```
!DATA ASSIMILATION SAMPLE PROGRAM (R. Stull, UBC, Nov 2000)
!Includes both Bratseth's Sucessive Correction
!-AND- Optimum Interpolation, for comparison.
!Reference:
!Bratseth 1986: "Statistical interpolation by means
!of successive corrections".  Tellus, 38A,439-447.


!===================== Modules (works like common blocks)

        module gridmodule               !grided analysis data (as meteorological fields)
        integer :: nx                   !max number of grid points in x direction
        real :: dx                      !delta_x = grid increment in x direction
        real :: xwest                   !location of left edge (west-most) x grid point
        character (len=5) :: gridunits  !example "km"
        real, dimension (:), allocatable :: xgrid   !grid point x-locations
        real, dimension (:), allocatable :: agrid   !analysis grid point scalar values (eg, temperature)
        real, dimension (:), allocatable :: bgrid   !background (first-guess) scalar values (eg, temperature)
        real, dimension (:), allocatable :: fgrid   !analysis increment (analysis - firstguess)
        end module gridmodule

        module obsmodule                !observation data (with same units as for x grid)
        integer :: nobs                 !max number of observations
        real, dimension (:), allocatable :: xobs    !observation x-locations
        real, dimension (:), allocatable :: sobs    !actual observation values (eg, temperature)
        real, dimension (:), allocatable :: aobs    !analysis values estimated at obs locations
        real, dimension (:), allocatable :: bobs    !background grid values interpolated to obs locations
        real, dimension (:), allocatable :: fobs    !observation increment (sobs - bobs)
        real, dimension (:), allocatable :: aobsnew !updated analysis values at obs locations
        real :: xo, xscale              !Offset and and scaling factor: Bratseth -> reality
        real :: so, sscale              !Offset and and scaling factor: Bratseth -> reality
        end module obsmodule

        module statmodule               !correlation and error statistics
        real :: sigmaBR                 !Bratseth dimensionless radius of influence
        real :: sigmaR,sigmaR2          !physical radius of influence, and its square
        real :: sigmax                  !x std deviation (assumed same for all grid points)
        real, dimension (:), allocatable :: sigmaobs    !observation std dev at each obs station
        end module statmodule

        module iterationmodule          !misc iteration variables
        integer :: nu                   !iteration counter
        real :: sumcor                  !sum of correction magnitudes
        real :: sumf                    !sum of analysis magnitudes
        real :: epsilon  = 0.001        !relative convergence criterion
        character (len=1) :: tab=achar(9)   !ascii tab character
        end module iterationmodule

        module optimumodule             !only for comparison with optimum interpolation
        real, dimension (:), allocatable :: ogrid       !optmum analysis at grid locations
        real, dimension (:,:), allocatable :: amatrix   !normalized covariance matrix (obs)
        real, dimension (:), allocatable :: bmatrix     !normalized RHS matrix (obs,grid)
        real, dimension (:), allocatable :: vv          !implicit scalings for each row
        integer, dimension (:), allocatable :: indx     !LU decomposition vector
        real :: TINY=1.0e-20                            !prevention of singular matrix
        end module optimumodule



!====================== main program ====================
        program scmain                  !successive correction main program
!declare variables
        use gridmodule                  !analysis grid info
        use obsmodule                   !observation station info
        use statmodule                  !statistics of station errors and correlations
        use iterationmodule             !misc iteration info
        implicit none                   !enforce strong typing
        logical :: done                 !true when converged
```

```
!set up
       call welcome                 !welcome the user
       call setgrid                 !set up the analysis grid
       call getobs                  !get the observation data points
       call firstguess              !get the first-guess (background) gridded analysis
       call getstats                !get observation-error statistics
!iterate
       call preprocess              !set up for iterations
       do nu=1,25                   !for each successive iteration, nu
           call refine              !make a better analysis estimate
           call savefields          !save results for future display
           if (done()) exit         !leave iteration loop if converged
       enddo
       call optimum                 !Optimum interpolation (for comparison)
       call postprocess             !do any postprocessing
       end program scmain



!===============================================
       subroutine welcome           !Welcome the interactive user
       implicit none                !enforce strong typing
       write(*,*)
       write(*,*) "Welcome to SUCCESSIVE CORRECTION ... a Data Assimilation Method"
       write(*,*)
       write(*,*) "Based on Bratseth, A.M. 1986: Statistical interpolation "
       write(*,*) "by means of successive corrections. Tellus, 38A,439-447."
