ocean_grid_generator_guide

March 5, 2019

```
In [42]: %%javascript
         MathJax.Hub.Config({
             TeX: { equationNumbers: { autoNumber: "AMS" } }
         });

         MathJax.Hub.Queue(
           ["resetEquationNumbers", MathJax.InputJax.TeX],
           ["PreProcess", MathJax.Hub],
           ["Reprocess", MathJax.Hub]
         );

<IPython.core.display.Javascript object>
```

1 Grids and Coordinates

A (computational) grid in ESMs is a specification for indexing a selected set of (grid) points on the Earth. In GFDL numerical ocean models, the horizontal ocean grid is a set of 2D arrays $\{x_{ji}, y_{ji}\}$ where each index pair ij specifies a logically rectangular "grid cell" centered at longitude and lattitude coordinates (x_{ji}, y_{ji}) (in degrees). This set is augmented by the corresponding finite-element "metrics" of the grid cells $\{dx_{ji}, dy_{ji}, da_{ji}, angle_{ji}\}$ corresponding to the lengths in x and y directions (meters), the cell area (meters squared) and the angle that the dx makes with the lattitude circle at that point.

The goal of this tool set is to generate such a grid that covers the world oceans for a given nominal resolution e.g., $\delta = x_{10} - x_{00}$ at the equator.

For a properly small region of the Earth (hence flat), this could easily be accomplished by a regular, rectangular coordinate patch, namely $x_{ji} = i\delta, y_{ji} = j\delta, dx = dy = \delta, da = \delta^2, angle = 0$. For a global curved surface like that of Earth however it is topologically impossible to come up with a single, everywhere non-singular coordinate patch that covers the whole surface. We must hence resort to using multiple (at least 2) coordinate patches for the globe. The choice of the number and shape of these patches usually come from the restrictions of the numerical model that is using the grid (e.g., removing the singularity of the lat-lon grid at the northern polar ocean), or the choice of the investigators that want to put more resolutions in some parts of the model.

The preferred horizontal grid in the GFDL Modular Ocean Models (MOM6) is composed of 4 coordinate patches (stiched together) as follows: 1. A Mercator projection grid spanning the equator 2. A Murray northern bipolar cap 3. A regular spherical (lat-lon) grid spanning the southern ocean 4. A regular spherical sothern cap with an optional displaced south pole

2 Constructing a Grid on Sphere by Projection

A general mathematical remark is helpful here. We can construct a grid on a sphere via a number of projections. A projection is an invertible smooth map P from the surface of the sphere onto a subset T of the Euclidian plane and can be expressed by

$$(\xi_1, \xi_2) = P(\lambda, \phi) \tag{1}$$

Now, if we can make a coordinate patch for the target set T such that all image points for our grid have integer coordinates $(\xi_1, \xi_2) = (i, j)$ then we can apply the inverse map P^{-1} to this set (reproject) to obtain a grid of $(\lambda_{ji}, \phi_{ji})$ on the sphere.

Among the simplest of these projections are the "spherical projections" $(\xi_1, \xi_2) = (R\lambda, R\phi)$ and $(\xi_1, \xi_2) = (R\lambda, R\sin(\phi))$ which give rise to the familiar regular lat-lon grids on the sphere. The Mercator projection described below is also a "spherical projection" with some additional features that are helpful to oceanographers.

3 Mercator Projection

For a given point on a sphere of radius R with $(longitude, latitude) = (\lambda, \phi)$ the (isotropic) Mercator projection is defined by:

$$\xi_2 = M(\phi) = R \ln \left[\frac{1 + \sin\phi}{\cos\phi} \right]. \tag{2}$$

The corresponding inverse function can be written as:

$$\phi = M^{-1}(\xi_2) = \tan^{-1}\left(\sinh(\xi_2/R)\right). \tag{3}$$

Here R is the radius of the "globe" representing the Earth and is determined by the scale of the projection. E.g., to project the Earth surface to a rectangular map of width L one can use a "principal scale" of $L/(2\pi a_E)$ (a_E) being radius of the isotropic Earth), which corresponds to a "globe" of radius $R = L/2\pi$. Note that this function does not describe the geometrical projection (as of light rays onto a screen) from the centre of the globe to the cylinder, which is only one of an unlimited number of ways to conceptually project a cylindrical map.

These are one of a few equivalent analytic forms of the projection function. For an overview and more information refer to the wikipedia article on Mercator Projection.

