Appendix III

Chain-growth model parameter estimation

This Appendix describes the chain growth model fitting program used for parameter estimation from the model presented in Chapter 4. The program is written in FORTRAN and has been run on the Main Frame IBM 3081.

3.1 Program Minimum

This is a program that can provide parameter estimates for a function of several variables, by minimizing the function using a quasi-Newton method. The user must provide a subroutine to calculate the function and its gradient (first partial derivatives with respect to the variables of interest.) This function is the objective function, which consists of the summation of the square of the difference of the experimental data points and the function evaluated at these points using the current estimates of the solution vector. The user initially inputs the number of variables (2 in our case) and an estimate or initial guess of the solution vector for these variables, and the original experimental data points. The program iteratively searches for the parameter values that minimize the provided objective function. Finally, the program output gives the final solution vector, and the function values that correspond to the experimental data points, as well as a termination indicator that indicates whether convergence occurred.

'MINIMUM' is the name of the FORTRAN main program used here; it is a general error minimization routine that can be used for any user-supplied function. The Subroutine 'SUMDATA' provides the chain growth model objective function and its partial derivatives with respect to the two parameters, τ and τ_m. The Subroutine 'FUNC' that is called by subroutine 'SUMDATA' actually evaluates the $F_n(t, n, \tau, \tau_m)$ that is obtained as a solution for the chain growth model presented in Chapter 4 at input values of these four parameters. If a different solution function needs to be used, it can be modified in this subroutine. The user also inputs as data the F (rise) curves obtained from the transient response isotopic tracer experiments described in Chapter 4. The input requires the carbon number of the F curve, initial guesses of τ and τ_m and finally, t_i, $F^{expt}(t_i)$ pairs. A listing of the program follows.

PROGRAM MINIMUM

```
C-----------------------------------------------------------------------

      Program Minimum

      implicit none

      INTEGER   INPUTUNIT,MAXDATA,N
      integer   nvar,ihess,maxf,irhess,iwhess,iprint
      INTEGER   IERR,ISWTCH,ITER,NFCALL,I
      parameter (nvar=2,ihess=.5*nvar*(nvar+1))
      real*8    tolx,tolg,sprec,extbnd,rfn,objf,histry,
     2          x(nvar),g(nvar),solvec(nvar),srhvec(nvar),grdvec(nvar)
      REAL*8    HESS(IHESS),SCRVEC(NVAR),NORM,T(100),FEXP(100)
      real*8    fn,deltau,deltaub
      LOGICAL   ALWAYSTRUE
c external function calls for psi and phi

      external sumdata

c initialize input parameters for the minimization routine

      data maxf,tolx,tolg,sprec,extbnd,irhess,iwhess,iprint,ierr,
     2     iswtch,rfn/500,1d-6,1d-10,.1,2.,0,0,0,0,2,1d-6/

      data AlwaysTrue/.true./
      data inputunit/20/
      DATA OBJF/0.0D0/

      COMMON /EXPTDATAR/T,FEXP
      common /exptdataI/MaxData
      COMMON /CARBON/N
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

c     OPEN (UNIT=INPUTUNIT,FILE='INPUT.DAT',STATUS='UNKNOWN')
c     OPEN(UNIT=INPUTUNIT)
c initial guess on parameters

c     READ(INPUTUNIT,*) X
c     WRITE(6,*)X
      DO 1 I=1,NVAR
      READ(20,*)X(I)
 1    CONTINUE
c data points used in fitting function

c     I=0
c     DO WHILE (ALWAYSTRUE)

c        I=I+1
c        READ(INPUTUNIT,*,END=999,ERR=999)T(I),FEXP(I)
c
c     ENDDO
c
C99   CLOSE (UNIT=INPUTUNIT)
```

```
C       MAXDATA = I-1
C       MAXDATA IS THE # OF DATA POINTS;N IS THE CARBON # OF DATA
        READ(20,*)N,MAXDATA
        DO 2 I=1,MAXDATA
        READ(20,*)T(I),FEXP(I)
C       WRITE(6,*)'T=',T(I),'FEXP',FEXP(I)
 2      CONTINUE
        do i=1,ihess
           hess(i)=0.0d0
        enddo

        call gqbfgs(nvar,x,iswtch,maxf,iter,iprint,tolx,tolg,rfn,
     2       sprec,extbnd,objf,ihess,hess,g,ierr,nfcall,srhvec,
     3       solvec,grdvec,scrvec,histry,irhess,iwhess,sumdata)

        WRITE(22,*)'OBJF: ',OBJF

        NORM = 0.0D0
        do i = 1,nvar
           norm = norm + g(i)**2
        enddo
        NORM =DSQRT(NORM)

        WRITE(22,*)'FINAL X: ',SOLVEC
        WRITE(22,*)'GRADIENT: '
        WRITE(22,*)(G(I),I=1,NVAR)
        WRITE(22,*)'NORM: ',NORM
        WRITE(22,*)'CARBON#',N
        WRITE(22,*)'ITER: ',ITER
        WRITE(22,*)'IERR: ',IERR
        WRITE(22,*)'NFCALL: ',NFCALL
        WRITE(22,*)

        WRITE(23,*) '"MODEL',N,'".'