       write(*,*)
       write(*,*) "Coded by R. Stull, UBC, Nov 2000."
       write(*,*)
       end subroutine welcome


!=======================
       subroutine setgrid           !Set grid locations x
!Calculates evenly-spaced grid locations using location offset and scale
!values that were read from an input file.
       use gridmodule               !common grid variables
       implicit none                !enforce strong typing
       integer :: ix                !dummy grid index
       character (len=50) :: header !text header from grid file
!Initialize
       nx = 100                     !max number of grid locations in x direction
       gridunits = "km"             !units in x direction
       xwest = 0.0                  !location of west (leftmost) grid point
       dx = 1.0                     !grid increment delta_x
!Get grid-location info from a disk file (as ascii text)
       write(*,*)
       write(*,*) "========== SETGRID - Set up the grid"
       write(*,*) "Hit RETURN to search for grid input file (in.GRID)"
       read(*,*)                    !wait until user responds
       open(1,file="",status="old") !activates Macintosh file finder box
       read(1,"(a50)") header ; print *, header    !this line useful in input file
       read(1,*) nx,gridunits,xwest,dx !numb of grid points, units, starting(leftmost) x, delta_x
       print *, nx, gridunits, xwest, dx   !echo to screen
       close(1)                     !de-activate the input file
       write(*,*) "SETGRID RESULTS: nx = ",nx, ", xwest & delta_x = ",xwest,dx," (",trim(gridunits),")"
!initialize grids
       allocate(xgrid(nx))          !dynamically allocate memory for x-location grid
       allocate(agrid(nx))          !dynamically allocate memory for analysis grid
       allocate(bgrid(nx))          !dynamically allocate memory for background grid
       allocate(fgrid(nx))          !dynamically allocate memory for difference grid
       do ix = 1,nx                 !for each grid location
           xgrid(ix)=xwest + (ix-1)*dx !compute the physical location
       enddo
       write(*,*) "...ending  SETGRID" !status report
       end subroutine setgrid
```

```
!======================
       subroutine getobs               !Get the observation info (locations & values)
!Observations are paired (x,s), where s is the meteorological value
!(eg, temperature) at location x. Input is the raw (dimensionless) Bratseth values,
!which we then scale into reasonable meteorological and distance values.
!Read raw observation info from file.  The file format, by line, is:
!   number of observations.  Eg: 2
!   scaling in x direction, where input is X0 and XSCALE.  Eg: 40.0,10.0
!   scaling of s scalar, where input is S0 and SSCALE.  Eg: 0.0,10.0
!   header text for scaled (not raw Bratseth) fields.  Eg: x(km),F(degC)
!   first raw Bratseth "observation" as X, S:  Eg:  0.0,1.0
!   more rows of raw Bratseth obs as X,S, until number of rows = number of observations
!
