EMEP Projection and Interpolation Package

Heiko Klein

December 5, 2003

Contents

1 Projection

The EMEP projection is a polar-stereographic projection with a cell-size of (50km x 50km) at $\lambda_0 = 60°$ latitude with a longitude of $\phi_0 = 32°$ parallel to the y-axis.

1.1 polarStereographicTolonglat

The functions to convert from polar-stereographic coordinates to spherical coordinates (in degree) are packed together in the module *polarStereographicTolonglat*

The package estimates the earth-radius as 6370km.

1.1.1 Functions and Subroutines

Name	**Description**
define_phi0 (real phi0)	Longitude ϕ_0 in degree, default: 60°
define_lambda0 (real lambda0)	Latitude λ_0 in degree, default: 32°
define_ps_grid_distance (real distance)	grid-distance in km at ϕ_0, default: 50km
define_ps_northpole (real x, real y)	define the position of the northpole in coordinates of the projection, default: 8, 110
define_emep50official()	defines the parameters corresponding to the (132x111) official emep domain. This is the default.
define_emep50model()	defines the parameters corresponding to the (170x132) emep model domain. The position of the northpole is (43, 121)
define_emep150()	defines the parameters corresponding to the former emep150 grid with a distance of 150km and a position of the northpole at (3,37)
polarStereographic2longlat (real x, real y, real long, real lat)	converts the polar-stereographic coordinates x and y to the spherical coordinates long and lat.
longlat2polarStereographic (real long, real lat, real x, real y)	converts the spherical coordinates long and lat to the polar-stereographic projection coordinates x and y.
emep_coordinate2cell (real x, real y, integer i, integer j)	converts the real coordinate (x,y) to the corresponding cell (i,j). A grid-cell is defined by its center coordinate.

1.2 Other Projections

Please send us other projections used to convert to or from the emep coordinates. They will be included into the projections package and might be useful for other users. Please send to emep.mscw@met.no.

2 Interpolation

When changing from one grid projection to another, not only the coordinates, but also the cell-sizes and positions are usually changed. Therefore, an interpolation-mechanism has to be used.

The most commonly used interpolation methods are:

- bilinear
- cubic convolution
- nearest neighbour

The bilinear and the cubic convolution interpolation handle continuous data and assume a certain data-distribution within a cell in correspondence to neighbouring cells. Since the emissions are not continuous but localized, those interpolations cannot be used. Even for depositions the localized emissions play an important role and these interpolations should not be used.

For localized data, the *nearest neighbour* interpolation is used. Usual implementations of this method, i.e. Grass (`r.proj`), use the center of each output-gridcell and add the nearest cell of the input-grid. This makes sure, that all output-gridcells are set, but do not garantee mass-consistence since some input-gridcells might not have been used, or some input-gridcells might have been used several times. Furthermore, the change of size of the gridcells is not respected.

The method used in the *reverseNextNeighbour* module is related to the *nearest neighbour* interpolation, but solves its deficiencies. Mass-consistence is garanteed by working in the opposite way, taking each input-cell and adding its contents to the closest output-cell. To respect the change of size of grid-cells, the input-grid can be subdivided in smaller cells (usually 100), which will be interpolated. The subdivision in smaller cells will in addition solve the problem of undefined output-cells.

2.1 reverseNextNeighbour

The Fortran90 module *reverseNextNeighbour* assumes, that the input grid-cells are defined by its center, i.e. the cell named (i,j) has the corners (i-0.5, j-0.5) and (i+0.5, j+0.5).

2.1.1 Interfaces

To use the module, the user needs to implement the following interfaces:

projection(xi, yi, xo, yo)	Projection subroutine converting coordinates (xi, yi) to (xo, yo). The `polarStereographic2longlat` or `lonlat2polarStereographic` subroutines from section 1.
coord2array(x, y, i, j)	A mapping function to convert a coordinate (x,y) to the cell of the output-grid. The `emep_coordinate2cell` is an example.