        do i = 1,MaxData
           call FUNC(T(i),FN,DELTAU,DELTAUB,X,MaxData)
           WRITE(23,*) T(I),FN
        enddo
        WRITE(23,*)
        WRITE(23,*) '"EXP',N,'".'
        DO I = 1,MAXDATA
           WRITE(23,*) T(I),FEXP(I)
        ENDDO

        stop
        end


cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c note: all real variables in this routine are double precision
c so make sure that any variable transfered into the routine is
c double precision (64 bit)

      subroutine gqbfgs(nvar,x,iswtch,maxf,iter,iprint,tolx,tolg,
     1           rfn,sprec,extbnd,objf,ihess,hess,g,ierr,nfcall,srhvec,
     2           solvec,grdvec,scrvec,histry,irhess,iwhess,funct)
```

```
c
      implicit real*8 (a-h,o-z)
c
      dimension  x(nvar),hess(ihess),g(nvar),srhvec(nvar),
     1           solvec(nvar),grdvec(nvar),scrvec(nvar)
c
      external funct
c
c     this subroutine minimizes a function of several variables
c     objf(x(1),x(2),....,x(nvar))  using a quasi-newton method
c     optionally employing the  dfp  and  bfgs  updating formulas.
c     the user must provide a subroutine to calculate objf(x) and
c     its gradient (first partial derivative) vector g(x).
c
c     **********************************************************
c     this routine invokes the package modules  search  and  uphess
c     and the user supplied subroutine  funct.
c     **********************************************************
c
c     on input:
c
c        nvar  is the number of variables. it is also the dimension of
c           the vectors  x, g, srhvec, solvec, grdvec  and  scrvec.
c
c        x  contains an estimate of the solution vector
c           (x(1),x(2),...,x(nvar)).
c
c        iswtch  is a parameter set equal to  k  which selects the
c           formula used to update the approximation to the hessian
c           inverse. for
c
c           k = 1 - the  dfp  update,
c           k = 2 - the  bfgs  update.
c
c           the  bfgs  update is recommended.
c
c        maxf  is the limit on the number of calls to the function
c           evaluation routine  funct.
c
c        funct  is a user supplied subroutine to evaluate  objf(x)
c           and the components of the gradient  g(x)  at the estimate
c           x(i), i = 1,2,...,nvar. Decternal in calling
c           routine.
c
c        tolx,tolg are the accuracies required in the solution, i.e. a
c           normal return from the routine occurs if the difference
c           between the components of two successive estimates of the
c           solution are not greater than  max(tolx*abs(x(i)),tolx)
c           for all  i, and the l2 norm of the gradient is not greater
c           than  tolg.
c
c        rfn  is an estimate of the expected reduction in  objf(x).
c           this estimate is used only on the first iteration so an
c           order of magnitude estimate will suffice. the information
c           can be provided in the following ways depending upon the
c           value of  rfn. for
```

```
c
c      rfn .gt. 0.0 - the setting of  rfn  itself will be taken
c                     as the expected reduction in  objf(x),
c
c      rfn = 0.0    - it is assumed that an estimate of the
c                     minimum value of  objf(x)  has been set
c                     in the argument  objf, and the expected
c                     reduction in  objf(x)  will be computed
c                     as (initial function value) minus objf,
c
c      rfn .lt. 0.0 - a multiple  abs(rfn)  of the modulus of
c                     the initial function value will be taken
c                     as the expected reduction.
