A User's Guide for SCRIP: A *S*pherical *C*oordinate *R*emapping and *I*nterpolation *P*ackage

Version 1.4

Philip W. Jones
Theoretical Division
Los Alamos National Laboratory

Contents

Chapter 1

Introduction

SCRIP is a software package used to generate interpolation weights for remapping fields from one grid to another in spherical geometry. The package currently supports four types of remappings. The first is a conservative remapping scheme that is ideally suited to a coupled model context where the area-integrated field (e.g. water or heat flux) must be conserved. The second type of mapping is a basic bilinear interpolation which has been slightly generalized to perform a local bilinear interpolation. A third method is a bicubic interpolation similar to the bilinear method. The last type of remapping is a distance-weighted average of nearest-neighbor points. The bilinear and bicubic schemes can only be used with logically-rectangular grids; the other two methods can be used for any grid in spherical coordinates.

SCRIP is available via the web at
http://climate.acl.lanl.gov/software/SCRIP/
NOTE: This location has changed since the 1.2 release.

The next chapter describes how to compile and run SCRIP, while the following sections describe the remapping methods in more detail.

Chapter 2

Compiling and Running

The distribution file is a gzipped tarfile, so you must uncompress the file using "gunzip" and then extract SCRIP from the tar file using "tar -xvf scrip1.4.tar". The extraction process will create a directory called SCRIP with two subdirectories named "source" and "grids". The source directory contains all the source files and the makefile for building the package. The grids directory contains some sample grid files and routines for creating or converting other grid files to the proper format.

2.1 Compiling

In order to compile SCRIP, you need access to a Fortran 90 compiler and a netCDF library (version 3 or later). In the source directory, you must edit the makefile to insert the appropriate compiler and compiler options for whatever machine you happen to work on. The makefile currently only has SGI compiler options. In addition, you must edit the paths in the makefile to find the proper netCDF library - if netCDF is in your default path, you may delete the path altogether.

Once the makefile has been edited to reflect your local environment, type "make" and let it build. By default, the makefile builds two executables in the main SCRIP directory; the first executable is called scrip and computes all the necessary weights for a remapping. The second executable is a simple test code scrip_test which will test the weights output by scrip.

Figure 2.1: Required input namelist.

```
&remap_inputs
    num_maps = 2
    grid1_file = 'grid_1_file_name'
    grid2_file = 'grid_2_file_name'
    interp_file1 = 'map_1_output_file_name'
    interp_file2 = 'map_2_output_file_name'
    map1_name = 'name_for_mapping_1'
    map2_name = 'name_for_mapping_2'
    map_method = 'conservative'
    normalize_opt = 'frac'
    output_opt = 'scrip'
    restrict_type = 'latitude'
    num_srch_bins = 90
    luse_grid1_area = .false.
    luse_grid2_area = .false.
/
```

2.1.1 Compile-time Parameters

There are a few compile-time parameters that can be changed before compiling (see Table 2.1). For the most part, the defaults are adequate, but you may wish to change these at some point. The use of these parameters will become clear in the chapters describing the remapping algorithms.

2.2 Running

Once the code is compiled, a few input files are needed. The first is a namelist input file and the other two required files are grid description files.

2.2.1 Namelist Input

The namelist input file must be called scrip_in and contain a namelist as shown in Fig. 2.1.

Table 2.1: Compile-time parameters

Routine	Parameter Name	Default Value	Description
remap_conserv.f	north_thresh	1.42	threshold latitude (in radians) above which a coordinate transformation is used to perform intersection calculation
remap_conserv.f	south_thresh	-2.00	same for south pole
remap_conserv.f	max_subseg	10000	maximum number of sub-segments allowed (to prevent infinite loop)
remap_bilinear.f	max_iter	100	max number of iterations to determine local i,j
remap_bilinear.f	converge	1×10^{-10}	convergence criterion for bilinear iteration
remap_bicubic.f	max_iter	100	max number of iterations to determine local i,j
remap_bicubic.f	converge	1×10^{-10}	convergence criterion for bicubic iteration
remap_distwgt.f	num_neighbors	4	number of nearest neighbors to use for distance-weighted average
iounits.f	stdin	5	I/O unit reserved for standard input
iounits.f	stdout	6	I/O unit reserved for standard output
iounits.f	stderr	6	I/O unit reserved for standard error output
timers.f	max_timers	99	max number of CPU timers

The num_maps variable determines the number of mappings to be computed. If you'd like mappings only from a source grid (grid 1) to a destination grid (grid 2), then num_maps should be set to one. If you'd also like weights for a remapping in the opposite direction (grid 2 to grid 1), then num_maps should be set to two.

The map_method variable determines the method to be used for the mapping. A conservative remapping is map_method 'conservative'; a bilinear mapping is map_method 'bilinear'; a distance-weighted average is map_method 'distwgt'; a bicubic mapping is map_method 'bicubic'.

The restrict_type variable and num_srch_bins determines how the software restricts the range of grid points to search to avoid a full N^2 search. There are currently two options for restrict_type: 'latitude' and 'latlon'. The first was used in all previous versions of SCRIP and restricts the search by dividing the grid points into num_srch_bins latitude bins. The 'latlon' choice divides the spherical domain into latitude-longitude boxes and thus provides a way to restrict the longitude range as well. Note that for the latlon option, the domain is divided by num_srch_bins in *each* direction so that the total number of bins is the square of num_srch_bins. Generally, the larger the number of bins, the more the search can be restricted. However if the number of bins is too large, more time will be taken restricting the search than the search itself. For coarse grids, choosing the latitude option with 90 bins (one degree bins) is sufficient.

