000  0.000000000000000  - e.000  0.0000000000000000000000000000000E+00
! V Intersect
!30    0.809  0.808637610994461  -0e.278  0.39089020916738279307280679408463000
! 0.Intersect
!31    1.496  1.495787022704859  -0e.913  1.28235514056058195740042720651622400
```

```
! Using:
!30           0.808637610994461  -0e.278  0.39089020916738279307280679408463000

!Pass  5 of  40            om =   -0e.278                2018.10.03 13:59:49:906
!RSS =    0.39089020916738279307280679408463000  0.390890E+00 / 1
!Lz#                Residual Error               Change        weight
!    1    0.39089020916738279307280679408463000      -1.01415967    1.000000000
!----- ----.12345678901234567890123456789012345
!     *:unchanged    1         2         3        -Floats decimal place counter

!Kp#                    Value                    Change         delta
!    1    3.06447811636775589591704865441639100       0.13343492    0.000001000
!    2    2.20469619832500718803596272359361200      -0.02030151    0.000001000
!----- ----.12345678901234567890123456789012345
!     *:unchanged    1         2         3        -Floats decimal place counter

!Non-Linear Equation: Z=P1*X**P2 :
!om results, by iteration, were:                          Elapsed time: ~.07sec
!Pass  0 of  40           om = undefined                2018.10.03 13:59:49:890
!Pass  1 of  40           om =   -0e.995  RSS X(19)     2018.10.03 13:59:49:890
!Pass  2 of  40           om =   -0e.555  RSS X(18)     2018.10.03 13:59:49:906
!Pass  3 of  40           om =   -0e.115  RSS X(18)     2018.10.03 13:59:49:906
!Pass  4 of  40           om =   -1e.719      X(18)     2018.10.03 13:59:49:906
!Pass  5 of  40           om =   -0e.278      X(30)     2018.10.03 13:59:49:906
!Pass  6 of  40           om =   -1e.385      X(18)     2018.10.03 13:59:49:921
!Pass  7 of  40           om =   -2e.129      X(18)     2018.10.03 13:59:49:921
!Pass  8 of  40           om =   -5e.556      X(18)     2018.10.03 13:59:49:921
!Pass  9 of  40           om =   -7e.707      X(18)     2018.10.03 13:59:49:937
!Pass 10 of  40           om =  -11e.388      X(31)     2018.10.03 13:59:49:937
!                            =  -11e.370  for X(14): BtBinv*BtZ was a contender
!Pass 11 of  40           om =   -0e.394      X(13)     2018.10.03 13:59:49:937
!Pass 12 of  40           om =   -0e.768      X(17)     2018.10.03 13:59:49:953
!Pass 13 of  40           om =   -0e.891      X(30)     2018.10.03 13:59:49:953
!Pass 14 of  40           om =   -0e.370      X( 2)     2018.10.03 13:59:49:953
!Pass 15 of  40           om =   +1e.1                  2018.10.03 13:59:49:953
!RSS =    0.0000000002951647233157376886132E-20 0.295165E-29 / 1
!Lz#                Residual Error               Change        weight
!    1*   0.0000000002951647233157376886132E-20 0.0000000000E+00    1.000000000
!----- ----.12345678901234567890123456789012345
!     *:unchanged    1         2         3        -Floats decimal place counter

!Kp#                    Value                    Change         delta
!    1*   3.10000000000000000000000000000002000 0.0000000000E+00  0.1000000E-27
!    2*   2.20000000009999999999999999999995300 0.0000000000E+00  0.1000000E-27
!----- ----.12345678901234567890123456789012345
!     *:unchanged    1         2         3        -Floats decimal place counter

!Solving Linear Equation: log10(Z) = log10(P1)+log10(P2)*X :
!om results, by iteration, were:                         Elapsed time: ~.015sec
!Pass  0 of  40           om = undefined                2018.10.03 14:25:27:734
!Pass  1 of  40           om =  -26e.713 /1.4e26  X(30) 2018.10.03 14:25:27:750
!Pass  2 of  40           om =   -5e.170 /590000. X(14) 2018.10.03 14:25:27:750
!Pass  3 of  40           om =   -0e.957 /1.04    X(12) 2018.10.03 14:25:27:750
!Pass  4 of  40           om =   -0e.997          X(12) 2018.10.03 14:25:27:765
!Pass  5 of  40           om =   +1e.1                  2018.10.03 14:25:27:765
!                                                     ^: best X multiplier
!Numerical roundoff errors degrade locating analytical minima at the extreme
!  lower end. In this example, multiplier X(12) = .8 prevailed.