c
c    sprec  is the accuracy required in the linear search technique
c       invoked by  gqbfgs, i.e. a point  xm  is accepted as the
c       minimum along the search direction if the ratio of the
c       directional derivative at  xm  over the directional
c       derivative at the initial point is not greater than  sprec.
c
c       the setting  0.100  is recommended.
c
c    extbnd  is the upper bound on the multiplicative increase in
c       the search scang the extrapolation phase of the
c       linear search technique.
c
c       the setting  2.000  is recommended.
c
c    objf  contains an estimate of the minimum value of  objf(x)
c       if  rfn = 0.0. otherwise it is only an output parameter.
c
c    ihess  is a parameter set equal to the dimension of  hess
c       which is at least  nvar*(nvar+1)/2.
c
c    histry  is a dummy parameter.
c
c  on output
c
c
c    x  contains the best available estimate of the solution vector.
c
c    objf  contains the function value at  x.
c
c
c    hess  is an array of dimension  ihess  which contains the
c       upper triangle of the most recent approximation to the
c       hessian inverse stored row-wise.
c
c    g  contains the components of the gradient at  x.
c
c    ierr  is a parameter set equal to  k  which gives the following
c       termination indications
c
c       normal termination,
c         k = 0,
c       intermediate termination.
```

```
c             k = -n(n any integer) - user termination,
c             k = 1 - failure to converge in  maxf  calls of  funct,
c             k = 2 - linear search technique indicates that it is
c                     likely that no minimum exists,
c
c        nfcall  is the number of calls to  funct.
c
c        srhvec  contains the current search direction vector.
c
c        solvec  contains the current solution vector.
c
c        grdvec  contains the current gradient vector.
c
c        scrvec  is a scratch vector.
c
c     written by k. e. hillstrom, march, 1976.
c
c
c     initialize the following parameters
c
c       ierr   - the termination indicator
c       nfcall - the number of calls to  funct
c       redfcn - the initial predicted reduction in  objf
c       iter   - the current iteration number
c
      ierr = 0
      iter=0
      nfcall = 1
      temp = objf
c
c     ************************************************************
      call funct(nvar,x,objf,g)
c     ************************************************************
c
      sqgrad=0.0
      do 220 iii=1,nvar
  220     sqgrad=sqgrad+g(iii)**2
c
      if (ierr .lt. 0) go to 410
      redfcn = rfn
      if (rfn .eq. 0.0) redfcn = objf - temp
      if (rfn .lt. 0.0) redfcn = abs(redfcn * objf)
      if (redfcn .le. 0.0) redfcn = 1.0
c
c    read initial estimate of hessian inverse, if desired
      if(irhess.eq.0) go to 200
      read(11,202) (hess(ii),ii=1,ihess)
  202   format(5e16.9)
      go to 300
c
c     begin the quasi-newton process by initializing the approximation
c       to the hessian inverse to unity
c
  200 if(iprint.ne.0) write(10,201)
  201 format(//2x,'>>>> quasi-newton procedure started, with search',
     1       ' direction set to -g')
```

```
      k = 1 + nvar * (nvar+1) / 2
c
      do 210 i = 1, nvar
c
         do 205 j = 1, i
            k = k - 1
            hess(k) = 0.0
  205    continue
c
         hess(k) = 1.0
  210 continue
c
  300 if(iprint.eq.0) go to 301
      euclid=sqrt(sqgrad)
      if(iprint.ge.20.and.mod(iter,10).ne.0) go to 308
      write(6,307) iter,nfcall,objf,euclid
  307 format(1x,'iteration:',i5,26x,'function evaluation:',i6,
     1      /1x,'objective function:',e20.13,2x,'gradient norme:',
     2       e20.13)
  308 if(mod(iter,iprint).ne.0) go to 301
      write(10,302) iter,nfcall
  302 format(////2x,'iteration no ',i5//2x,'number of function and ',
     1       'gradient evaluations = ',i5//2x,'parameter values')
      write(10,303) (j,x(j),j=1,nvar)
  303 format(/3(2x,'x(',i4,') = ',e16.8))
      write(10,304) objf
  304 format(//2x,'function value objf = ',e16.8///2x,'gradient')
      write(10,306) (j,g(j),j=1,nvar)
  306 format(/3(2x,'g(',i4,') = ',e16.8))
  301 iter=iter+1
c
c     begin an iteration by saving the current best estimate of the
c       function and the solution and gradient vectors.