The normalize_opt variable is used to choose the normalization of the remappings for the conservative remapping method. By default, normalize_opt is set to be 'fracarea' and will include the destination area fraction in the output weights; other options are 'none' and 'destarea' (see chapter on the conservative remapping method). The latter two are useful when dealing with masks that are dynamic (e.g. variable ice fraction). Keep in mind that in such a case, the area fractions must be computed explicitly by the remapping routine at the time the remappings are actually computed (see the example in Fig. 2.4).

The grid$x$_file are names (with relative paths) of the grid input files. The first grid file (grid1_file) *must* be the source grid if num_maps=1. If this mapping uses the conservative remapping method, the first grid file must also be the grid with the master mask (e.g. a land mask) – grid fractions on the second grid will be determined by this mask.

Names of the output files for the remapping weights are determined by the interp_file$x$ filenames (again with paths). Map 1 refers to a mapping from grid 1 to grid 2; map 2 is in the opposite direction.

A descriptive name for the remappings are determined by the map$x$_name variables. These should be descriptive enough to know exactly which grids and methods were used.

The output_opt variable determines the format of the netCDF output file. The two currently-supported options are 'scrip' and 'ncar-csm'. The latter is to generate files for use in the NCAR CSM Flux Coupler for coupled climate modeling. The primary difference between the formats is the choice of variable names.

The two logical flags luse_grid$x$_area are for using an input area to normalize the conservative weights. If these are set to true, the input grid files must contain the grid areas. This option is provided primarily for making the weights consistent with internal model-computed areas (which may differ somewhat from the SCRIP-computed areas).

2.2.2 Grid Input Files

The grid input files are in netCDF format as shown by the sample ncdump grid output in Fig. 2.2. If you're unfamiliar with ncdump output, it's important to not that ncdump shows the array dimensions in C ordering. In Fortran, the order is reversed (e.g. arrays are dimensioned (grid_corners,grid_size). In the grids subdirectory of the distribution there are some fortran source codes for creating these grid files for some special cases. See the README file in that subdirectory for details.

The name of the grid is given as the title and will be used to specify the grid name throughout the remapping process.

The grid_size dimension is the total size of the grid; grid_rank refers to the number of dimensions the grid array would have when used in a model code. The number of corners (vertices) in each grid cell is given by grid_corners. Note that if your grid has a variable number of corners on grid cells, then you should set grid_corners to be the highest value and use redundant points on cells with fewer corners.

The integer array grid_dims gives the length of each grid axis when used in a model code. Because the remapping routines read the grid properties as a linear list of grid cells, the grid_dims array is necessary for reconstructing the grid, particularly for a bilinear mapping where a logically rectangular structure is needed.

Figure 2.2: A sample input grid file.

```
netcdf remap_grid_T42 {
dimensions:
        grid_size = 8192 ;
        grid_corners = 4 ;
        grid_rank = 2 ;

variables:
        long grid_dims(grid_rank) ;
        double grid_center_lat(grid_size) ;
                grid_center_lat:units = "radians" ;
        double grid_center_lon(grid_size) ;
                grid_center_lon:units = "radians" ;
        long grid_imask(grid_size) ;
                grid_imask:units = "unitless" ;
        double grid_corner_lat(grid_size, grid_corners) ;
                grid_corner_lat:units = "radians" ;
        double grid_corner_lon(grid_size, grid_corners) ;
                grid_corner_lon:units = "radians" ;

// global attributes:
                :title = "T42 Gaussian Grid" ;
}
```

The integer array grid_imask is used to mask out grid cells which should not participate in the remapping. The array should by zero for any points (e.g. land points) that do not participate in the remapping and one for all other points.

Coordinate arrays provide the latitudes and longitudes of cell centers and cell corners. Although the above reports the units as "radians", the code happily accepts "degrees" as well. The grid corner coordinates *must* be written in an order which traces the outside of a grid cell in a counterclockwise sense. That is, when moving from corner 1 to corner 2 to corner 3, etc., the grid cell interior must always be to the left.

2.2.3 Output Files

The remapping output files are also in netCDF format and contain some grid information from each grid as well as the remapping addresses and weights. An example ncdump of the output files is shown in Fig. 2.3.

The grid information is simply echoing the input grid file information and adding grid_area and grid_frac arrays. The grid_area array currently is *only* computed by the conservative remapping option; the others will write arrays full of zeros for this field. The grid_frac array for the conservative remapping returns the area fraction of the grid cell which participates in the remapping based on the source grid mask. For the other two remapping options, the grid_frac array is one where the grid point participates in the remapping and zero otherwise, based again on the source grid mask (and *not* on the grid_imask for that grid).

The remapping data itself is written as if for a sparse matrix multiplication. Again, the Fortran array must be dimensioned (num_wgts,num_links) rather than the C order shown in the ncdump. The dimension num_wgts refers to the number of weights for a given remapping and is one for bilinear and distance-weighted remappings. For the conservative remapping, num_wgts is 3 as it contains two additional weights for a second-order remapping. The bicubic remappings require four weights as for gradients in each direction plus a term for the cross gradient. The dimension num_links is the number of unique address pairs in the remapping and is therefore the number of entries in a sparse matrix for the remapping. The integer address arrays contain the source and destination address for each "link". So, a Fortran code to complete the conservative remappings might look like that shown in Fig. 2.4.