!In pass 1 the V Intersect edged out the Inverter solution by a factor of 2,
!  but X(14) = 1.0 produced om = -25e.144, and V Intersect was operating solely
!  along the Inverter's Pr(1:2).Pstep(2,1) direction.
!  The direction was about +77.4 degrees (CW)  for a distance of ~ 2.254e0.
!RSS =    0.0000015842560053571902194673172E-20  X(30)

!To get the pass 2 om = -5e.170,
!  the direction was about -49.4 (CCW) degrees for a distance of ~ .85e-25.
!RSS =    0.0000000000026863788799753185686E-20  X(14)

!X(12)'s two poke-&-hopes in iterations 3 & 4 yielded a minor improvement:
```

```
!RSS =    0.0000000000025707810391102583998E-20  X(12)
!The granularity of the least significant bits is evident.

!The Fortran environment being use here is 32-bit, so real*16's are 128 bits.
!Using a 64 bit environment with 256-bit quad precision reals will improve the
!potential accuracy of these  results by an additional om ~= -32e.100

!Real measurements are usually noisy, with noise floors far above om = -18e.100

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

Function om(Value1,Value0,iP)
!2018.10.03.1245cdt JMS- My "Order of Magnitude" scale.
!                      - Traveler2/Athlon64/WinXPPro-32/APF9.0-32
!The objective: to quickly grasp large numeric order-of-magnitude changes.

!om   = Function omScale(Value1,Value0,iP)
!Expresses the value1/Value0 ratio of positive numbers in exponential form, &
! places the signed exponent on the left side of the result.
!Exponents > 0 represent ratios >= 1.
!Exponents <=0 represent ratios <  1.
!1. > mantissa >= .1, except when Value1 = 0.
!                           (Actual arguments are "_16" quad-precision values.)
!Examples:    Value1    Value0  = Value1/Value0  = om character expression
! omScale(    6.5   ,   4.32 ,_) = .150462963e+1  = "  +1e.150"
! omScale(    6.5   ,  43.2  ,_) = .150462963e+0  = "  -0e.150"
! omScale(    6.5e5 ,  43.2e1,_) = .150462963e+4  = "  +4e.150"
! omScale(   43.2e1 ,   6.5e5,_) = .664615e-3      = "  -3e.665"
! omScale(   43.2   ,   6.5  ,_) = .664615e+1      = "  +1e.665"
! omScale(    6.5e12,  43.2e1,_) = .150462963e+11 = " +11e.150"
!                                                   "__sme.nnn" <-9 characters
!                                                   "_smme.nnn"   "
!Special cases:                                     "__s_e.***"   "
! omScale(    6.5   ,   6.5  ,_) = .100000e+1      = "  +1e.1  "  equal
! omScale(   -6.5   ,   4.32 ,_) = .100000e+1      = "    e.err"  outside range
! omScale(    0.0   ,   0.0  ,_) = .000000e+0      = "    e.nul"  both zero
! omScale(    0.0   ,   4.32 ,_) = .000000e+0      = "  - e.000"  Value1=0.
! omScale(    6.5   ,   0.0  ,_) = .000000e+0      = "  + e.inf"  Value0=0.
 !---                                                                No globals
 implicit none                                                       !arguments
  real*16  :: Value1  !Value- ending/current                                >0.
  real*16  :: Value0  !     - beginning/reference/previous                  >0.
  character:: om*9    !     - Value1/Value0 measured in omScale units.
  integer*4:: iP
  !---                                                               !internals
  real*16  :: Ratio
  integer*4:: eFactor
  real*16  :: Mantissa
   !-----                                                             !end defs
   if((Value1 > 0._16).and.(Value0 > 0._16)) then
     Ratio    = Value1/value0
     eFactor  = floor(log10(Ratio)+1.e-30) + 1
     Mantissa = Ratio/(10._16**eFactor)
     if(Mantissa > .999_16) Mantissa = .999_16
     if(eFactor > 9) then
       write(om,"(' +' ,i2,'e',SS,f4.3)") efactor,Mantissa
      elseif(eFactor > 0) then
       write(om,"('  +' ,i1,'e',SS,f4.3)") efactor,Mantissa
      elseif(eFactor > -10) then
       write(om,"('  -' ,i1,'e',SS,f4.3)") abs(efactor),Mantissa
      else
       write(om,"(' -' ,i2,'e',SS,f4.3)") abs(efactor),Mantissa
     endif!eFactor>9
     if(abs((Value1-Value0)/Value0) <1.e-28_16 )  om= "  +1e.1  ";goto 10
   endif! Value1>0. & Value0>0.
   if((Value1 <0._16).or. (Value1 =="NAN")) then; om= "    e.err";goto 10;endif
   if((Value0 <0._16).or. (Value0 =="NAN")) then; om= "    e.err";goto 10;endif
   if((Value1==0._16).and.(Value0 ==0._16)) then; om= "    e.nul";goto 10;endif
   if(Value1 ==0._16)                       then; om= "  - e.000";goto 10;endif
   if(Value0 ==0._16)                       then; om= "  + e.inf";goto 10;endif
```