3.1 Mercator Grid Specification

Suppose we want to generate an isotropic Mercator grid corresponding to a nominal resolution of $\Delta\lambda_n$ at the equator (e.g., 1/4 degree). Here are the defining parameters of our Mercator super-grid (a grid with twice the desired resolution that is used in ocean models):

0. Like other projection constructions, the supergrid should come from the re-projection of a planar grid patch of integer coordinates.
1. The super-grid cell resolution in longitude direction should be $\Delta\lambda = \Delta\lambda_n/2$ (e.g., 1/8).

2. The super-grid cell resolution in latitude direction should be the same as in longitude direction at the equator $\Delta\phi(\phi = 0) = \Delta\lambda$.

3. With the resolution being fixed by the above two, we should specify the nominal extent of the grid to be from $\tilde{\Phi}_s$ to $\tilde{\Phi}_n$.

To achieve these we must:

0. set $(\xi_1, \xi_2) = (i, j)$ and hence $\Delta\xi_1 = \Delta\xi_2 = 1$

1. partition the equator to N_i intervals such that $\Delta\lambda = \frac{360}{N_i}$ (e.g., 2880),

$$N_i = 2\frac{360}{\Delta\lambda_n}. \tag{4}$$

2. calculate Δy given $\Delta\phi$ using equation (3):

$$\Delta\phi = \Delta\xi_2(\frac{d\phi}{d\xi_2})|_{\xi_2=0} = \Delta\xi_2 \frac{1}{R}\frac{\cosh(\xi_2/R)}{1+\sinh^2(\xi_2/R)}|_{\xi_2=0} = \frac{\Delta\xi_2}{R}$$

The above conditions fix the value of the projection scale R as follows:

$$R = \frac{\Delta\xi_2}{\Delta\phi}|_{\xi_2=0} = \frac{\Delta\xi_2}{\Delta\lambda}|_{\xi_2=0} = \frac{N_i}{2\pi} \tag{5}$$

This result holds for any "spherical projection", in fact this is my definition of "spherical projection".

3. Generally the nominal extents $\tilde{\Phi}_s$ and $\tilde{\Phi}_n$ do not produce integer values for ξ_2 from equation (2). Instead, we shall calculate the nearest integer values $j_s = int(M(\tilde{\Phi}_s))$ and $j_n = int(M(\tilde{\Phi}_n))$ and use them as the grid extents in the j-direction. The actual lattitudinal extents of the grid are then determined from (3) to be $M^{-1}(j_s)$ and $M^{-1}(j_n)$. In our tool we also shift the grid indices such that $j_s = 0$.

4 Bipolar Northern Cap Projection

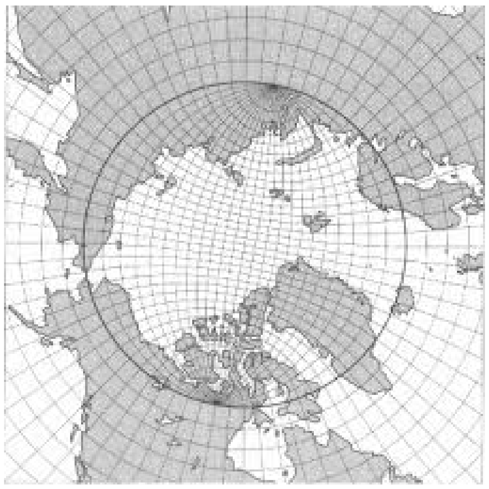
Here we construct the bipolar northern cap grid using a geometric method different from but inspired by Murray's 1996 paper [1].

At the heart of the algorithm are again a projection from sphere onto Euclidian plane, coordinate the image by integers and finally reproject back to the sphere. In the present case the useful projection is streographic.

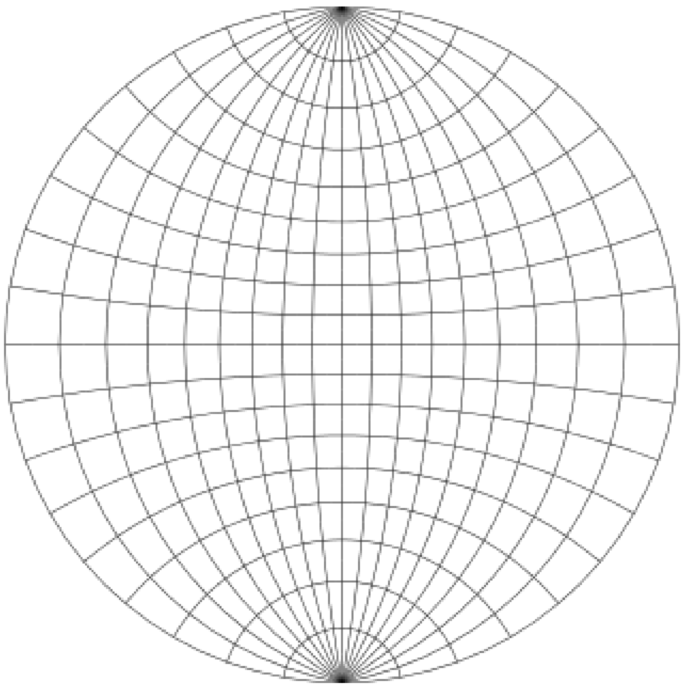
It is helpful to first look at the end product to get a geometrical intuition. We want to end up with a coordinate system for the northern ocean that is not "singular" at the North pole. Figure below shows one such grid consructed in Murray's 1996 paper. It has two "poles" but they are moved away from the ocean and located on land.