       use obsmodule                   !common observation variables
       implicit none                   !enforce strong typing
       integer :: iobs                 !dummy observation index
       character (len=50) :: header    !text header (to help check units)
       write(*,*)
       write(*,*)  "========== GETOBS - Get the observation info "
!Read the observation info from file
       write(*,*)
       write(*,*) "Hit RETURN to search for observation input file (in.OBS)"
       read(*,*)                       !wait until user responds
       open(1,file="",status="old")    !activate Macintosh file finder box
       read(1,"(a50)") header ; print *, header    !this line useful in input file
       read(1,*) nobs ; print *, nobs  !read and echo number of obs
       allocate(xobs(nobs))            !dynamically allocate memory for obs x locatinos
       allocate(sobs(nobs))            !dynamically allocate memory for obs s values
       allocate(aobs(nobs))            !dynamically allocate memory for obs a values
       allocate(bobs(nobs))            !dynamically allocate memory for obs b values
       allocate(fobs(nobs))            !dynamically allocate memory for obs a-b values
       allocate(aobsnew(nobs))         !dynamically allocate memory for new a values
       read(1,*) xo,xscale     ; print *, xo,xscale    !scale x Bratseth -> reality
       read(1,*) so,sscale     ; print *, so,sscale    !scale s Bratseth -> reality
       read(1,"(a50)") header ; print *, "   ", header
       do iobs = 1,nobs                    !for each observation point
           read(1,*) xobs(iobs),sobs(iobs) !Bratseth's dimensionless "obs"
           print *, xobs(iobs),sobs(iobs)  !echo to screen
           xobs(iobs)=xo+xscale*xobs(iobs) !Scale into physically-realistic obs x locations
           sobs(iobs)=so+sscale*sobs(iobs) !Scale into physically realsitic scalar values
           print *, "  --> Scaled (x,s):", xobs(iobs),sobs(iobs)   !echo to screen
       enddo
!Interpolate the background first guess to the observation points
       close(1)                        !de-activate this input file
       write(*,*) "...ending  GETOBS"  !status report
       end subroutine getobs


!======================
       subroutine firstguess
!Set the first-guess (background) analysis, usually from the previous forecast (prog)
!This first-guess info is read from a file, with one grid-point value per line.
       use gridmodule                  !common grid variables
       use obsmodule                   !common observation variables
       implicit none                   !enforce strong typing
       write(*,*)
       write(*,*)  "========== FIRSTGUESS - Get the first guess gridded analysis "
       write(*,*)
       write(*,*) "Hit RETURN to search for first-guess input file (in.FIRST)"
       read(*,*)                       !wait until user responds
       open(1,file="",status="old")    !activate Macintosh file finder box
       read(1,*) bgrid                 !read whole first-guess (background) vector
       close(1)                        !de-activate the input file
       print *, "First guess is: ",bgrid(1) !echo first element of bgrid to screen
       agrid=bgrid                     !vector equality: set first analysis = background first-guess
       fgrid=0.0                       !initial state of difference vector (a-b)
       call interpolate                !interpolate background to obs locations
```

```
        bobs = aobs                      !copy interpolated results into background at obs points
        fobs = sobs-bobs                 !find observation increment
        write(*,*) "aobs =",aobs
        write(*,*) "bobs =",bobs
        write(*,*) "fobs =",fobs
        write(*,*) "...ending  FIRSTGUESS"  !status report
        end subroutine firstguess


!======================
        subroutine interpolate           !interpolate current gridded analysis to obs locations
!linearly interpolate the analysis from the grid points to all the obs locations
        use obsmodule                    !common observation variables
        use gridmodule                   !common grid variables
        implicit none                    !enforce strong typing
        integer :: iobs                  !dummy observation index
        integer :: ilow                  !grid index just left of obs location
        real :: position                 !relative obs position between neighboring grid points
!linear interpolation
        do iobs=1,nobs                                  !for each observation location
            ilow=floor((xobs(iobs)-xwest+1)/dx)         !calculate adjacent grid point location
            ilow=max(1,ilow) ; ilow=min(ilow,(nx-1))    !stay within grid
            position = ((xobs(iobs)-xgrid(ilow))/(xgrid(ilow+1)-xgrid(ilow)))   !rel.position
            aobs(iobs)=agrid(ilow)+(position*(agrid(ilow+1)-agrid(ilow)))       !interpolate
        enddo