Figure 2.3: A sample output file for mapping data in scrip format.

```
netcdf rmp_POP43_to_T42_cnsrv {
dimensions:
        src_grid_size = 24576 ; dst_grid_size = 8192 ;
        src_grid_corners = 4  ; dst_grid_corners = 4 ;
        src_grid_rank = 2     ; dst_grid_rank = 2 ;
        num_links = 42461 ; num_wgts = 3 ;
variables:
        long src_grid_dims(src_grid_rank) ;
        long dst_grid_dims(dst_grid_rank) ;
        double src_grid_center_lat(src_grid_size) ;
        double dst_grid_center_lat(dst_grid_size) ;
        double src_grid_center_lon(src_grid_size) ;
        double dst_grid_center_lon(dst_grid_size) ;
        long src_grid_imask(src_grid_size) ;
        long dst_grid_imask(dst_grid_size) ;
        double src_grid_corner_lat(src_grid_size, src_grid_corners) ;
        double src_grid_corner_lon(src_grid_size, src_grid_corners) ;
        double dst_grid_corner_lat(dst_grid_size, dst_grid_corners) ;
        double dst_grid_corner_lon(dst_grid_size, dst_grid_corners) ;
        double src_grid_area(src_grid_size) ;
                src_grid_area:units = "square radians" ;
        double dst_grid_area(dst_grid_size) ;
                dst_grid_area:units = "square radians" ;
        double src_grid_frac(src_grid_size) ;
        double dst_grid_frac(dst_grid_size) ;
        long src_address(num_links) ;
        long dst_address(num_links) ;
        double remap_matrix(num_links, num_wgts) ;
// global attributes:
         :title = "POP 4/3 to T42 Conservative Mapping" ;
         :normalization = "fracarea" ;
         :map_method = "Conservative remapping" ;
         :history = "Created: 07-19-1999" ;
         :conventions = "SCRIP" ;
         :source_grid = "POP 4/3 Displaced-Pole T grid" ;
         :dest_grid = "T42 Gaussian Grid" ;
}
```

Figure 2.4: Sample Fortran code for performing a first-order conservative remap.

```
dst_array = 0.0

select case (normalize_opt)
case ('fracarea')

  do n=1,num_links
    dst_array(dst_address(n)) = dst_array(dst_address(n)) +
              remap_matrix(1,n)*src_array(src_address(n))
  end do

case ('destarea')

  do n=1,num_links
    dst_array(dst_address(n)) = dst_array(dst_address(n)) +
             (remap_matrix(1,n)*src_array(src_address(n)))/
             (dst_frac(dst_address(n)))
  end do

case ('none')

  do n=1,num_links
    dst_array(dst_address(n)) = dst_array(dst_address(n)) +
             (remap_matrix(1,n)*src_array(src_address(n)))/
       (dst_area(dst_address(n))*dst_frac(dst_address(n)))
  end do

end select
```

The address arrays are sorted by destination address and are linear addresses that assume standard Fortran ordering. They can therefore be converted to logical address space if necessary. For example, a point on a two-dimensional grid with logical coordinates (i, j) will have a linear address n given by $n = (j-1)*\mathrm{grid\_dims}(1)+i$. Alternatively, if the code is run on a serial machine, the multi-dimensional arrays can be passed into linear dummy arrays and the addresses can be used directly. Such a storage/sequence association may not be valid in a distributed-memory context however. The scrip_test code shows an example of how the remappings can be implemented.

2.3 Testing

In order to test the weights computed by the SCRIP package, a simple test code is provided. This code reads in the weights and remaps analytic fields. Three choices for the analytic field are provided. The first is a cosine bell function $f = 2+\cos(\pi r/L)$, where r is the distance from the center of the hill and L is a length scale. Such a function is useful for determining the effects of repeated applications. The other two fields are representative of spherical harmonic wavefunctions. A relatively smooth function $f = 2 + \cos^2\theta\cos(2\phi)$ is similar to a spherical harmonic with $\ell = 2$ and $m = 2$, where ℓ is the spherical harmonic order and m is the azimuthal wave number. The function $f = 2 + \sin^{16}(2\theta)\cos(16\phi)$ is similar to a spherical harmonic with $\ell = 32$ and $m = 16$ and is useful for testing a field with relatively high spatial frequency and rapidly changing gradients. The choice of which field is remapped in determined by the input namelist scrip_test_in.

For conservative remappings, the test code tests three different remappings: the first is a first-order remapping, the second is a second-order remapping using only latitude gradients, and the third is a full second-order remapping. The second is performed in order to determine which weights are causing problems when errors occur. The code prints out three diagnostics to standard output and writes many quantities to a netCDF output file.

First, it prints out the minimum and maximum of the source and destination (remapped) fields. This is a test for monotonicity (although only the first-order conservative remapping is monotone by default).

Second, the test code prints out the maximum and average relative error $\epsilon = |(F_{dst}-F_{analytic})/F_{analytic}|$, where $F_{analytic}$ is the source function evaluated at the destination grid points and F_{dst} is the destination (remapped) field.