There are a few ways to construct such a grid, here we describe one (very similar to the idea of a Wulf net).

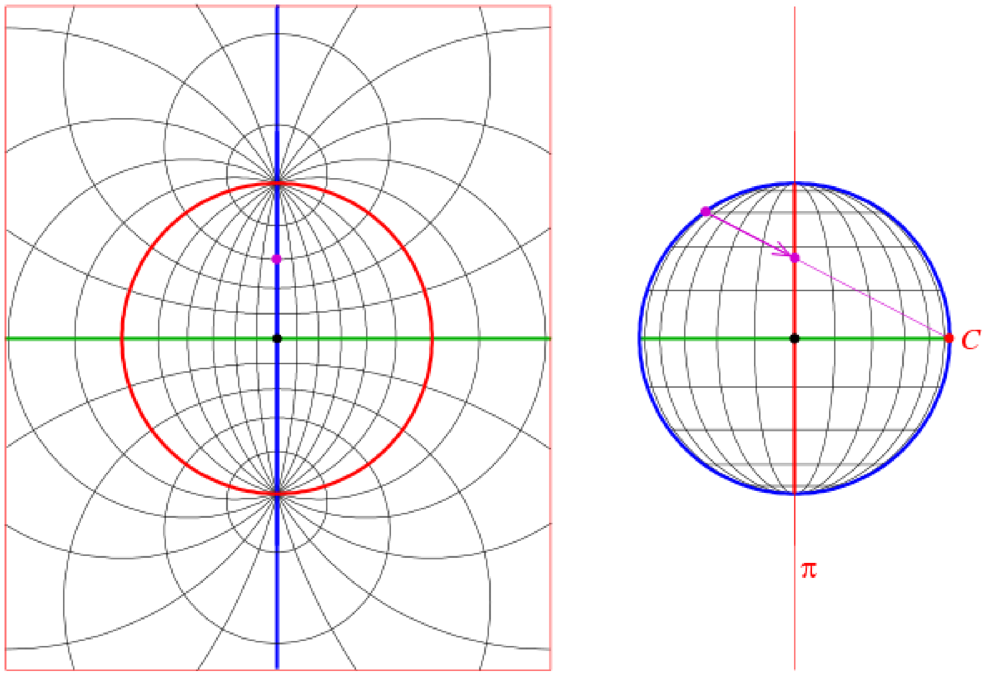
Consider a globe sphere of radius r with the regular lon,lat grid but the grid lines rotated by 90 degrees around an axis that crosses the equator at the longitude $\lambda_P + 90$, where λ_P is the desired longitude of the displaced bi-poles (-300 in MIDAS construction). We denote these set of grid lines by Λ_c and Φ_c. Note that this coordinate patch has two opposing poles at the equator (see the right part of the figure above and imagine the North pole is at the antipole of point C).



Murray’s bipolar grid



Wulf net



Wulf net construction

First we do a stereographic projection P_r of the these two sets of circles onto a plane tangent at the North pole. This results in a polar coordinate patch on that plane (see the left part of the figure above).

Next we do an inverse stereographic projection P_R^{-1} of these polar coordinate curves back on a sphere with a larger radius R to relocate the poles to the desired latitude of the bipolar grid ϕ_P. We denote these sets by Λ_s and Φ_s, these form the desired orthogonal bipolar coordinate patch for the North cap.