        end subroutine interpolate


!=======================
        subroutine getstats              !get error statistics from file
!File format is:
!   epsilon         !relative convergence criterion (eg, 0.001)
!   sigmaBR         !Bratseth's dimensionless radius of influence (eg, 1.0)
!   sigmax          !first-guess error std deviation at grid points (same for all grid points, eg, 1.0)
!   sigmaobs        !array of obs-error std deviations, with each obs location on separate line (eg, 0.0)
        use gridmodule                   !common grid info
        use obsmodule                    !common observation info
        use statmodule                   !common error and correlation statistics
        use iterationmodule              !common iteration info
        implicit none                    !enforce strong typing
        character (len=50) :: header     !text header (to help check units)
        allocate(sigmaobs(nobs))         !dynamically allocate memory for obs std dev
        write(*,*)
        write(*,*)  "========== GETSTATS - Get error statistics "
        write(*,*)
        write(*,*) "Hit RETURN to search for statistics input file (in.STATS)"
        read(*,*)                        !wait until user responds
        open(1,file="",status="old")     !activate Macintosh file finder box
        read(1,"(a50)") header ; print *, header    !this line useful in input file
        read(1,*) epsilon                !read convergence criterion
        read(1,*) sigmaBR                !read Bratseth's dimensionless radius of influence
        read(1,*) sigmax                 !read std deviation at grid point x
        read(1,*) sigmaobs               !read whole array of observation std deviations
        close(1)                         !de-active the input file
        sigmaR = sscale*sigmaBR          !find physical radius of influence
        sigmaR2 = sigmaR*sigmaR          !square of radius of influence
        write(*,*) "convergence criterion = ",epsilon
        write(*,*) "Bratseth sigmaR =",sigmaBR
        write(*,*) "physical sigmaR = ",sigmaR," (",trim(gridunits),")"
        write(*,*) "gridded std deviation error: ",sigmax
        write(*,*) "array of observation std deviation errors: ",sigmaobs
        print *, "...ending GETSTATS"   !status report
        end subroutine getstats


!=======================
        subroutine preprocess            !set up for iterations
        use gridmodule                   !common grid info
```

```
        use iterationmodule             !common iteration counters and convergence info
        implicit none                   !enforce strong typing
        integer :: ix                   !dummy grid index
        write(*,*)
        write(*,*) "========== PREPROCESS is starting."
        write(*,*)
        write(*,*) "Hit RETURN to open a new OUTPUT file: "
        read(*,*)                       !wait until user responds
        open(2,file="",status="new")    !activate Macintosh file finder box
        write(2,*) (tab, xgrid(ix), ix=1,nx)             !first row list x grid locations
        write(2,*) "   0", (tab, agrid(ix), ix=1,nx)     !2nd row is first-guess analysis
        open(3,status="scratch",form="unformatted")      !open scratch file to temporarily hold obs analyses
        write(*,*)
        write(*,*) "Iteration, Correction Sum / Field Sum = RelativeError" !header for screen
        end subroutine preprocess


!========================
        subroutine refine               !improve the analysis estimate during one iteration
        use obsmodule                   !common observation variables
        use gridmodule                  !common grid variables
        use statmodule                  !common error and correlation statistics
        use iterationmodule             !common iteration variables
        implicit none                   !enforce strong typing
        real :: correction              !correction at any one point during an iteration
        real :: axj, aij                !amplification weights (maps obs error to correction)
        real :: rxj, rij, rjk           !correlation coefficients
        real :: sumw                    !sum of correlation coefs
        real :: varb                    !background (first-guess) error variance (assumed homogeneous*)
        real :: varobsi, varobsj        !error variances of observations at locations i & j
        real :: varratio                !ratio of error variances
        integer :: ix                   !dummy grid index
        integer :: iobs, jobs, kobs     !dummy observation indices
        sumcor = 0.0                    !initialize sum of corrections for this iteration
        sumf = 0.0                      !initialize sum of field values for this iteration
        varb = sigmax*sigmax            !error variance of first guess (assumed homogeneous*)
!       call interpolate                !find aobs by interpolating from grided analysis to obs locations