The errors here can sometimes be misleading. For example, if a conservative remapping is performed from a fine grid to a coarse grid, the destination array will contain the field averaged over many source cells, while $F_{analytic}$ is the analytic field evaluated at the cell center point. Another instance which leads to relatively large errors is near mask boundaries where the remapped field is correctly returning values indicative of the edge of a grid cell, while $F_{analytic}$ is again computing cell-center values. To avoid the latter problem, the error is only computed where the destination grid fraction is greater than 0.999.

Lastly, the test code prints out the area-integrated field on the source and destination grids in order to test conservation. This diagnostic returns zeros for all but conservative remappings. For a first-order conservative remapping, these numbers should agree to machine accuracy. For a second-order conservative remapping, they will be very close, but may not exactly agree due to mask boundary effects where it is not possible to perform the exact area integral.

The netCDF output file from the test code contains the source and destination arrays as well as the error arrays so the error can be examined at every grid point to pinpoint problems. The arrays in the netCDF file are written out in arrays with rank grid_rank (e.g. two-dimensional grids are written as proper 2-d arrays rather than vectors of values). These arrays can then be viewed using any visualization package.

Chapter 3

Conservative Remapping

The SCRIP package implements a conservative remapping scheme described in detail in a separate paper (Jones, P.W. 1999 *Monthly Weather Review*, **127**, 2204-2210). A brief outline will be given here to aid the user in understanding what this portion of the package does.

To compute a flux on a new (destination) grid which results in the same energy or water exchange as a flux f on an old (source) grid, the destination flux F at a destination grid cell k must satisfy

$$\overline{F}_k = \frac{1}{A_k} \int\int_{A_k} f dA, \tag{3.1}$$

where $\overline{F}$ is the area-averaged flux and A_k is the area of cell k. Because the integral in (3.1) is over the area of the destination grid cell, only those cells on the source grid that are covered at least partly by the destination grid cell contribute to the value of the flux on the destination grid. If cell k overlaps N cells on the source grid, the remapping can be written as

$$\overline{F}_k = \frac{1}{A_k} \sum_{n=1}^{N} \int\int_{A_{nk}} f_n dA, \tag{3.2}$$

where A_{nk} is the area of the source grid cell n covered by the destination grid cell k, and f_n is the local value of the flux in the source grid cell (see Figure 3.1). Note that (3.2) is normalized by the destination area A_k corresponding to the normalize_opt value of 'destarea'. The sum of the weights for a destination cell k in this case would be between 0 and 1 and would be the area fraction if f_n were identically 1 everywhere on the source grid.

The normalization option 'fracarea' would actually divide by the area of the source grid overlapped by cell k:

$$\sum_{n=1}^{N} \int\int_{A_{nk}} dA. \tag{3.3}$$

For this normalization option, remapping a function f which is 1 everywhere on the source grid would result in a function F that is exactly one wherever the destination grid overlaps a non-masked source grid cell and zero otherwise. A normalization option of 'none' would result in the actual angular area participating in the remapping.

Assuming f_n is constant across a source grid cell, (3.2) would lead to the first-order area-weighted schemes used in current coupled models. A more accurate form of the remapping is obtained by using

$$f_n = \overline{f}_n + \nabla_n f \cdot (\vec{r} - \vec{r}_n), \tag{3.4}$$

where $\nabla_n f$ is the gradient of the flux in cell n and $\vec{r}_n$ is the centroid of cell n defined by

$$\vec{r}_n = \frac{1}{A_n} \int\int_{A_n} \vec{r} dA. \tag{3.5}$$

Such a distribution satisfies the conservation constraint and is equivalent to the first terms of a Taylor series expansion of f around $\vec{r}_n$. The remapping is thus second-order accurate if $\nabla_n f$ is at least a first-order approximation to the gradient.

The remapping can now be expanded in spherical coordinates as

$$\overline{F}_k = \sum_{n=1}^{N} \left[\overline{f}_n w_{1nk} + \left(\frac{\partial f}{\partial \theta}\right)_n w_{2nk} + \left(\frac{1}{\cos\theta} \frac{\partial f}{\partial \phi}\right)_n w_{3nk} \right], \tag{3.6}$$

where θ is latitude, ϕ is longitude and the three remapping weights are

$$w_{1nk} = \frac{1}{A_k} \int\int_{A_{nk}} dA, \tag{3.7}$$

$$\begin{aligned} w_{2nk} &= \frac{1}{A_k} \int\int_{A_{nk}} (\theta - \theta_n) dA \\ &= \frac{1}{A_k} \int\int_{A_{nk}} \theta dA - \frac{w_{1nk}}{A_n} \int\int_{A_n} \theta dA, \end{aligned} \tag{3.8}$$

and

$$\begin{aligned} w_{3nk} &= \frac{1}{A_k}\int\int_{A_{nk}} \cos\theta(\phi-\phi_n)dA \\ &= \frac{1}{A_k}\int\int_{A_{nk}} \phi\cos\theta dA - \frac{w_{1nk}}{A_n}\int\int_{A_n} \phi\cos\theta dA. \end{aligned} \tag{3.9}$$

Again, if the gradient is zero, (3.6) reduces to a first-order area-weighted remapping.

The area integrals in equations (3.7)–(3.9) are computed by converting the area integrals into line integrals using the divergence theorem. Computing line integrals around the overlap regions is much simpler; one simply integrates first around every grid cell on the source grid, keeping track of intersections with destination grid lines, and then one integrates around every grid cell on the destination grid in a similar manner. After the sweep of each grid, all overlap regions have been integrated.