4.1 Grid Construction

- globe sphere vs. earth sphere
- (λ_g, ϕ_g): geographical longitude and latitude of a point on a globe sphere
- (λ_c, ϕ_c): intermediate coordinate patch on the globe sphere (rotated-lat-lon with poles on Equator)
- (λ_s, ϕ_s): final curvilinear coordinate patch after streographic projection and re-projection

- ϕ_{bp} : the latitude of the bounding parallel of bipolar grid (the bipolar fold)
- λ_{bp} : the longitude of the split bipole(s).
- Start with a simple lon,lat coordinate patch covering the Northern globe hemisphere:

$$\lambda_g = \lambda_{bp} + i\frac{360}{N_i} \tag{6}$$
$$\phi_g = j\frac{90}{N_j} \tag{7}$$

- Next consider the following two sets of circles Λ_c and Φ_c on the globe sphere:
- Λ_c : The set of great circles with the common axis that crosses the equator at longitude λ_P. Note that these are NOT the geographical meridians, but their 90 degrees rotated images. These circles are formed by the intersection of planes through origin with unit normal vectors $\mathbf{u} = (0, \sin\phi_g, \cos\phi_g)$ and the globe, where ϕ_g is the angle between $\mathbf{u}$ and y axis assuming we set x axis in the direction of λ_{bp}. Hence they obey $\mathbf{u}.(\cos\phi\cos(\lambda - \lambda_{bp}), \cos\phi\sin(\lambda - \lambda_{bp}), \sin\phi) = 0$ which gives:

$$\tan\phi = -\tan\phi_g \sin(\lambda - \lambda_{bp}).$$

- Φ_c : The set of parallel circles with the same axis as above. Note that these are NOT the geographical parallels, but their 90 degrees rotated images.
These circles are formed by the intersection of planes perpendicular to the axis of the great circles above with unit normal vectors $\mathbf{u} = (1, 0, 0)$ and the globe, we set x axis in the direction of λ_P. Hence they obey:

$$\cos\phi\cos(\lambda - \lambda_P) = \cos(\lambda_g - \lambda_{bp}).$$

- The curvilinear coordinate patch (λ_c, ϕ_c) on the globe sphere is obtained by solving the above two equations simultaneously, yielding:

$$\frac{\sin(\lambda_c - \lambda_{bp})}{\tan\phi_c} = -\frac{1}{\tan\phi_g} \equiv \beta \tag{8}$$
$$\cos(\lambda_c - \lambda_{bp})\cos\phi_c = \cos(\lambda_g - \lambda_{bp}) \equiv \alpha \tag{9}$$

which give:

$$\phi_c = \sin^{-1}\left[\left(\frac{1-\alpha^2}{1+\beta^2}\right)^{1/2}\right] \tag{10}$$
$$\lambda_c - 90 = \sin^{-1}\left[\left(\beta^2\frac{1-\alpha^2}{\alpha^2+\beta^2}\right)^{1/2}\right] \tag{11}$$

Note that because of the square roots in the above equations the signs have to be chosen so that the solution remains continuous.

- Finally to achieve the stereographic projection/re-projection (in the spirit of Murray96) we only need to observe a simple relation between the angels in the geometric construction (this is the same as Murray96 equations 1 and 2):

$$\tan(\frac{(90-\phi_s)}{2}) = \tan(\frac{(90-\phi_{bp})}{2})\tan(\frac{(90-\phi_c)}{2}) \tag{12}$$

$$\lambda_s = \lambda_c \tag{13}$$

and so:

$$\phi_s = 90 - 2\tan^{-1}\left(r_P\tan(\frac{(90-\phi_c)}{2})\right)\ ;\ r_P \equiv \tan(\frac{(90-\phi_{bp})}{2}) \tag{14}$$

$$\lambda_s = \lambda_c \tag{15}$$

4.2 Additional degree of freedom in the grid construction

There is an additional degree of freedom in the bipolar grid construction that if ignored would cause a difference with Murray96 grid. We can (and perhaps should) construct the initial northern globe grid latitudes ϕ_g not with a uniform but with a variable resolution such that the final projected/reprojected grid ϕ_s has the desired uniform resolution along the "symmetry meridian" of longitude $\lambda_{bp} - 90$. I.e., we want

$$\Delta\phi_s(\lambda_c = \lambda_P - 90) = Constant = \frac{90-\phi_{bp}}{N_j}. \tag{16}$$

We can find the required ϕ_g that produces this by inverting equation(14):

$$\phi_c = 90 - 2\tan^{-1}\left(\frac{1}{r_P}\tan(\frac{(90-\phi_s)}{2})\right). \tag{17}$$

and noting that according to equation(8) $\phi_c = \phi_g$ along the symmetry meridian $\lambda_c = \lambda_{bp} - 90$. To summarize these transformations schematically:

$$(x,y,z) \xrightarrow{\text{Spherical latlon}} (\lambda_g^0, \phi_g^0) \tag{18}$$

$$\xrightarrow{\text{Inverse Stereographic}} (\lambda_g, \phi_g) = (\lambda_g^0, P^{-1}(\phi_g^0)) \tag{19}$$

$$\xrightarrow{\text{Curvilinear}} (\lambda_c, \phi_c) \tag{20}$$

$$\xrightarrow{\text{Stereographic}} (\lambda_s, \phi_s) = (\lambda_c, P(\phi_c)) \tag{21}$$

