C NCLFORTSTART

subroutine cremapbin(plev ,plato ,plono ,plat ,plon ,

1 xx ,yy ,clat ,clon ,clato,

2 clono ,nlat ,nlato ,bin\_factor ,

3 xxmsg )

c

c--------1---------2---------3---------4---------5---------6---------7--

c

c Grid-Box Binning

c

c-----------------------------------------------------------------------

c

implicit none

c

c-----------------------------------------------------------------------

c

integer plev ! vertical dimension of input/output field

integer plato ! latitude dimension of output field

integer plono ! longitude dimension of output field

integer plat ! latitude dimension of input field

integer plon ! longitude dimension of input field

c

double precision xx(plon ,plat ,plev) ! input analysis field

double precision yy(plono,plato,plev) ! horizontally interpolated

c ! (output) field

double precision clat (plat ) ! Input latitude in degrees

c ! oriented S->N

double precision clon (plon ) ! Input longitude in degrees

c ! oriented W->E

double precision clato(plato) ! Output latitude in degrees

c ! oriented S->N

double precision clono(plono) ! Output longitude in degrees

c ! oriented W->E

integer nlat ! Number of Global Gaussian latitudes (input)

integer nlato ! Number of Global Gaussian latitudes (output)

double precision bin\_factor ! bin-box area expansion/contraction factor

c ! relative to output grid-box area.

double precision xxmsg

c

c-----------------------------------------------------------------------

c

C NCLEND

c

c---------------------------Local workspace-----------------------------

c

c ! Max # of box segments

integer max\_segs

parameter (max\_segs = 100000 )

c

integer i, j, ii, jj, k, jfirst, jfirsto ! Indices

integer nx, ny, nx\_max, ny\_max

integer plon2, plonhalf

integer i\_in(max\_segs),i\_out(max\_segs)

integer j\_in(max\_segs),j\_out(max\_segs)

integer grid\_flag, grido\_flag ! grid flags: 0=Regular, 1=Gaussian

double precision xx\_loc(plon\*2, plat, plev)

double precision pi, pio180, pio2, factor

double precision flat (plat) ,flon (plon\*2 )

double precision flato (plato ),flono (plono )

double precision flati (plat+1), floni (plon\*2+1)

double precision flatoi(plato+1),flonoi(plono+1 )

double precision tmps, tmpn, tmp(plono,plato)

double precision edge\_w (plon\*2), edge\_e (plon\*2), edge\_s (plat ),

& edge\_n (plat )

double precision edgeo\_w(plono ), edgeo\_e(plono ), edgeo\_s(plato),

& edgeo\_n(plato)

double precision sin\_s (plat ),sin\_n (plat )

double precision sino\_s(plato),sino\_n(plato)

double precision dx(max\_segs), dy(max\_segs)

double precision distmin, dist, zero, three

double precision dlat, dlato, eps

c

c - the following are only relevant for grids that are Gaussian

c

double precision, allocatable :: flat\_glob (:) ! Global Gaussian latitudes (based on input grid resolution)

c ! (radians)

double precision, allocatable :: flato\_glob(:) ! Global Gaussian latitudes (based on output grid resolution)

c ! (radians)

double precision, allocatable :: gw\_glob (:) ! Global Gaussian weights (based on input grid resolution)

double precision, allocatable :: gwo\_glob (:) ! Global Gaussian weights (based on output grid resolution)

integer ierror

c

c-----------------------------------------------------------------------

c

zero = 0.d0

three = 3.d0

plon2 = plon\*2

plonhalf = plon/2

pi = 4.d0\*atan(1.d0)

pio180 = pi/180.d0

pio2 = pi/2.d0

eps = 1.d-5

c

c Sanity checks

c

c djs if(bin\_factor .lt. 0.05d0) then

if(bin\_factor .lt. 1.00d0) then

write(6,\*) 'ERROR ("CREMAPBIN"): binning factor out of range'

write(6,\*) 'bin\_factor = ', bin\_factor

call abort

end if

if(clat(3) - clat(2) .lt. zero) then

write(6,\*) 'ERROR ("CREMAPBIN"): Input latitudes oriented'