```
!  Conversion error:
                                                  om= "  + e.unk"
10 continue                                         ;         if(iP < 6) return
   write(iP,"(/a9,' = omScale(',e20.12,',',e20.12,',', i2,')')") &
              om               ,Value1,   Value0 ,   iP
   write(iP,"(20x,e20.12,i5,e20.12/)") Ratio,eFactor,Mantissa
                                                                         return
End Function om
!---------------------------------------------------------------------------7-9

Subroutine FloatWrite(R16In,a40out)
!2013.10.23.1300cdt JMS- converts r16's to a 41-character-float string.
!                      - Traveler2/Athlon64/WinXPPro-32/APF9.0-32
 !---                                                                No globals
 implicit none                                                       !arguments
  real*16  ::R16In
  character::a40out*40
  !---                                                               !internals
  real*16  ::Real16InL
   !---
   Real16InL=R16In
   if(      abs(R16In).gt. 1.e30_16) then; write(a40out,"(e40.32   )") R16In
     elseif(abs(R16In).gt. 1.e20_16) then; write(a40out,"(f40.12   )") R16In
     elseif(abs(R16In).gt. 1.e10_16) then; write(a40out,"(f40.21   )") R16In
     elseif(abs(R16In).gt.999.e0_16) then; write(a40out,"(f40.29   )") R16In
     elseif(abs(R16In).gt.1.e-10_16) then
                        write(a40out,"(f40.32       )")                R16In
     elseif(abs(R16In).gt.1.e-20_16) then
                        write(a40out,"(f36.31,'E-10')") Real16InL*10._16**10
     elseif(abs(R16In).gt.1.e-30_16) then
                        write(a40out,"(f36.31,'E-20')") Real16InL*10._16**20
     elseif(abs(R16In).gt.1.e-40_16) then
                        write(a40out,"(f36.31,'E-30')") Real16InL*10._16**30
     else                                ; write(a40out,"(e40.31   )") R16In
   endif!
   return
End Subroutine FloatWrite
!---------------------------------------------------------------------------7-9

Subroutine FDate23(DaTime)                             !Year Mo D  Hr Mn Sec~ms
 !A snapshot of the computers current date & time:      2018.09.20 06:12:20:390
 !-----
 implicit none                                                       !arguments
  character::DaTime*23
 !integer*4::i,iDMY(3),iHMS(3)
  integer*4::MyValues(8)
 !----------
  !call iDate(iDMY)
  !call iTime(iHMS)
   call DATE_AND_TIME(Values=MyValues)
  !write(DaTime,"(i4,'.',i2.2,'.',i2.2,'.',i2.2,i2.2'.',i2.2,' L')")  &
  !              (iDMY(i),i=3,1,-1)       ,(iHMS(i),i=1,3),0
  !write(DaTime,"(i4,'.',i2.2,'.',i2.2,'.',i2.2,i2.2,':',i2.2,':',i3.3)")  &
   write(DaTime,"(i4,'.',i2.2,'.',i2.2,' ',i2.2,':',i2.2,':',i2.2,':',i3.3)")&
                  MyValues(1:3)           ,MyValues(5:8)                ;return
End Subroutine Fdate23
!---------------------------------------------------------------------------7 9

Subroutine Beamer(n,nTot)
!2018.08.27.1155cdt JMS- DOS screen iteration-progress bar (2% increments)
!                            -&- cumulative time predictor (seconds) .
!                      - Traveler2/Athlon64/WinXPPro-32/APF9.0-32
 !---                                                                No globals
 implicit none                                                       !arguments
  integer*4::n
  integer*4::nTot
  !---                                                               !internals
  integer*4::Init=0
  real*4   ::dT,tarray(2),dTime
  integer*4::nm
```

```
   !-----                                                            !end defs
   if(    n       ==    1) then; write(6,"('Beamer '\)"); dT=dtime(tarray)
                                 nm=nTot/50; if(nm==0) nm =1      ;return;endif
   if(    n       == nTot) then; write(6,"(' done.' )")                  ;endif
   if(mod(n,nm)   /=    1)                                         return
   if(    n/nm    ==    1) then;                          dT=dtime(tarray)
                                 write(6,"(f9.2,'sec.'\)") dT*50.d0      ;endif
   if(mod(n,nm*5) ==    1) then; write(6,"('*'\)")                ;return;endif
                                 write(6,"(':'\)")                ;return
End Subroutine Beamer
!---------------------------------------------------------------------------7-9

```