4.3 Algorithm

- Generate a mesh using simple lat-lon grid covering the cap

$$(\lambda_g)_{ij} = \lambda_{bp} + i\frac{360}{N_i} \tag{22}$$

$$(\phi_g^0)_{ij} = \phi_{bp} + j\frac{90-\phi_{bp}}{N_j} \tag{23}$$

$$(\phi_g)_{ij} = 90 - 2\tan^{-1}\left(\frac{1}{r_P}\tan(\frac{(90-(\phi_g^0)_j)}{2})\right) \tag{24}$$

- Generate the mesh for $(\lambda_c)_{ij}$ and $(\phi_c)_{ij}$ using equations (9) and (8). Note that we should shift the $(\lambda_c)_{ij}$ by multiples of π to be continuous with $(\lambda_g)_{ij}$ along the Equator.
- Generate the mesh for $(\lambda_s)_{ij}$ and $(\phi_s)_{ij}$ using equations (14).

5 Generating a southern cap displaced pole grid

Here we present the algorithm and code for generating a compute grid for the Antarctic region with the south pole displaced by a given amount. The reason for displacing the pole is to map it as far as possible from the two biggest ice shelves on the continent (**Ross and Ronne**) to have a viable compute grid over them. We generate the grid starting from a usual lat-lon grid on the sphere followed by a series of transformations from the spherical surface of the Earth to a tangent plane at the south pole [2][3]. Particularly the displacement of the pole is accomplished via a conformal mapping in the tangent plane [4].

5.1 Grid Parameters

- ϕ_{SO}: ending latitude of the southern ocean grid
- ϕ_{join}: starting latitude of the antarctic cap grid (the join). Here we assume $\phi_{join} = \phi_{SO}$
- λ_{dp}: longitude of the displaced pole
- ϕ_{dp}: the latitude of the displaced pole

- r_{dp}: the fractional distance of the displaced pole from the geographic pole in units of $\tan(90 + \phi_{join})$

5.2 Algorithm

- Generate a regular lat-lon grid with desired resolution covering the Earth south of $\phi_{join} = \phi_{SO}$, (λ, ϕ) .
- Project the lat-lon grid to a plane tangent at the south pole (simple gnomonic projection)(not conformal) to planar polar coordinates:

$$r = R_E \tan(90 + \phi) \tag{25}$$
$$\theta = \lambda - \lambda_p \tag{26}$$

- Rescale r so that the parallel at ϕ_{join} maps to the unit circle

$$r = \tan(90 + \phi)/r_{join}; \quad r_{join} \equiv \tan(90 + \phi_{join}) \tag{27}$$
$$\tag{28}$$

- Use the inverse conformal map to transform θ so that the angular resolution at the unit circle in w-plane (see the next step) matches the corresponding resolution in z-plane (which matches the longitudinal resolution at the joint) :

$$z_0 = r_{dp} e^{i\lambda_p} \tag{29}$$

$$e^{i\theta'} = w^{-1}(e^{i\theta}) = \frac{e^{i\theta} + z_0}{1 + z_0^* e^{i\theta}} \tag{30}$$

- Define

$$z = re^{i\theta'} \tag{31}$$

- Do a conformal transformation to map the unit disk to a unit disk with a displaced center:

$$w = \frac{z - z_0}{1 - z_0^* z} \tag{32}$$

- Project back to the sphere tangent at the geographical south pole to form the desired curvilinear coordinates:

$$\lambda_c = \arg(w) \tag{33}$$

$$\phi_c = \tan^{-1}\left(|w| r_{join}\right) - 90 \tag{34}$$

6 Grid Metrics

We can generate the "metrics" by analytical manipulations. The length of a curve which is determined by integrating the infinitesimal length ds along the curve is independent of the coordinates. This statement determines the "metric" tensor g of the coordinates:

$$ds^2 = \sum_{\mu,\nu} g^{\mu\nu} d\xi_\mu d\xi_\nu \tag{35}$$

In this writing μ and ν range over $1, 2, 3$, but the above formula is much more general than that. For a Cartesian coordinate $g_{\mu\nu} = \delta_{\mu,\nu}$ and this is simply the Eucldian length $dx_1^2 + dx_2^2 + dx_3^2 = dx^2 + dy^2 + dz^2$. For a general orthogonal coordinate system such as (λ_s, ϕ_s) the metric is diagonal and is sometimes denoted by a set of h's:

$$ds^2 = \sum_{\mu} g^{\mu\mu} (d\xi_\mu)^2 \equiv \sum_{\mu} (h_\mu)^{-2} (d\xi_\mu)^2 \tag{36}$$