write(6,\*) ' N->S. Should be S->N'

call abort

end if

if(clato(3) - clato(2) .lt. zero) then

write(6,\*) 'ERROR ("CREMAPBIN"): Output latitudes oriented'

write(6,\*) ' N->S. Should be S->N'

call abort

end if

if(clon(3) - clon(2) .lt. zero) then

write(6,\*) 'ERROR ("CREMAPBIN"): Input longitudes oriented'

write(6,\*) ' E->W. Should be W->E'

call abort

end if

if(clono(3) - clono(2) .lt. zero) then

write(6,\*) 'ERROR ("CREMAPBIN"): Output longitudes oriented'

write(6,\*) ' E->W. Should be W->E'

call abort

end if

c

c Determine if input/output grids are Regular or Gaussian

c

dlat = ( clat (plat ) - clat (1) ) /(plat -1)

dlato = ( clato(plato) - clato(1) ) /(plato-1)

grid\_flag = 0

grido\_flag = 0

do j = 1,plat-1

if( abs (clat (j+1) - clat (j) - dlat ) .gt. eps) grid\_flag = 1

end do

do j = 1,plato-1

if( abs (clato(j+1) - clato(j) - dlato) .gt. eps) grido\_flag =1

end do

c

c Get global lats/weights for those grids that are Gaussian

c

allocate( flat\_glob(nlat) )

allocate( gw\_glob (nlat) )

if(grid\_flag .eq. 1) then

if(nlat .lt. plat) then

write(6,\*) 'ERROR ("CREMAPBIN"): number of latitudes for '

write(6,\*) 'the input grid cannot be greater than the '

write(6,\*) 'global number of latitudes for that grid '

write(6,\*) 'resolution'

write(6,\*) 'nlat, plat = ', nlat, plat

call abort

end if

call binning\_get\_global\_lats\_wgts(nlat, flat\_glob, gw\_glob)

end if

c

allocate( flato\_glob(nlato) )

allocate( gwo\_glob (nlato) )

if(grido\_flag .eq. 1) then

if(nlato .lt. plato) then

write(6,\*) 'ERROR ("CREMAPBIN"): number of latitudes for '

write(6,\*) 'the output grid cannot be greater than the '

write(6,\*) 'global number of latitudes for that grid '

write(6,\*) 'resolution'

write(6,\*) 'nlato, plato = ', nlato, plato

call abort

end if

call binning\_get\_global\_lats\_wgts(nlato, flato\_glob, gwo\_glob)

end if

c

c Copy input data to wrap-around array (wrap half-way around globe

c at each end of x-direction)

c

do k = 1,plev

do j = 1,plat

ii = plonhalf

do i = 1,plon2

ii = ii + 1

if(ii .gt. plon) ii = 1

xx\_loc(i,j,k) = xx(ii,j,k)

end do

end do

end do

c

c Convert input/output grid coordinates to radians (wrap half-way around

c globe at each end of x-direction of input grid)

c

ii = plonhalf

do i = 1,plon2

ii = ii + 1

if(ii .gt. plon ) ii = 1

if(i .le. plonhalf) flon(i) = clon(ii)\*pio180-4\*pio2

if(i .gt. plonhalf+plon) flon(i) = clon(ii)\*pio180+4\*pio2

if(i .gt. plonhalf .and. i .le. plonhalf+plon)

& flon(i) = clon(ii)\*pio180

end do

do j = 1,plat

flat (j) = clat (j)\*pio180

end do

do i = 1,plono

flono(i) = clono(i)\*pio180

end do

do j = 1,plato

flato(j) = clato(j)\*pio180

end do

c

c Map "regional" latitudes into global latitude arrays for input/output grids

c

if(grid\_flag .eq. 1) then

call binning\_map\_lats(nlat , plat , flat , flat\_glob , jfirst )

end if

if(grido\_flag .eq. 1) then

call binning\_map\_lats(nlato, plato, flato, flato\_glob, jfirsto)