!Update the analysis estimate at the grid points (agrid)
        do ix=1,nx                                       !for each grid point
            correction = 0.0                             !initialize correction amount
            do jobs=1,nobs                               !for each obs location
                varobsj = sigmaobs(jobs)*sigmaobs(jobs)  !error variance of obs at location j
                sumw = varobsj/varb                      !initialize sum of the weights w
                do kobs=1,nobs                           !for each weight w
                    rjk = exp(-0.5*((xobs(kobs)-xobs(jobs))**2)/sigmaR2) !correl drop-off with j-k distance
                    sumw = sumw + abs(rjk)               !sum of the weights w
                enddo
                rxj = exp(-0.5*((xgrid(ix)-xobs(jobs))**2)/sigmaR2) !correlation drop-off with x-j distance
                axj = rxj/sumw                           !eq (15b), for optimum solution
                correction = correction + axj*(sobs(jobs)-aobs(jobs))   !RHS of eq (4)
            enddo
            agrid(ix) = agrid(ix) + correction           !eq (4)
            sumcor = sumcor + abs(correction)            !sum of correction magnitudes, to detect convergence
            sumf = sumf + abs(agrid(ix))                 !sum of analysis field magnitudes, ditto
        enddo
!Update the analysis estimate at the observation points (aobs)
        do iobs=1,nobs                                   !for each obs point
            correction = 0.0                             !initialize correction amount
            varobsi = sigmaobs(iobs)*sigmaobs(iobs)      !error variance of obs at location i
            do jobs=1,nobs                               !for each obs location
                varobsj = sigmaobs(jobs)*sigmaobs(jobs)  !error variance of obs at location j
                sumw = varobsj/varb                      !initialize sum of the weights w
                do kobs=1,nobs                           !for each weight w
                    rjk = exp(-0.5*((xobs(kobs)-xobs(jobs))**2)/sigmaR2) !correl drop-off with j-k distance
                    sumw = sumw + abs(rjk)               !sum of the weights w
                enddo
                rij = exp(-0.5*((xobs(iobs)-xobs(jobs))**2)/sigmaR2)    !correlation drop-off with x-j distance
                varratio = 0.0                           !initialize variance ratio
```

```
                if (iobs==jobs) varratio=varobsi/varb   !set variance ratio only for certain obs points
                aij = (rij+varratio)/sumw               !eq (15a), for optimum solution
                correction = correction + aij*(sobs(jobs)-aobs(jobs))   !RHS of eq (4')
            enddo
            aobsnew(iobs) = aobs(iobs) + correction     !eq (4')
        enddo
        aobs=aobsnew                                    !replace old analysis with new, at obs locations
        end subroutine refine


!=======================
        function done() result (converged)  !check if analysis has converged
        use iterationmodule                 !common iteration and convergence info
        use gridmodule                      !common grid info
        implicit none                       !enforce strong typing
        real :: relerror                    !relative error
        logical :: converged                !.true. if successive-corr has converged
        converged = .false.                 !initialize state to NOT converged
        relerror = sumcor/(sumcor+sumf)     !compute correction amount relative to the total value
        write(*,*) nu,sumcor,sumf,relerror  !display iteration status report
        write(*,*)
        converged = (relerror .LT. epsilon) !.true. when relative error is small enough
        end function done


!=======================
        subroutine savefields           !save results in a text file
        use iterationmodule             !common iteration variables
        use gridmodule                  !common grid variables
        use obsmodule                   !common observation variables
        implicit none                   !enforce strong typing
        integer :: ix, iobs             !dummy array indices
        write(2,*) " ",nu, (tab,agrid(ix),ix=1,nx)  !write gridded analysis to disk, for each iteration
        write(3) nu,aobs(1:nobs)        !write obs analysis to scratch file, for each iteration
        end subroutine savefields


!=======================
        subroutine postprocess          !postprocessing
        use iterationmodule             !common iteration variables
        use obsmodule                   !common observation variables
        implicit none                   !enforce strong typing
        integer :: iobs                 !dummy observation index
        integer :: ios                  !input/output status flag
        write(*,*)
        write(*,*) "========== POSTPROCESS is starting."