Choosing appropriate functions for the divergence, the integrals in equations (3.7)–(3.9) become

$$\int\int_{A_{nk}} dA = \oint_{C_{nk}} -\sin\theta d\phi, \tag{3.10}$$

$$\int\int_{A_{nk}} \theta dA = \oint_{C_{nk}} [-\cos\theta - \theta\sin\theta]d\phi, \tag{3.11}$$

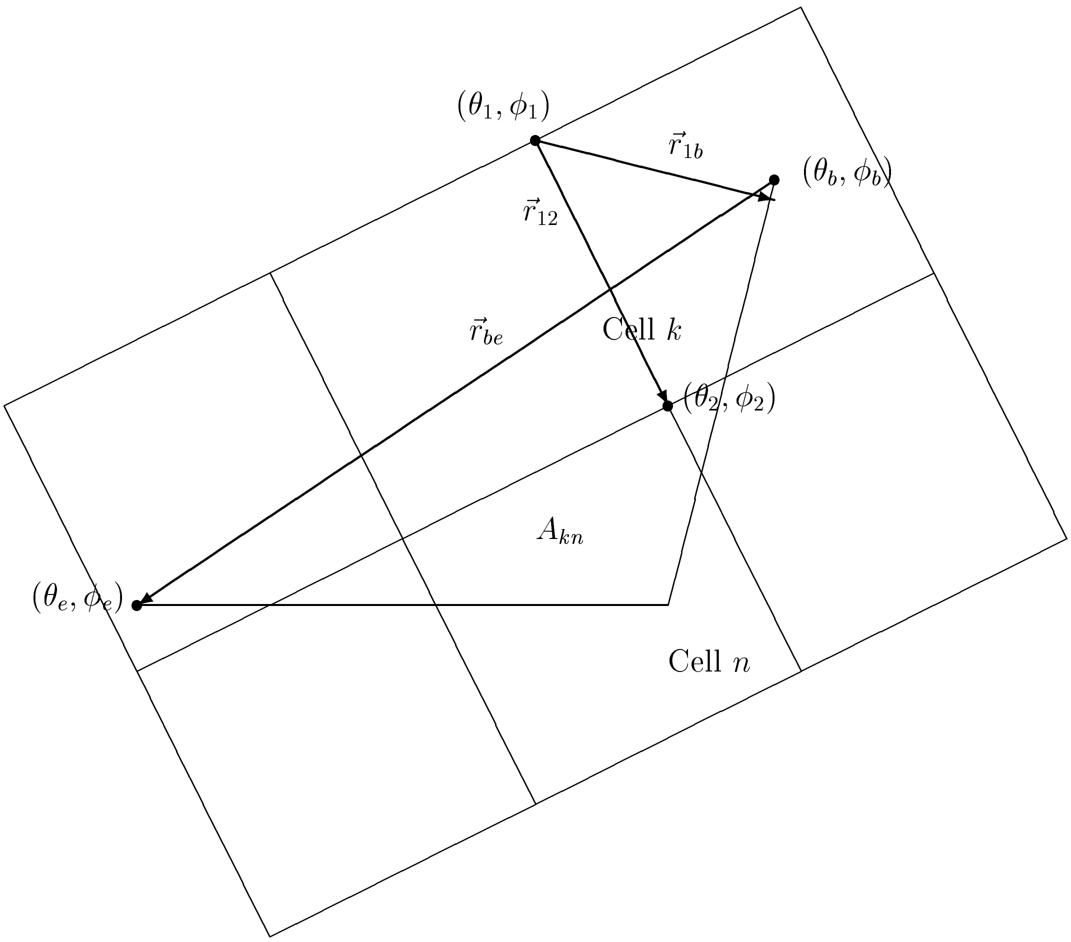
$$\int\int_{A_{nk}} \phi\cos\theta dA = \oint_{C_{nk}} -\frac{\phi}{2}[\sin\theta\cos\theta + \theta]d\phi, \tag{3.12}$$

where C_{nk} is the counterclockwise path around the region A_{nk}. Computing these three line integrals during the sweeps of each grid provides all the information necessary for computing the remapping weights.

3.1 Search algorithms

As mentioned in the previous section, the algorithm for computing the remapping weights is relatively simple. The process amounts to finding the location of the endpoint of a segment and then finding the next intersection with the other grid. The line integrals are then computed and summed according to which grid cells are associated with that particular subsegment. The most time-consuming portion of the algorithm is finding which cell on one grid

Figure 3.1: An example of a triangular destination grid cell k overlapping a quadrilateral source grid. The region A_{kn} is where cell k overlaps the quadrilateral cell n. Vectors used by search and intersection routines are also labelled.



contains an endpoint from the other grid. Optimal search algorithms can be written when the grid is well structured and regular. However, if one requires a search algorithm that will work for any general grid, a hierarchy of search algorithms appears to work best. In SCRIP, each grid cell address is assigned to one or more latitude bins. When the search begins, only those cells belonging to the same latitude bin as the search point are used. The second stage checks the bounding box of each grid cell in the latitude bin. The bounding box is formed by the cells minimum and maximum latitude and longitude. This process further restricts the search to a small number of cells.