end if

c

c Compute box edges for input and output grids

c

call binning\_map\_edges(plat , plon2 , nlat , jfirst ,

1 flon , flat , gw\_glob ,

2 grid\_flag , floni , flati )

call binning\_map\_edges(plato , plono , nlato , jfirsto ,

1 flono , flato , gwo\_glob ,

2 grido\_flag, flonoi, flatoi )

c

c Copy grid interfaces to "edge" arrays

c

do i = 1,plon\*2

edge\_w(i) = floni(i )

edge\_e(i) = floni(i+1)

end do

do j = 1,plat

edge\_s(j) = flati(j )

edge\_n(j) = flati(j+1)

sin\_s (j) = sin(edge\_s(j))

sin\_n (j) = sin(edge\_n(j))

end do

c

c Expand/contract bin box area for each output grid box by "bin\_factor"

c

factor = sqrt(bin\_factor)

do i = 1,plono

edgeo\_w(i) = flono(i) - ( flono (i ) - flonoi(i) )\*factor

edgeo\_e(i) = flono(i) + ( flonoi(i+1) - flono (i) )\*factor

end do

do j = 1,plato

tmps = flato(j) - ( flato (j ) - flatoi(j) )\*factor

tmpn = flato(j) + ( flatoi(j+1) - flato (j) )\*factor

edgeo\_s(j) = max( tmps, -pio2) - max( ( tmpn - pio2), zero)

edgeo\_n(j) = min( tmpn, pio2) + max( (-pio2 - tmps), zero)

sino\_s (j) = sin(edgeo\_s(j))

sino\_n (j) = sin(edgeo\_n(j))

end do

c

c Make vector of box segments in x-direction

c

nx = 0

do i = 1,plono

do ii = 1,plon\*2

if(edge\_e (ii) .gt. edgeo\_w( i) .and.

& edgeo\_e( i) .gt. edge\_w (ii) ) then

nx = nx + 1

if(nx .gt. max\_segs) then

write(6,\*) 'ERROR ("CREMAPBIN"): number of box'

write(6,\*) 'segments greater than "max\_segs"'

call abort

end if

i\_in (nx) = ii

i\_out(nx) = i

dx (nx) = min(min(min(edge\_e(ii)-edge\_w(ii),

& edgeo\_e(i)-edgeo\_w(i) ),

& edge\_e(ii)-edgeo\_w(i) ),

& edgeo\_e(i)-edge\_w(ii) )

end if

if(edge\_w (ii) .ge. edgeo\_e( i)) exit

end do

end do

c

c Make vector of box segments in y-direction

c

ny = 0

do j = 1,plato

do jj = 1,plat

if(edge\_n (jj) .gt. edgeo\_s( j) .and.

& edgeo\_n( j) .gt. edge\_s (jj) ) then

ny = ny + 1

if(ny .gt. max\_segs) then

write(6,\*) 'ERROR ("CREMAPBIN"): number of box'

write(6,\*) 'segments greater than "max\_segs"'

call abort

end if

j\_in (ny) = jj

j\_out(ny) = j

distmin = edge\_n(jj)-edge\_s(jj)

dy(ny) = sin\_n (jj)-sin\_s (jj)

dist = edgeo\_n(j)-edgeo\_s(j)

if(dist .lt. distmin) then

distmin = dist

dy(ny) = sino\_n(j)-sino\_s(j)

end if

dist = edge\_n(jj)-edgeo\_s(j)

if(dist .lt. distmin) then

distmin = dist

dy(ny) = sin\_n(jj)-sino\_s(j)

end if

dist = edgeo\_n(j)-edge\_s(jj)

if(dist .lt. distmin) then

distmin = dist

dy(ny) = sino\_n(j)-sin\_s(jj)

end if

end if

if(edge\_s (jj) .ge. edgeo\_n( j)) exit

end do

end do

nx\_max = nx

ny\_max = ny

c

c Begin weighted binning

c

do k = 1,plev

do j = 1,plato

do i = 1,plono

yy(i,j,k) = 0.