c
      do 310 i = 1, nvar
         solvec(i) = x(i)
         grdvec(i) = g(i)
  310 continue
c
      tobjf = objf
c
c     calculate the search direction vector in  srhvec  and the
c       directional derivative in  dirdev
c
      do 340 i = 1, nvar
         ij = i
         z = 0.0
c
         do 330 j = 1, nvar
            z = z - g(j) * hess(ij)
            if (j .ge. i) go to 325
            ij = ij + nvar - j
            go to 330
  325       ij = ij + 1
  330    continue
c
```

```
         srhvec(i) = z
  340 continue
c
      dirdev = 0.0
c
      do 350 i = 1, nvar
         dirdev = dirdev + srhvec(i) * g(i)
  350 continue
c
c     if the directional derivative  dirdev  is .gt. 0, there is no
c       guarantee that a search in the  w  direction  will result in a
c       smaller  objf. therefore, the  quasi-newton  process is
c       restarted at the current estimate of the solution with  srhvec
c       set to  -g.
c
      if (dirdev .gt. 0.0) go to 200
      if (dirdev .eq. 0.0) go to 500
c
c     compute the initial search scaha  and conduct the
c       linear search by means of a call to  search
c
      alpha = -2.0 * redfcn / dirdev
      if (alpha .gt. 1.0) alpha = 1.0
      redfcn = objf
c
c     **********************************************************
      call search(nvar,x,g,srhvec,objf,alpha,dirdev,sprec,
     1            extbnd,nfcall,scrvec,ierr,funct,maxf)
c     **********************************************************
c
c     test for abnormal termination
c
      if (ierr .lt. 0) go to 500
      if (nfcall .ge. maxf) go to 400
      if ((alpha .lt. 1.0e-20) .or.
     1    (alpha .gt. 1.0e20)) go to 410
c
c     test for convergence
c
      sqgrad = 0.0
      iconv = 0
c
      do 360 i = 1, nvar
         temp = alpha * srhvec(i)
         sqgrad = sqgrad + g(i) * g(i)
         t = tolx * abs(x(i))
         if (t .le. tolx) t = tolx
         if (abs(temp) .gt. t) iconv = 1
  360 continue
c
      if (sqgrad .gt. tolg*tolg) iconv = 1
      if (sqgrad .eq. 0.0) iconv = 0
      if (iconv .eq. 0) go to 500
c
c     the linear search technique has located a minimum. call  uphess
c       to update the approximation to the hessian inverse using the
```

```
c        dfp  or  bfgs  updating formulas
c
c      ***************************************************************
       call uphess(nvar,x,g,ihess,hess,solvec,grdvec,scrvec,iswtch,iexit)
c      ***************************************************************
c
c      if the update is not successful the quasi-newton process is
c        restarted with a descent step at the current estimate of the
c        solution
c
       redfcn = redfcn - objf
       if (iexit .ne. 0) go to 200
c
c      now start a new iteration
c
c  write the current estimate of the hessian inverse, if desired.
       if(iwhess.eq.0) go to 300
       write(12,202) (hess(ii),ii=1,ihess)
c
       go to 300
c
c      error return because there have been at least  maxf  calls of
c        funct
c
  400  ierr = 1
       go to 450
c
c      error return because linear search technique indicates that it is
c        likely that no minimum exists
c
  410  ierr = 2
c
  450  do 455 i = 1, nvar
          x(i) = solvec(i)
          g(i) = grdvec(i)
  455  continue
c
       objf = tobjf
  500  return
c
       end

cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
       subroutine uphess(n,x,g,ih,h,solvec,grdvec,scrvec,iswtch,iexit)
c
       implicit real*8 (a-h,o-z)
c
       dimension  x(n),g(n),h(ih),solvec(n),
      1           grdvec(n),scrvec(n)
c
c      this subroutine updates an approximation to the hessian inverse
c        using the  dfp  or  bfgs  formula
c
c      on input
c
c         n  is the dimension of the vectors  x, g, solvec, grdvec  and
```

```
c           scrvec.
c
c        x  contains an estimate of the solution vector.
c
c        g  contains the components of the gradient corresponding to
c           the  x  vector.
c
c        ih  is a parameter set equal to the dimension of  h  which is
c           at least  n*(n+1)/2.
c
c        h  is an array of dimension  ih  which contains the upper
c           triangle of an approximation to the hessian inverse stored
c           by rows.
c
c        solvec  contains the current solution vector.
c
c        grdvec  contains the current gradient vector.