Once the search has been restricted, a robust algorithm that works for most cases is a cross-product test. In this test, a cross product is computed between the vector corresponding to a cell side ($\vec{r}_{12}$ in Figure 3.1) and a vector extending from the beginning of the cell side to the search point ($\vec{r}_{1b}$). If

$$\vec{r}_{12} \times \vec{r}_{1b} > 0, \tag{3.13}$$

the point lies to the left of the cell side. If (3.13) holds for every cell side, the point is enclosed by the cell. This test is not completely robust and will fail for grid cells that are non-convex.

3.2 Intersections

Once the location of an initial endpoint is found, it is necessary to check to see if the segment intersects with the cell side. If the segment is parametrized as

$$\begin{aligned} \theta &= \theta_b + s_1(\theta_e - \theta_b) \\ \phi &= \phi_b + s_1(\phi_e - \phi_b) \end{aligned} \tag{3.14}$$

and the cell side as

$$\begin{aligned} \theta &= \theta_1 + s_2(\theta_2 - \theta_1) \\ \phi &= \phi_1 + s_2(\phi_2 - \phi_1), \end{aligned} \tag{3.15}$$

where $\theta_1, \phi_1, \theta_2, \phi_2, \theta_b$, and θ_e are endpoints as shown in Figure 3.1, the intersection of the two lines occurs when θ and ϕ are equal. The linear system

$$\begin{bmatrix} (\theta_e - \theta_b) & (\theta_1 - \theta_2) \\ (\phi_e - \phi_b) & (\phi_1 - \phi_2) \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} (\theta_1 - \theta_b) \\ (\phi_1 - \phi_b) \end{bmatrix} \tag{3.16}$$

is then solved to determine s_1 and s_2 at the intersection point. If s_1 and s_2 are between zero and one, an intersection occurs with that cell side.

It is important also to compute identical intersections during the sweeps of each grid. To ensure that this will occur, the entire line segment is used to compute intersections rather than using a previous or next intersection as an endpoint.

3.3 Coincidences

Often, pairs of grids will share common lines (e.g. the Equator). When this is the case, the method described above will double-count the contribution of these line segments. Coincidences can be detected when computing cross products for the search algorithm described above. If the cross product is zero in this case, the endpoint lies on the cell side. A second cross product between the line segment and the cell side can then be computed. If the second cross product is also zero, the lines are coincident. Once a coincidence has been detected, the contribution of the coincident segment can be computed during the first sweep and ignored during the second sweep.

3.4 Spherical coordinates

Some aspects of the spherical coordinate system introduce additional problems for the method described above. Longitude is multiple valued on one line on the sphere, and this branch cut may be chosen differently by different grids. Care must be taken when calculating intersections and line integrals to ensure that the proper longitude values are used. A simple method is to always check to make sure the longitude is in the same interval as the source grid cell center.

Another problem with computing weights in spherical coordinates is the treatment of the pole. First, note that although the pole is physically a point, it is a line in latitude-longitude space and has a nonzero contribution to the weight integrals. If a grid does not contain the pole explicitly as a grid vertex, the pole contribution must be added to the appropriate cells. The pole contribution can be computed analytically.

The pole also creates problems for the search and intersection algorithms described above. For example, a grid cell that overlaps the pole can result in a nonconvex cell in latitude-longitude coordinates. The cross-product test described above will fail in this case. In addition, segments near the pole typically exhibit large changes in longitude even for very short segments. In such a case, the linear parametrizations used above result in inaccuracies for determining the correct intersections.

To avoid these problems, a coordinate transformation can be used pole-ward of a given threshold latitude (typically within one degree of the pole). A possible transformation is the Lambert equivalent azimuthal projection

$$\begin{aligned} X &= 2\sin\left(\frac{\pi}{4}-\frac{\theta}{2}\right)\cos\phi \\ Y &= 2\sin\left(\frac{\pi}{4}-\frac{\theta}{2}\right)\sin\phi \qquad (3.17) \end{aligned}$$

for the North Pole. The transformation for the South Pole is similar. This transformation is only used to compute intersections; line integrals are still computed in latitude-longitude coordinates. Because intersections computed in the transformed coordinates can be different from those computed in latitude-longitude coordinates, line segments which cross the latitude threshold must be treated carefully. To compute the intersections consistently for such a segment, intersections with the threshold latitude are detected and used as a normal grid intersection to provide a clean break between the two coordinate systems.

3.5 Conclusion

The implementation in the SCRIP code follows closely the description above. The user should be able to follow and understand the process based on this description.

Chapter 4

Bilinear Remapping

Standard bilinear interpolation schemes can be found in many textbooks. Here we present a more general scheme which uses a local bilinear approximation to interpolate to a point in a quadrilateral grid. Consider the grid points shown in Fig. 4.1 labelled with logically-rectangular indices (e.g. (i, j)).