end do

end do

end do

do k = 1,plev

do ny = 1,ny\_max

j = j\_out(ny)

jj = j\_in (ny)

do nx = 1,nx\_max

i = i\_out(nx)

ii = i\_in (nx)

yy(i,j,k) = yy(i,j,k) + xx\_loc(ii,jj,k)\*dx(nx)\*dy(ny)

end do

end do

end do

c

c Normalize

c

do j = 1,plato

do i = 1,plono

tmp(i,j) = (edgeo\_e(i) - edgeo\_w(i))\*(sino\_n(j) - sino\_s(j))

end do

end do

do k = 1,plev

do j = 1,plato

do i = 1,plono

yy(i,j,k) = yy(i,j,k)/tmp(i,j)

end do

end do

end do

c

deallocate( flat\_glob )

deallocate( gw\_glob )

deallocate( flato\_glob )

deallocate( gwo\_glob )

c

c CRUDE ....

c . At any level where the input "xx" has a missing value

c . set the corresponding "yy" level to missing.

c

do k = 1,plev

do j = 1,plat

do i = 1,plon

if (xx(i,j,k).eq.xxmsg) then

do jj = 1,plato

do ii = 1,plono

yy(ii,jj,k) = xxmsg

end do

end do

end if

go to 100

end do

end do

100 continue

end do

return

end

c

c-----------------------------------------------------------------------

c-----------------------------------------------------------------------

c

subroutine binning\_get\_global\_lats\_wgts(nlat, lat\_glob, gw\_glob)

c

c--------1---------2---------3---------4---------5---------6---------7--

c

c Compute Global Gaussian latitudes/weights based upon # of latitudes

c

c-----------------------------------------------------------------------

c

implicit none

c

c-----------------------------------------------------------------------

c

integer nlat ! Number of Global Gaussian latitudes

double precision lat\_glob (nlat) ! Global Gaussian latitudes (radians)

double precision gw\_glob (nlat) ! Global Gaussian weights

c

c---------------------------Local workspace-----------------------------

c

integer ierror, lwork

double precision pio2

double precision, allocatable :: work(:) ! Work array

c

c-----------------------------------------------------------------------

c

pio2 = 2.d0\*atan(1.d0)

c

if(nlat .le. 2) then

write(6,\*) 'Error in "cremapbin": Not enough Gaussian latitudes'

write(6,\*) 'nlat = ', nlat

call abort

end if

c

lwork = 4\*nlat\*(nlat+1)+2

allocate( work(lwork) )

call gaqdncl(nlat,lat\_glob,gw\_glob,work,lwork,ierror)

deallocate( work )

if(ierror .ne. 0) then

write(6,\*)

write(6,\*) 'Error: in call to routine "gaqdncl", ierror = ',

& ierror

if(ierror .eq. 1) then

write(6,\*) "Not enough work space declared for number of"

write(6,\*) "Gaussian latitudes"

write(6,\*) 'lwork, nlat = ', lwork,nlat

write(6,\*) 'lwork should be = ', 4\*nlat\*(nlat+1)+2

end if

call abort

end if

c

lat\_glob(:) = lat\_glob(:) - pio2

c

return

end

c

c-----------------------------------------------------------------------

c-----------------------------------------------------------------------

c

subroutine binning\_map\_lats(nlat, plat, flat, flat\_glob, jfirst)

c

c--------1---------2---------3---------4---------5---------6---------7--

c

c Map "regional" latitudes into global latitude arrays for input/output grids

c and check that the grid latitudes are an identical subset of the global array

c

c-----------------------------------------------------------------------

c

implicit none

c

c-----------------------------------------------------------------------

c

integer nlat ! Number of Global Gaussian latitudes

integer plat ! Number of grid Gaussian latitudes

integer jfirst ! index of Global lat array that maps

c ! into the first grid lat array

double precision flat\_glob(nlat) ! Global Gaussian latitudes (radians)

double precision flat (plat) ! grid Gaussian latitudes (radians)

c

c---------------------------Local workspace-----------------------------

c

integer j, jj

double precision eps

logical found

c

c-----------------------------------------------------------------------

c

eps = 1.d-5

c

c Find latitude in Global array that corresponds to the first latitude

c of the grid array.