c
c        iswtch  is a parameter set equal to  k  which selects the
c           updating formula. for
c
c           k = 1 - the  dfp  formula is used,
c           k = 2 - the  bfgs  formula is used.
c
c     on output
c
c        iexit  is a parameter set equal to  k  which indicates the
c           following. for
c
c           k = 0 - the update was successful,
c           k = 1 - the update failed due to zero divisors.
c
c        h  contains the updated approximation to the hessian inverse
c           if  iexit = 0.
c
c        scrvec  is a scratch vector.
c
c     written by k. e. hillstrom, march, 1976.
c
c
c     initialize the exit indicator  iexit
c
      iexit = 0
c
c     calculate the solution and gradient difference vectors from two
c       consecutive iterations. from this section on
c
c       solvec - contains  delta, the solution difference vector
c       grdvec - contains  gamma, the gradient difference vector
c
  100 do 110 i = 1, n
         solvec(i) = x(i) - solvec(i)
         grdvec(i) = g(i) - grdvec(i)
  110 continue
c
c     calculate  z = (gamma transpose) * delta and alpha =
```

```
c       (gamma transpose) * (hessian inverse) * gamma  occuring as
c       denominators in the  dfp  formula. from this section on
c
c       h      - contains the approximation to the hessian inverse
c       scrvec - contains the successive elements of  (gamma transpose)
c                * (hessian inverse)
c
      z = 0.0
      alpha = 0.0
c
      do 130 i = 1, n
         wt = grdvec(i)
         z = z + wt * solvec(i)
         k = i
         wt = 0.0
c
         do 120 j = 1, n
            wt = wt + grdvec(j) * h(k)
            if (j .ge. i) go to 115
            k = k + n - j
            go to 120
  115       k = k + 1
  120    continue
c
         alpha = alpha + wt * grdvec(i)
         scrvec(i) = wt
  130 continue
c
c     error exit if the  dfp  or  bfgs  formula breaks down due to zero
c       divisors  z  and/or  alpha
c
      if ((z .eq. 0.0) .or.
     1    (alpha .eq. 0.0 .and. iswtch .eq. 1)) go to 200
c
c     update the approximation to the hessian inverse using the  dfp
c       or  bfgs  updating formula
c
      k = 1
c
      do 160 i = 1, n
c
         do 150 j = i, n
            if (iswtch .eq. 1) go to 135
            h(k) = h(k) - (solvec(i) * scrvec(j) + scrvec(i) *
     1             solvec(j)) / z + (1.0 + alpha / z) * (solvec(i) *
     2             solvec(j) / z)
            go to 140
  135       h(k) = h(k) + solvec(i) * solvec(j) / z - scrvec(i) *
     1             scrvec(j) / alpha
  140       k = k + 1
  150    continue
c
  160 continue
c
      go to 300
c
```

```
c     error return due to zero divisors in the updating formula
c
  200 iexit = 1
  300 return
c
      end
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine search(n,x,g,s,f,alpha,dirdev,sprec,extbnd,
     1                  nfcall,w,ierr,funct,maxf)
c
      implicit real*8 (a-h,o-z)
c
      dimension x(n),g(n),s(n),w(n)
c
c       this subroutine obtains an estimate of the solution
c     xm = x0 + alpha * s  which minimizes  f  by means of a linear
c     search in the  s  direction.
c
c     on input
c
c        n  is the number of variables. it is also the dimension of the
c           vectors  x, g  and  s.
c
c        x  contains an estimate of the solution vector
c           (x0(1),x0(2),...,x0(n)).
c
c        s  contains the search direction vector.
c
c        f  contains the objective function  f(x).
c
c        funct  is a user supplied routine to evaluate  f(x)  and the
c           components of the gradient  g(x)  at the estimate in  x.
c
c        alpha  is  the initial step sca
c
c        dirdev  is the directional derivative at x.
c
c        sprec  is the accuracy required in the search, i.e. a point  xm
c           is accepted as the minimum along direction  s  if the ratio
c           of the directional derivative at  xm  over the directional
c           derivative at  x0  is not greater than  sprec.
c
c        extbnd  is the upper bound on the multiplicative increase in
c           alpha  during extrapolation.
c
c        nfcall  is the number of calls to the function evaluation
c           subroutine  fcn.
c
c        w  is a scratch vector.