Let the latitude-longitude coordinates of point 1 be $(\theta(i, j), \phi(i, j))$, the coordinates of point 2 be $(\theta(i + 1, j), \phi(i + 1, j))$, etc. Now let α and β be continuous local coordinates such that the coordinates (α, β) of point 1 are $(0, 0)$, point 2 are $(1, 0)$, point 3 are $(1, 1)$ and point 4 are $(0, 1)$. If point P lies inside the cell formed by the four points above, the function f at point P can be approximated by

$$\begin{aligned} f_P &= (1-\alpha)(1-\beta)f(i,j) + \alpha(1-\beta)f(i+1,j) + \\ &\quad \alpha\beta f(i+1,j+1) + (1-\alpha)\beta f(i,j+1) \qquad (4.1) \\ &= w_1 f(i,j) + w_2 f(i+1,j) + w_3 f(i+1,j+1) + w_4 f(i,j+1). \end{aligned}$$

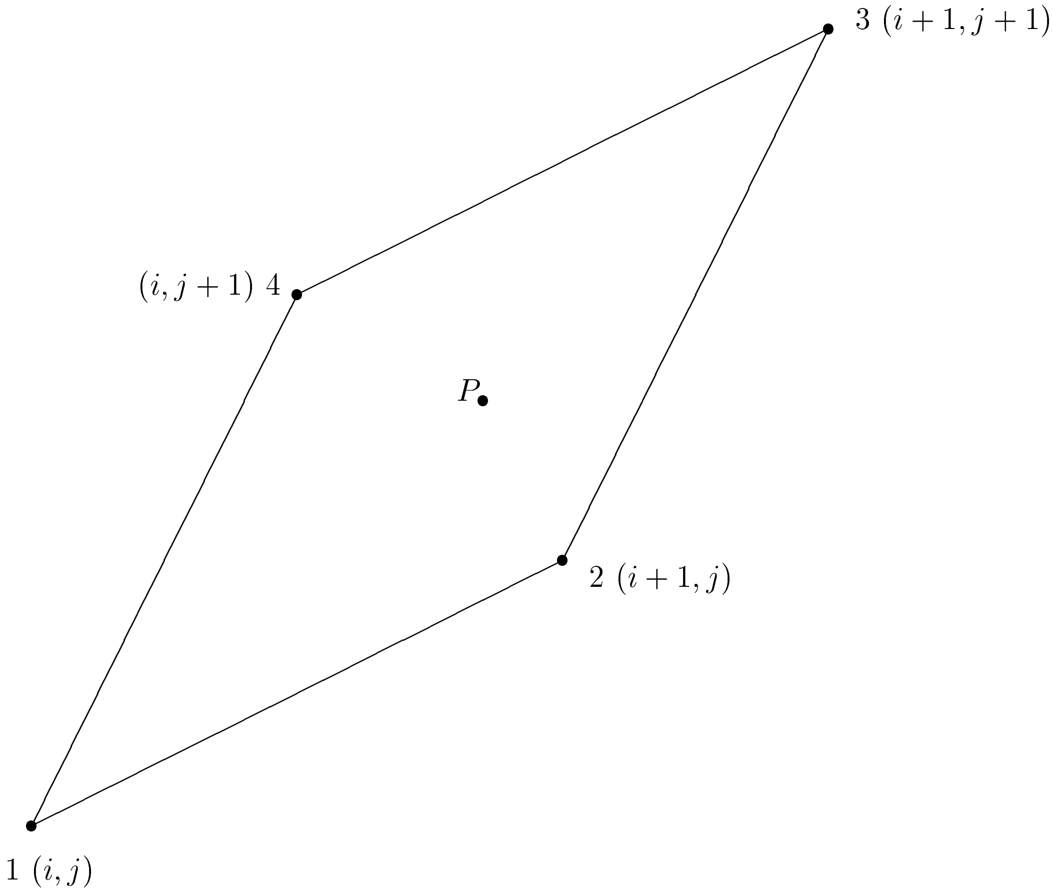
The remapping weights must therefore be computed by finding α and β at point P.

The latitude-longitude coordinates (θ, ϕ) of point P are known and can also be approximated by

$$\begin{aligned} \theta &= (1-\alpha)(1-\beta)\theta_1 + \alpha(1-\beta)\theta_2 + \alpha\beta\theta_3 + (1-\alpha)\beta\theta_4 \\ \phi &= (1-\alpha)(1-\beta)\phi_1 + \alpha(1-\beta)\phi_2 + \alpha\beta\phi_3 + (1-\alpha)\beta\phi_4. \qquad (4.2) \end{aligned}$$

Because (4.2) is nonlinear in α and β, we must linearize and iterate toward

Figure 4.1: A general quadrilateral grid.



a solution. Differentiating (4.2) results in

$$\begin{bmatrix} \delta\theta \\ \delta\phi \end{bmatrix} = A \begin{bmatrix} \delta\alpha \\ \delta\beta \end{bmatrix}, \tag{4.3}$$

where

$$A = \begin{bmatrix} (\theta_2 - \theta_1) + (\theta_1 - \theta_4 + \theta_3 - \theta_2)\beta & (\theta_4 - \theta_1) + (\theta_1 - \theta_4 + \theta_3 - \theta_2)\alpha \\ (\phi_2 - \phi_1) + (\phi_1 - \phi_4 + \phi_3 - \phi_2)\beta & (\phi_4 - \phi_1) + (\phi_1 - \phi_4 + \phi_3 - \phi_2)\alpha \end{bmatrix}. \tag{4.4}$$

Inverting this system,

$$\delta\alpha = \begin{vmatrix} \delta\theta & (\theta_4 - \theta_1) + (\theta_1 - \theta_4 + \theta_3 - \theta_2)\alpha \\ \delta\phi & (\phi_4 - \phi_1) + (\phi_1 - \phi_4 + \phi_3 - \phi_2)\alpha \end{vmatrix} \div \det(A), \tag{4.5}$$

and

$$\delta\beta = \begin{vmatrix} (\theta_2 - \theta_1) + (\theta_1 - \theta_4 + \theta_3 - \theta_2)\beta & \delta\theta \\ (\phi_2 - \phi_1) + (\phi_1 - \phi_4 + \phi_3 - \phi_2)\beta & \delta\phi \end{vmatrix} \div \det(A). \tag{4.6}$$

Starting with an initial guess for α and β (say $\alpha = \beta = 0$), equations (4.5) and (4.6) can be iterated until $\delta\alpha$ and $\delta\beta$ are suitably small. The weights can then be computed from (4.1). Note that for simple latitude-longitude grids, this iteration will converge in the first iteration.