        write(*,*) "Successive correction data assimilation is finished."
        write(*,*)
!append the original observations to the end of the file
        write(2,*) (tab,xobs(iobs),iobs=1,nobs)         !obs x-locations
        write(2,*) "Obs", (tab,sobs(iobs),iobs=1,nobs)  !actual obs values
!append the various analysis estimates at the observation points
        rewind 3                        !reset the scratch file (unit 3) to beginning
        do                              !do for each successful iteration that was saved in scratch
            read(3,iostat=ios) nu,aobs(1:nobs)  !read obs values from scratch file
            if (ios==-1) exit                   !exit loop if end of file
            write(2,*) " ",nu, (tab,aobs(iobs),iobs=1,nobs) !append write obs analysis to disk, for each iteratic
        enddo
        close(2)                        !close files and terminate
        close(3)                        !close files and terminate
        end subroutine postprocess


!=====================================================================
!====+ the following is NOT needed for successive corrections ==============

        subroutine optimum              !statistical (optimum) interpolation
!First, this routine solves Bratseth eq (3) for linear weights p.
```

```
!It does this by forming a set of linear equations  A X = B  ,
!where A is the obs covariance matrix, X is the vector of weights p,
!and B is the vector of grid vs. obs correlations.
!(Uses LU decomposition for solution, modified from Num. Recipes, to solve for X)
!Then, the weights p are used in Bratseth (1) to solve for the analysis field.
        use obsmodule                   !common observation variables
        use gridmodule                  !common grid variables
        use statmodule                  !common statistical variables
        use optimumodule                !common optimum interp variables
        implicit none                   !enforce strong typing
        integer :: ix, iobs, jobs       !dummy indices
        real :: rij, rxj                !correlation coefficients
        real :: varratio                !ratio of error variances
        character (len=1) :: tab=achar(9)   !ascii tab character
        write(*,*)
        write(*,*) "========== OPTIMUM INTERPOLATION is starting."
        write(*,*)
        allocate(amatrix(nobs,nobs))    !dynamically allocate memory for covar matrix
        allocate(bmatrix(nobs))         !dynamically allocate memory for RHS matrix
        allocate(indx(nobs))            !dynamically allocate memory for LU index
        allocate(vv(nobs))              !dynamically allocate memory scrap matrix
        allocate(ogrid(nx))             !dynamically allocate memory for optimum gridded analysis
!Fill the symmetric matrix A of covariances.  (Move this subsection to inside the ix loop if sigmax varies)
        do iobs = 1,nobs                !for each i observation location
            do jobs = iobs,nobs         !for each j obs location
                rij = exp(-0.5*((xobs(iobs)-xobs(jobs))**2)/sigmaR2)    !correl drop-off with dist
                varratio = 0.0                              !initialize variance ratio
                if (iobs==jobs) varratio = (sigmaobs(iobs)*sigmaobs(iobs))/(sigmax*sigmax) !ratio of error varian
                amatrix(jobs,iobs) = rij + varratio         !fill upper triangle of A matrix
                amatrix(iobs,jobs) = amatrix(jobs,iobs)     !fill bottom triangle. symmetric.
            enddo
        enddo
        call ludcmp                     !replace amatrix with LU decomposition
!Compute the gridded analysis
        do ix=1,nx                      !for each grid point
!Fill the B matrix
            do jobs = 1,nobs            !for each observation point
                rxj = exp(-0.5*((xgrid(ix)-xobs(jobs))**2)/sigmaR2) !correl drop-off with distance
                bmatrix(jobs) = rxj     !B matrix
            enddo
!Solve A X = B for solution vector X, which represents the weights p
            call lubksb                 !solve for the linear weights, which are returned in bmatrix
!Use these weights to find the gridded analysis
            ogrid(ix) = 0.0             !initialize analysis increment
            do iobs = 1,nobs            !for each observation point
                ogrid(ix)=ogrid(ix)+bmatrix(iobs)*fobs(iobs)    !accumulate the analysis increment
            enddo
            ogrid(ix) = bgrid(ix) + ogrid(ix)   !get analysis by adding first-guess and increment
        enddo
        write(2,*) "OPTIMUM", (tab,ogrid(ix),ix=1,nx)   !write optimum analysis to disk
        print *, "...ending OPTIMUM"    !status report
        end subroutine optimum


!==================
        SUBROUTINE ludcmp               !Lower-upper matrix decomposition
!Based on Numerical Recipes routine. (Modified to take I/O via modules.)