c
c     on output
c
c        x  contains the estimate of the minimum
c           (xm(1),xm(2),...,xm(n))
c
```

```
c     ierr  is a parameter set to a negative integer if the user
c           wishes to force an exit from  search. otherwise it is
c           unaltered.
c
c        g  contains the components of the gradient at  x.
c
c        f  contains the function value  f(x).
c
c        alpha  is the final step sca
c
c        dirdev  is the directional derivative at  x.
c
c        nfcall  is the number of calls to the function evaluation
c           subroutine  funct.
c
c
c     initialize the following parameters and indicators
c
c       tot        -  the sum of the extrapolation steps
c       cdirev     -  the current directional derivative
c       pdirev     -  the previous directional derivative
c       ierr       -  the error indicator
c
      tot = 0.0d0
      cdirev = dirdev
      pdirev = dirdev
c
c     test whether  alpha  is too small
c
  105 if (alpha .le. 1.0d-20) go to 150
c
c     begin the linear search by incrementing the solution vector  x
c       and calculating the function and gradient at the incremented  x.
c
      do 108 i = 1, n
         w(i) = x(i)
         x(i) = x(i) + alpha * s(i)
  108 continue
c
c     ***********************************************************
      call funct(n,x,ftest,g)
c     ***********************************************************
c
      nfcall = nfcall + 1
      if (maxf.lt.nfcall) go to 160
      if (ierr .lt. 0) go to 150
c
c     compute the directional derivative  dirdev  at  x + alpha * s
c
      dirdev = 0.0d0
c
      do 110 i = 1, n
         dirdev = dirdev + g(i) * s(i)
  110 continue
c
c     test whether  f(x + alpha * s)  is less than  f(x).
```

```
c
      if (ftest .ge. f) go to 120
c
c     if (dirdev / pdirev) is less than the search precision  sprec,
c       alpha  is accepted. otherwise  alpha  is modified
c
      if (abs(dirdev / pdirev) .le. sprec) go to 140
c
c     alpha  is modified, test whether  alpha  is to be revised by
c       extrapolation or interpolation
c
      if (dirdev .gt. 0.0d0) go to 120
c
c     alpha  is revised using an extrapolation formula and a new step
c       is taken if the sum of the steps already made is not too
c       the input parameter  extbnd  limits the multiplicative change
c       in  alpha
c
      tot = tot + alpha
      if (tot .gt. 1.0d10) go to 145
      temp = extbnd
      if (cdirev .lt. dirdev) temp = dirdev / (cdirev - dirdev)
      if (temp .gt. extbnd) temp = extbnd
      f = ftest
      cdirev = dirdev
      alpha = alpha * temp
      go to 105
c
c     x  is reset to the current estimate, alpha  is revised using the
c       cubic interpolation formula and a new step is taken if the
c       convergence criteria have not been satisfied.
c
  120 do 130 i = 1, n
         x(i) = w(i)
  130 continue
c
      temp = 3.0d0 * (f - ftest) / alpha + dirdev + cdirev
      wt = abs(temp)
      if(wt.lt.abs(dirdev)) wt=abs(dirdev)
      if(wt.lt.abs(cdirev)) wt=abs(cdirev)
      ww = temp / wt