In order to compute the weights using this general bilinear iteration, it must be determined in which box the point P resides. For this, the search algorithms outlined in the previous chapter are used with the exception that instead of using cell corners, the relevant box is formed by neighbor cell centers as shown in Fig. 4.1.

Chapter 5

Bicubic Remapping

The bicubic remapping exactly follows the bilinear remapping except that four weights for each corner point are required. Thus, num_wts is set to four for this option. The bicubic remapping is

$$
\begin{aligned}
f_P \quad = \quad & (1-\beta^2(3-2\beta))(1-\alpha^2(3-2\alpha))f(i,j) + \\
& (1-\beta^2(3-2\beta))\alpha^2(3-2\alpha)f(i+1,j) + \\
& \beta^2(3-2\beta)\alpha^2(3-2\alpha)f(i+1,j+1) + \\
& \beta^2(3-2\beta)(1-\alpha^2(3-2\alpha))f(i,j+1) + \\
& (1-\beta^2(3-2\beta))\alpha(\alpha-1)^2\frac{\partial f}{\partial i}(i,j) + \\
& (1-\beta^2(3-2\beta))\alpha^2(\alpha-1)\frac{\partial f}{\partial i}(i+1,j) + \\
& \beta^2(3-2\beta)\alpha^2(\alpha-1)\frac{\partial f}{\partial i}(i+1,j+1) + \\
& \beta^2(3-2\beta)\alpha(\alpha-1)^2\frac{\partial f}{\partial i}(i,j+1) + \\
& \beta(\beta-1)^2(1-\alpha^2(3-2\alpha))\frac{\partial f}{\partial j}(i,j) + \\
& \beta(\beta-1)^2\alpha^2(3-2\alpha)\frac{\partial f}{\partial j}(i+1,j) + \\
& \beta^2(\beta-1)\alpha^2(3-2\alpha)\frac{\partial f}{\partial j}(i+1,j+1) + \\
& \beta^2(\beta-1)(1-\alpha^2(3-2\alpha))\frac{\partial f}{\partial j}(i,j+1) +
\end{aligned}
$$

$$\alpha(\alpha-1)^2\beta(\beta-1)^2\frac{\partial^2 f}{\partial i\partial j}(i,j)+$$
$$\alpha^2(\alpha-1)\beta(\beta-1)^2\frac{\partial^2 f}{\partial i\partial j}(i+1,j)+$$
$$\alpha^2(\alpha-1)\beta^2(\beta-1)\frac{\partial^2 f}{\partial i\partial j}(i+1,j+1)+$$
$$\alpha(\alpha-1)^2\beta^2(\beta-1)\frac{\partial^2 f}{\partial i\partial j}(i,j+1) \tag{5.1}$$

where α and β are identical to those found in the bilinear case and are found using an identical algorithm. Note that unlike the conservative remappings, the gradients here are gradients with respect to the *logical* variable and not latitude or longitude. Lastly, the four weights corresponding to each address pair correspond to the weight multiplying the field value at the point, the weight multiplying the gradient with respect to i, the weight multiplying the gradient with respect to j, and the weight multiplying the cross gradient in that order.

Chapter 6

Distance-weighted Average Remapping

This scheme for remapping is probably the simplest in this package. The code simply searches for the num_neighbors nearest neighbors and computes the weights using

$$w = \frac{1/(d+\epsilon)}{\sum_n^{\mathrm{num\_neighbors}}[1/(d_n+\epsilon)]}, \tag{6.1}$$

where ϵ is a small number to prevent dividing by zero, the sum is for normalization and d is the distance from the destination grid point to the source grid point. The distance is the angular distance

$$d = \cos^{-1}\left(\cos\theta_d\cos\theta_s(\cos\phi_d\cos\phi_s + \sin\phi_d\sin\phi_s) + \sin\theta_d\sin\theta_s\right), \tag{6.2}$$

where θ is latitude, ϕ is longitude and the subscripts d, s denote destination and source grids, respectively.

When finding nearest neighbors, the distance is not computed between the destination grid point and every source grid point. Instead, the search is narrowed by sorting the two grids into latitude bins. Only those source grid cells lying in the same latitude bin as the destination grid cell are checked to find the nearest neighbors.

Chapter 7

Bugs

A file called 'bugs' is included in the distribution which lists current outstanding bugs as well as a version history. Any further bugs, comments, or suggestions should be sent to me at pwjones@lanl.gov.

The code does not have very useful error messages to help diagnose problems so feel free to pester me with any problems you encounter.

The package has also not been extensively tested on a variety of machines. It works fine on SGI machines and IBM machines, but has not been run on other machines. It is pretty vanilla Fortran, so should be ok on machines with standard-compliant F90 compilers.