c

found = .false.

jfirst = 0

do j = 1,nlat

if( abs(flat\_glob(j) - flat(1)) .lt. eps ) then

found = .true.

jfirst = j

exit

end if

end do

c

if(.not. found) then

write(6,\*) 'Error in "cremapbin":'

write(6,\*) "Could not map global lat array into grid array"

call abort

end if

c

if(plat+jfirst-1 .gt. nlat) then

write(6,\*) 'Error in "cremapbin":'

write(6,\*) "Stepping out of bounds of the global lat array"

call abort

end if

c

c Test that subsequent grid lats all match the global lat array

c

do j = 2,plat

if( abs(flat\_glob(j+jfirst-1) - flat(j)) .gt. eps ) then

write(6,\*) 'Error in "cremapbin":'

write(6,\*) "Gaussian latitudes in grid array do not"

write(6,\*) "match those in the global array"

call abort

end if

end do

c

return

end

c

c-----------------------------------------------------------------------

c-----------------------------------------------------------------------

c

subroutine binning\_map\_edges(plat , plon , nlat , jfirst ,

1 flon , flat , gw\_glob ,

2 grid\_flag, floni, flati )

c

c--------1---------2---------3---------4---------5---------6---------7--

c

c Based on input grid, compute grid-box edges for either Gaussian or

C Regular (evenly-spaced) grids.

c

c-----------------------------------------------------------------------

c

implicit none

c

c-----------------------------------------------------------------------

c

integer plat ! latitude dimension of input field

integer plon ! longitude dimension of input field

integer nlat ! Number of Global Gaussian latitudes

integer jfirst ! Index of global Gaussian lat array mapped into the first grid lat array

c

double precision flon (plon) ! longitudes in radians oriented W->E

double precision flat (plat) ! latitudes in radians oriented S->N

double precision gw\_glob (nlat) ! Global Gaussian weights

integer grid\_flag ! grid flags: 0=Regular, 1=Gaussian

double precision floni (plon+1) ! longitudes of box edges in radians oriented W->E

double precision flati (plat+1) ! latitudes of box edges in radians oriented S->N

c

c---------------------------Local workspace-----------------------------

c

integer i, j, platp1 ! Indices

double precision sum

double precision pi, pio2, half, one, two, three

c

c-----------------------------------------------------------------------

c

platp1 = plat + 1

half = 0.5d0

one = 1.d0

two = 2.d0

three = 3.d0

pi = 4.d0\*atan(one)

pio2 = pi/two

c

c Compute longitudes of box edges

c

floni( 1) = ( three\*flon( 1) - flon( 2) )\*half

floni(plon+1) = ( three\*flon(plon) - flon(plon-1) )\*half

do i = 2,plon

floni(i) = half\*(flon(i-1) + flon(i))

end do

c

c If Regular grid, use algebraic mean to determine latitudes of box edges (extrapolation for endpoints)

c Else, if Gaussian grid, use partial sums of Gaussian weights.

c

if(grid\_flag .eq. 0) then

flati(1 ) = ( three\*flat(1) - flat(2) )\*half

flati(1 ) = max( flati(1), -pio2)

flati(platp1) = ( three\*flat(plat) - flat(plat-1) )\*half

flati(platp1) = min( flati(platp1), pio2)

do j = 1,plat-1

flati(j+1) = half\*(flat(j) + flat(j+1))

end do

else

c

c Sum Gaussian weights up to first latitude of data grid to get first box edge

c

sum = 0.d0

if(jfirst .le. 1) then

flati(1) = -pio2

else

do j = 1,jfirst-1

sum = sum + gw\_glob(j)

end do

flati(1) = asin( sum-one )

end if

c

c Determine subsequent box edges

c

do j = 1,plat

sum = sum + gw\_glob(jfirst+j-1)

flati(j+1) = asin( min (one,(sum-one) ) )

end do

end if

c

return

end

c

c-----------------------------------------------------------------------

c