      ww = ww * ww - cdirev / wt * dirdev / wt
      if (ww .lt. 0.0d0) ww = 0.0d0
      ww = dsqrt(ww) * wt
      temp = 1.0d0 - (dirdev + ww - temp) / (2.0d0 * ww + dirdev -
     1       cdirev)
      alpha = alpha * temp
      go to 105
c
c     alpha is accepted
c
  140 f = ftest
  145 alpha = tot + alpha
  150 return
c
  160 do 170 i = 1, n
```

```
         x(i) = w(i)
  170 continue
      return
c
      end

      SUBROUTINE SUMDATA(NVAR,X,OBJF,G)
C     FILE TO GENERATE FN(T) FOR FISCHER-TROPSCH MODELLING AND FIT DATA
C     N IS CARBON NUMBER,NO IS THE NO OF DATA POINTS
C     TAU AND TAUB ARE GUESSES

      IMPLICIT REAL*8(A-H,O-Z)
      COMMON /EXPTDATAR/T,FEXP
      COMMON /EXPTDATAI/NO
      COMMON /CARBON/N
      DIMENSION T(100),FEXP(100),X(NVAR),G(NVAR),Y(2)
C*****************************************************************
C     OBJF IS SUM OF(F(EXPT)-F(T))**2;DOFTAU IS D/DTAU OF OBJF;
C     DOFTAUB IS D/DTAUB OF OBJF
C*****************************************************************

      OBJF=0.0D0
      DOFTAU=0.0D0
      DOFTAUB=0.0D0
      DTAU=0.0D0
      DTAUB=0.0D0
      DELT=1.0D-5

      DO 2 I=1,NO
         Y(1)=X(1)
         Y(2)=X(2)
         TIME=T(I)
         CALL FUNC(TIME,FN,DELTAU,DELTAUB,X,NVAR)
         OBJF=OBJF+(FEXP(I)-FN)**2
         DOFTAU=DOFTAU+(-2.0D0*(FEXP(I)-FN)*DELTAU)
         DOFTAUB=DOFTAUB+(-2.00*(FEXP(I)-FN)*DELTAUB)
         Y(1)=X(1)*(1.0 + DELT/2. )
         CALL FUNC(TIME,FNT,DELTA,DELTAB,Y,NVAR)
         Y(1)=X(1)*(1.0 - DELT/2. )
         CALL FUNC(TIME,FNT1,DELTA,DELTAB,Y,NVAR)
         DTAU=DTAU+(-2.0D0*(FEXP(I)-FN)*(FNT-FNT1)/(X(1)*DELT))

         Y(1)=X(1)
         Y(2)=X(2)*(1.0 + DELT/2.0D0)
         CALL FUNC(TIME,FNTB,DELTA,DELTAB,Y,NVAR)
         Y(2)=X(2)*(1.0 - DELT/2.0D0)
         CALL FUNC(TIME,FNTB1,DELTA,DELTAB,Y,NVAR)
         DTAUB=DTAUB+(-2.0D0*(FEXP(I)-FN)*(FNTB-FNTB1)/(X(2)*DELT))

  2   CONTINUE

      WRITE(6,*)'O.F',OBJF,DTAU,DTAUB
      WRITE(6,*)'TAU=',X(1),'TAUB',X(2)
c     G(1)=DOFTAU
c     G(2)=DOFTAUB
      G(1)=DTAU
```

```
      G(2)=DTAUB

      RETURN
      END

      SUBROUTINE FACT(M,J)
      J=1
      IF(M.EQ.0)GOTO 4
      DO 1 I=1,M
      J=J*I
 1    CONTINUE
4     RETURN
      END
C***********************************************************************
C     FUNC EVALUATES FN(TIME),AND PARTIAL DERIVATIVES WRT TAU AND TAUB
C         DELTAU AND  DELTAUB
C**********************************************************************
      SUBROUTINE FUNC(TIME,FN,DELTAU,DELTAUB,X,NVAR)
      IMPLICIT REAL*8(A-H,O-Z)
      COMMON /CARBON/N
      DIMENSION X(NVAR)
      TAU=X(1)
      TAUB=X(2)
      A1=TAUB/(TAU-TAUB)
      AK1=0.0D0
      AK2=0.0D0
      DO 1 I=1,N
      AK1=AK1+(A1**I)*((-1)**(I+1))
 1    CONTINUE
      AK2=AK2+((-A1)-1.)
      DO 2 I=2,N
C     WRITE(22,*)TIME,NI
      DO 3 IR1=1,I-1
      IR=IR1-1
      CALL FACT(I-IR-1,IRR)
C     WRITE(22,*)TIME,IRR
      IF(IRR.LE.0)GOTO 200
      AK2=AK2+(1.0D0/(TAU**I))*(TAU**(IR+1)*TIME**(I-IR-1)*(-1+(-A1)**(I
     1R+1))/IRR)
 3    CONTINUE
      AK2=AK2+(1.0D0)*((-A1)**(I)-1.)