Note that in our case the new coordinates are the indexes (I, J) (captalized to avoid confusion with their descrete values i, j) and not (λ_s, ϕ_s)! So, starting from the Cartesian coordinates (x_1, x_2, x_3) and using the chain rule we get:

$$
\begin{aligned}
ds^2 &= \sum_\mu dx_\mu^2 \\
&= \sum_\mu \left(\frac{\partial x_\mu}{\partial I} dI + \frac{\partial x_\mu}{\partial J} dJ \right)^2 \\
&= \sum_\mu \left[\left(\frac{\partial x_\mu}{\partial \lambda_s} \frac{\partial \lambda_s}{\partial I} + \frac{\partial x_\mu}{\partial \phi_s} \frac{\partial \phi_s}{\partial I} \right) dI + (I \to J)\, dJ \right]^2
\end{aligned}
\tag{37}
$$

We can obtain formulas for h's by considering curves along new coordinates:

$$
h_I^{-2} \equiv ds^2(dJ=0) = dI^2 \sum_\mu \left[(\frac{\partial x_\mu}{\partial \lambda_s})^2 (\frac{\partial \lambda_s}{\partial I})^2 + (\frac{\partial x_\mu}{\partial \phi_s})^2 (\frac{\partial \phi_s}{\partial I})^2 + crossterm \right]
\tag{38}
$$

Cross terms are generally zero if the new coordinates are orthogonal. We can easily verify this for the from of coordinate transformation at hand (which is a spherical lat-lon):

$$
\begin{aligned}
x_1 &= x = \cos(\phi_s)\cos(\lambda_s) \\
x_2 &= y = \cos(\phi_s)\sin(\lambda_s) \\
x_3 &= z = \sin(\phi_s)
\end{aligned}
\tag{39}
$$

$$
\begin{aligned}
crossterm = 2\sum_\mu ()()()() &= 2(\frac{\partial \lambda_s}{\partial I})(\frac{\partial \phi_s}{\partial I}) \left[\sum_\mu (\frac{\partial x_\mu}{\partial \lambda_s})(\frac{\partial x_\mu}{\partial \phi_s}) \right] \\
&= 2(\frac{\partial \lambda_s}{\partial I})(\frac{\partial \phi_s}{\partial I}) \left[\cos(\phi_s)\sin(\lambda_s)\sin(\phi_s)\cos(\lambda_s) - \cos(\phi_s)\cos(\lambda_s)\sin(\phi_s)\sin(\lambda_s) \right] \quad = 0
\end{aligned}
\tag{40}
$$

So,

$$
\begin{aligned}
ds^2(dJ=0) = h_I^{-2} &= dI^2 \left[(\frac{\partial \lambda_s}{\partial I})^2 \left(\cos^2(\phi_s)\sin^2(\lambda_s) + \cos^2(\phi_s)\cos^2(\lambda_s) + 0 \right) + (\frac{\partial \phi_s}{\partial I})^2 \left(\sin^2(\phi_s)\cos^2(\lambda_s) + \sin^2(\phi \right.\right. \\
&= dI^2 \left[(\frac{\partial \lambda_s}{\partial I})^2 \cos^2(\phi_s) + (\frac{\partial \phi_s}{\partial I})^2 \right]
\end{aligned}
\tag{41}
$$

$$
\begin{aligned}
ds^2(dI=0) = h_J^{-2} &= dJ^2 \left[I \to J \right] \\
&= dJ^2 \left[(\frac{\partial \lambda_s}{\partial J})^2 \cos^2(\phi_s) + (\frac{\partial \phi_s}{\partial J})^2 \right]
\end{aligned}
\tag{42}
$$

6.1 Metrics for Bipolar Cap Grid

Starting with the grid construction equation for the bipolar cap (14) and applying the chain rule we can calculate the terms one by one:

$$\begin{aligned}\frac{\partial\lambda_s}{\partial I} &= \frac{\partial\lambda_c}{\partial I}\\ &= \frac{\partial\lambda_c}{\partial\lambda_g}\frac{\partial\lambda_g}{\partial\lambda_g^0}\frac{\partial\lambda_g^0}{\partial I} + \frac{\partial\lambda_c}{\partial\phi_g}\frac{\partial\phi_g}{\partial\phi_g^0}\frac{\partial\phi_g^0}{\partial I}\\ &= \frac{\partial\lambda_c}{\partial\lambda_g}\frac{\partial\lambda_g^0}{\partial I}\end{aligned} \tag{43}$$