2.1.2 Functions and Subroutines

2.1.3 Example

```
program testInterpol
  use polarStereographicTolonglat
  use reverseNextNeighbour
  implicit none
  interface
     subroutine coord2array(x, y, ix, iy)
         real, intent(in) :: x, y
```

Name	**Description**
reverseNNInterpol (subroutine projection, subroutine coord2array, character(len = 8) sumAverage, real(:,:) inArray, real(:,:) outArray)	Interpolates the map given by inArray with the given projection to outArray. sumAverage should be `sum` for *emissions* and *depositions* and 'average' for *concentrations*.
setSubsteps(integer i)	set the number of substeps used in each direction. Defaults to 10 (= 100 subcells)
setInfo (logical l)	write some information after the interpolation

```
         integer, intent(out) :: ix, iy
     end subroutine coord2array
  end interface

  real, dimension(170,132)  :: eSox
  real, dimension(180, 90)  :: llArray

  real :: ei, ej, long, lat
  real, dimension(13) :: rowvalues
  integer :: i, j,ilong, ilat,cc
  character(len=8) :: sumAverage


! reading the input file in emep model coordinates
  eSOx = 0.
  open (2, file='gridSOx', status='old', action="read")
  do  ! while (.true.)
     read (2, end=10, *), cc, i,j, rowvalues
     eSOx(i,j) = eSOx(i,j) + sum(rowvalues(3:13))
  end do
10 continue

! declare the polarStereographicTolonglat to be in emep50model modus
  call define_emep50model()
  sumAverage = 'sum'
  call setInfo(.true.)

! interpolation the grids from emep50model to spherical coordinates
  call reverseNNInterpol(polarStereographic2longlat, coord2array, sumAverage,&
       eSOx, llArray)

! printing the
  do i = 1, size(llArray, 1)
     do j = 1, size(llArray, 2)
        call array2coord(i,j, long, lat)
        print *, long, lat, llArray(i,j)
     end do
  end do


end program
```

```
! implementing the function mapping longitude latitude coordinates
! to 0.5 x 0.5 longitude latitude cells with array(1,1) containing
! the cell(-29.5, 30.5)
subroutine coord2array(x, y, ix, iy)
  real, intent(in) :: x, y
  integer, intent(out) :: ix, iy

  ix = nint((x+30) * 2.)
  iy = nint((y-30) *2.)
end subroutine coord2array


! translating cell-indices back to spherical coordinates
subroutine array2coord(ix, iy, x, y)
  integer, intent(in) :: ix, iy
  real, intent(out) :: x, y

  x = (ix / 2.) -30
  y = (iy / 2.) + 30
end subroutine array2coord
```

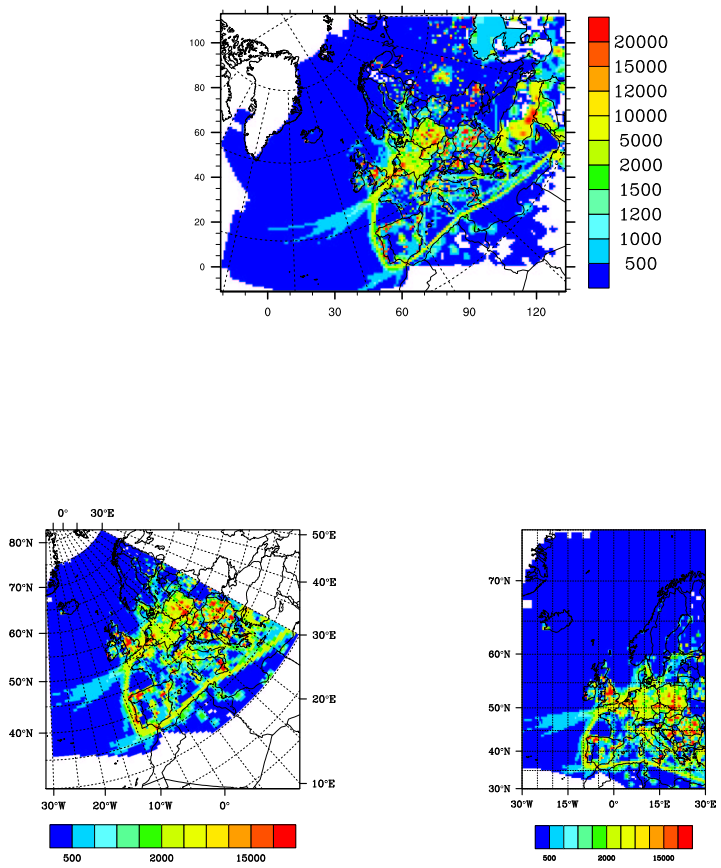

Figure 1: EMEP SO_x emissions in different projections: First the original emep50 polar-stereographic projection. Second the data in spherical coordinates (0.5x0.5°) plottet over a polar-stereographic projection. Third: Spherical coordinates plotted over a mercator projection.