!Solves a set of linear algebraic equations:  A X = B.
!This routine takes matrix A as input, and overwrites it with a LU decomposed matrix
!This LU version of A is then used by subroutine lubksb to solve for X.
!(for details: Press et al, 1992, Numerical Recipes in FORTRAN, 2Ed, Cambridge U Press)
        use obsmodule                   !common observation variables
        use optimumodule                !common optimum interp variables
        implicit none                   !enforce strong typing
        INTEGER :: i,imax,j,k
        REAL :: aamax,dum,sum
        do i=1,nobs
            aamax=0.
```

```
            do j=1,nobs
                if (abs(amatrix(i,j)).gt.aamax) aamax=abs(amatrix(i,j))
            enddo
        if (aamax.eq.0.) pause 'singular matrix in ludcmp'
        vv(i)=1./aamax
        enddo
        do j=1,nobs
            do i=1,j-1
                sum=amatrix(i,j)
                do k=1,i-1
                    sum=sum-amatrix(i,k)*amatrix(k,j)
                enddo
                amatrix(i,j)=sum
            enddo
            aamax=0.
            do i=j,nobs
                sum=amatrix(i,j)
                do k=1,j-1
                    sum=sum-amatrix(i,k)*amatrix(k,j)
                enddo
                amatrix(i,j)=sum
                dum=vv(i)*abs(sum)
                if (dum.ge.aamax) then
                    imax=i
                    aamax=dum
                endif
            enddo
            if (j.ne.imax)then
                do k=1,nobs
                    dum=amatrix(imax,k)
                    amatrix(imax,k)=amatrix(j,k)
                    amatrix(j,k)=dum
                enddo
                vv(imax)=vv(j)
            endif
            indx(j)=imax
            if(amatrix(j,j).eq.0.) amatrix(j,j)=TINY
            if(j.ne.nobs)then
                dum=1./amatrix(j,j)
                do i=j+1,nobs
                    amatrix(i,j)=amatrix(i,j)*dum
                enddo
            endif
        enddo
        end subroutine ludcmp


!===============
        SUBROUTINE lubksb                !Lower-upper matrix back substitution
!Based on Numerical Recipes routine. (Modified to take I/O via modules.)
!Solves a set of linear algebraic equations:  A X = B.
!Uses as input the LU decomposition from subroutine ludcmp, which was overwritten into amatrix.
!bmatrix holds eq RHS vector as input, but returns solution vector X as output
!(for details: Press et al, 1992, Numerical Recipes in FORTRAN, 2Ed, Cambridge U Press)
        use obsmodule                    !common observation variables
        use optimumodule                 !common optimum interp variables
        implicit none                    !enforce strong typing
        INTEGER :: i,ii,j,ll
        REAL :: sum
        ii=0
        do i=1,nobs
            ll=indx(i)
            sum=bmatrix(ll)
            bmatrix(ll)=bmatrix(i)
            if (ii.ne.0)then
                do j=ii,i-1
                    sum=sum-amatrix(i,j)*bmatrix(j)
                enddo
```

```
        else if (sum.ne.0.) then
            ii=i
        endif
        bmatrix(i)=sum
    enddo
    do i=nobs,1,-1
        sum=bmatrix(i)
        do j=i+1,nobs
            sum=sum-amatrix(i,j)*bmatrix(j)
        enddo
        bmatrix(i)=sum/amatrix(i,i)
    enddo
    end subroutine lubksb
```