Similarly

$$\frac{\partial\phi_s}{\partial I} = \frac{\partial\phi_s}{\partial\phi_c}\frac{\partial\phi_c}{\partial\lambda_g}\frac{\partial\lambda_g^0}{\partial I} \tag{44}$$

$$\frac{\partial\lambda_s}{\partial J} = \frac{\partial\lambda_c}{\partial\phi_g}\frac{\partial\phi_g}{\partial\phi_g^0}\frac{\partial\lambda_g^0}{\partial J} \tag{45}$$

$$\frac{\partial\phi_s}{\partial J} = \frac{\partial\phi_s}{\partial\phi_c}\frac{\partial\phi_c}{\partial\phi_g}\frac{\partial\phi_g}{\partial\phi_g^0}\frac{\partial\phi_g^0}{\partial J} \tag{46}$$

Now by brute force differentiation of the curvilinear transformation formulas equations (10),(11) we get

$$(\frac{\partial\lambda_c}{\partial\lambda_g})^2 = \frac{1+\beta^{-2}}{(1+\alpha^2\beta^{-2})^2} \tag{47}$$

$$(\frac{\partial\lambda_c}{\partial\phi_g})^2 = \frac{\alpha^2(1-\alpha^2)\beta^{-2}(1+\beta^{-2})}{(1+\alpha^2\beta^{-2})^2} \tag{48}$$

$$(\frac{\partial\phi_c}{\partial\lambda_g})^2 = \frac{\alpha^2\beta^{-2}}{1+\alpha^2\beta^{-2}} \tag{49}$$

$$(\frac{\partial\phi_c}{\partial\phi_g})^2 = \frac{1-\alpha^2}{1+\alpha^2\beta^{-2}} \tag{50}$$

And similarly differentiation of the stereographic transformation formula equations (14),(17) yield:

$$M^{-1} \equiv \frac{\partial \phi_s}{\partial \phi_c} = r_P \frac{1+\tan^2(\chi_c/2)}{1+r_P^2\tan^2(\chi_c/2)}; \;\; \chi_c \equiv 90 - \phi_c$$

$$N \equiv \left(\frac{\partial \phi_g}{\partial \phi_g^0}\right)^{-1} = r_P \frac{1+\tan^2(\chi_g/2)}{1+r_P^2\tan^2(\chi_g/2)}; \;\; \chi_g \equiv 90 - \phi_g \tag{51}$$

We note that

$$\frac{\partial \lambda_g^0}{\partial I} = 2\pi/N_i \frac{\partial \phi_g^0}{\partial J} = (90 - \phi_{bp})/N_j \tag{52}$$

Putting it all together we finally obtain the metric terms

$$h_I^{-2} = \left[\cos^2(\phi_s)\frac{1+\beta^{-2}}{(1+\alpha^2\beta^{-2})^2} + M^{-2}\frac{\alpha^2\beta^{-2}}{1+\alpha^2\beta^{-2}}\right]\left(\frac{2\pi}{N_i}\right)^2 \tag{53}$$

$$h_J^{-2} = \left[\cos^2(\phi_s)N^{-2}\frac{\alpha^2(1-\alpha^2)\beta^{-2}(1+\beta^{-2})}{(1+\alpha^2\beta^{-2})^2} + M^{-2}N^{-2}\frac{1-\alpha^2}{1+\alpha^2\beta^{-2}}\right]\left(\frac{\pi/2-\phi_{bp}}{N_j}\right)^2 \tag{54}$$

The value of the metric tensor is that it can yield the infinitesimal length along coordinate curves. Noting that $dI = dJ = 1$ for a grid element we have:

$$i - direction\ length\ element = \sqrt{ds^2(dI=1, dJ=0)} = h_I^{-1} \tag{55}$$

$$j - direction\ length\ element = \sqrt{ds^2(dI=0, dJ=1)} = h_J^{-1} \tag{56}$$

$$grid - cell\quad area\ element = h_I^{-1}.h_J^{-1} \tag{57}$$

7 Appendix A

7.1 Murray 96 Comparison

Murray's grid is based on the same idea of projecting and re-projecting from globe sphere to plane to Earth sphere, with the difference that he has the plane tangent at the equator at the longitude of the desired final bipoles. Hence he has an extra 90 degrees rotation to deal with to bring the grid onto the Northern cap. This is rather complicated since it exchanges the roles of latitude and longitude at some point in the construction which is not totally explained in his paper. Otherwise the form of the constructing formulas are equivalent to the ones above in spirit , but left their interpretation is not clear. We copy Murray 96 formulas below for comparison.