 2    CONTINUE
      T1=TIME/TAU
      T2=TIME/TAUB
      IF(T1.LT.75.D0.AND.T2.LT.75.D0)GO TO 20
      FN=1.0D0
      IF(T1.LT.75.D0)FN=1.0D0+AK2*DEXP(-TIME/TAU)/N
      IF(T2.LT.75.D0)FN=1.0D0+AK1/N*DEXP(-TIME/TAUB)
      GO TO 21
 20   FN=1.0+(AK1/N)*DEXP(-TIME/TAUB)+AK2*DEXP(-TIME/TAU)/N
 21   AL1=0.0D0
      AL2=AK2
      AL3=0.0D0
      AL4=0.0D0
      DO 4 I=1,N
```

```
      AL1=AL1+(-TAUB)**I*I/((TAU-TAUB)**(I+1))                          MIN08380
    4 CONTINUE                                                          MIN08390
      AL3=AL3+1.0*((-A1)-1.0D0)/TAU                                     MIN08400
      AL4=AL4+1.0D0/TAU*((-A1-1.0D0)-TAU*(-A1)/(TAU-TAUB))              MIN08410
      DO 5 I=2,N                                                        MIN08420
      DO 6 IR1=1,I-1                                                    MIN08430
      IR=IR1-1                                                          MIN08440
      CALL FACT(I-IR-1,IRR)                                             MIN08450
      IF(IRR.LE.0)GO TO 200                                             MIN08460
      AL3=AL3+1.00/TAU**(I+1)*I*(TAU)**(IR+1)*TIME**(I-IR-1)/IRR*(-1+(-AMIN08470
     1I)**(IR+1))                                                       MIN08480
      AL4=AL4+1.0/TAU**I*TIME**(I-IR-1)/IRR*((IR+1)*TAU**IR*(-1+(-A1)**(MIN08490
     1IR+1))-(TAU**(IR+1)*((IR+1)*(-A1)**(IR+1)/(TAU-TAUB))))           MIN08500
    6 CONTINUE                                                          MIN08510
      AL3=AL3+I/TAU*((-A1)**I-1.0D0)                                    MIN08520
      AL4=AL4+I/TAU*((-A1)**I-1.0D0-TAU*(-A1)**I/(TAU-TAUB))            MIN08530
    5 CONTINUE                                                          MIN08540
      IF(T1.LT.75.D0.AND.T2.LT.75.D0)GO TO 22                           MIN08550
      DELTAU=0.0D0                                                      MIN08560
      IF(T1.LT.75.D0)DELTAU=DEXP(-T1)/N*(TIME/TAU**2*AL2-AL3+AL4)       MIN08570
      IF(T2.LT.75.D0)DELTAU=AL1/N*DEXP(-TIME/TAUB)                      MIN08580
      GO TO 23                                                          MIN08590
   22 DELTAU=AL1/N*DEXP(-TIME/TAUB)+DEXP(-TIME/TAU)/N*(TIME/TAU**2*AL2-AMIN08600
     1L3+AL4)                                                           MIN08610
C     WRITE(6,*)'TIME',TIME,'DELTAU',DELTAU,X(1),X(2)                   MIN08620
   23 AM2=0.0                                                           MIN08630
      AM3=0.0                                                           MIN08640
      DO 7 I=1,N                                                        MIN08650
    7 AM2=AM2+(-1)**(I+1)*I*TAU*TAUB**(I-1)/(TAU-TAUB)**(I+1)           MIN08660
      AM3=AM3-TAU/(TAU-TAUB)**2                                         MIN08670
      DO 8 I=2,N                                                        MIN08680
      DO 9 IR1=1,I-1                                                    MIN08690
      IR=IR1-1                                                          MIN08700
      AM3=AM3-1./TAU**I*(TAU**(IR+2)*TIME**(I-IR-1)*(IR+1)*(-TAUB)**IR/IMIN08710
     1RR/(TAU-TAUB)**(IR+2))                                            MIN08720
    9 CONTINUE                                                          MIN08730
      AM3=AM3-TAU*I*(-A1)**(I-1)/(TAU-TAUB)**2                          MIN08740
    8 CONTINUE                                                          MIN08750
      IF(T1.LT.75.D0.AND.T2.LT.75.D0)GO TO 24                           MIN08760
      DELTAUB=0.0D0                                                     MIN08770
      IF(T2.LT.75.D0)DELTAUB=EXP(-T2)/N*(TIME/TAUB**2*AK1+AM2)          MIN08780
      IF(T1.LT.75.D0)DELTAUB=AM3/N*DEXP(-T1)                            MIN08790
      RETURN                                                            MIN08800
   24 DELTAUB=EXP(-TIME/TAUB)/N*(TIME/TAUB**2*AK1+AM2)+EXP(-TIME/TAU)/N*MIN08810
     1AM3                                                               MIN08820
      RETURN                                                            MIN08830
  200 WRITE(6,201)                                                      MIN08840
  201 FORMAT(1X, 'FACTORIAL RETURNED NEGATIVE')                         MIN08850
      END                                                               MIN08860
                                                                        MIN08870
                                                                        MIN08880
```