Murray96 equations (1) and (2) are

$$Murray1: \quad \tan(\frac{\chi_s}{2}) = r_P\tan(\frac{\chi_c}{2}) \tag{58}$$

$$Murray2: \quad r_P = \tan(\frac{\chi_P}{2}) \tag{59}$$

This is a geometric property of a stereographic projection of sphere onto a tangent plane and then projecting back onto a sphere of larger radius tangent at the same point to the plane. These two sphere have a common axis RQ passing through their centers and the tangent point and these are simple the relations between three spherical angels; 1) χ_P is the spherical angle between RQ and the line joining the center of the (larger) earth sphere to the north pole of the (smaller) globe sphere, 2) χ_c is the spherical angle between RQ and the line joining the center of the globe sphere to a given point X on that sphere, 3) χ_s is the spherical angle between RQ and the line joining the center of the Earth sphere to the re-pojected image of the point X on that sphere (refer to FIG.3 of Murray96).

Murray96 equations (5) and (6) for generating the curvilinear grid are

$$Murray5a: \quad \lambda_s = \lambda_0 + 90 - atan2(sin\lambda_c, \tan\phi_c) \tag{60}$$

$$Murray5b: \quad \phi_s = 90 - 2\tan^{-1}\left(r_p \tan(\frac{\chi_c}{2})\right) \tag{61}$$

$$Murray6: \quad \chi_c = \cos^{-1}\left(\cos\lambda_c \cos\phi_c\right) \tag{62}$$

The above equation(14) would be identical to Murray's Eq(5b) if we assume $\chi_c = 90 - \phi_c$. But then Murray's Eq(6) $\cos\chi_c = \cos\lambda_c \cos\phi_c$, would be in conflict with that assumption (relates ϕ_c to λ_c)! On the other hand Murray's Eq(6) is somewhat similar to the equation (9) above if we assume $\chi_c = \lambda_g - \lambda_P$, which would then be in apparent conflict with equation(14) above.

Murray's Eq(5a) is similar to equation(8) above if we assume that $\lambda_s - \lambda_P = \phi_g$, which is again in apparent conflict with all other equations.

Moreover in MIDAS (and FMS) implementation λ_c and phi_c are derived from ϕ_g and λ_g respectively, contrary to the intuition (λ_c is derived via solving Murray's Eq(5a) for λ_c at $\phi_c = 0$ given $\lambda_s = \phi_g$ and ϕ_c is set to $90 - \lambda_g$.)

All these conflicts point to a subtle switching of the roles of latitude and longitude at the connection line in the transformation defined in Murray which he mentions only in passing at section 2.2.

8 References

[1] Explicit Generation of Orthogonal Grids for Ocean Models, Ross J.Murray, Journal of Computational Physics Volume 126, Issue 2, July 1996, Pages 251-273

[2] Roberts, J. L., Heil, P., Murray, R. J., Holloway, D. S., and Bindoff, N. L.: Pole relocation for an orthogonal grid: An analytic method, Ocean Model., 12, 16–31, doi:10.1016/j.ocemod.2005.03.004, 2006.

[3] Xu, S., Wang, B., and Liu, J.: On the use of Schwarz–Christoffel conformal mappings to the grid generation for global ocean models, Geosci. Model Dev., 8, 3471-3485, https://doi.org/10.5194/gmd-8-3471-2015, 2015.

[4] Basic Complex Analysis, J.E. Marsden, M.J. Hoffman, 2nd Edition, p.355.

8.1 Explore the Grid Generation Tool

This tool can be called to generate a grid with nominal resolution of 1/4 degrees by the following command line:

```
ocean_grid_generator.py -f tripolar_res4_areacorrected.nc -r 4 –south_cutoff_row 81
```

You can also explore the tool by the following jupyter notebook.

```
In [13]: import imp
         import ocean_grid_generator as ogg
         import matplotlib.pyplot as plt
         %matplotlib inline
         import math
         import numpy as np
```

```
In [14]: ### Specify the grid properties
         inverse_nominal_resolution=4 # This is the inverse of the nominal resolution in degrees
         lenlon=360  # global longitude range
         Ni=2*lenlon* inverse_nominal_resolution #Number of longitude partitions Eq(5)

         #Mercator grid nominal extents
         #These values were chosen to generate a grid very close to the MIDAS grid for 1/4 degre
         phi_s_Merc, phi_n_Merc = -66.85954724706843, 64.0589597296948

         lon0=-300. #Starting longitude (also the longitude of the Northern bipoles)
                    #Why this particular value? I think it's chosen so that North America will b
```

```
In [15]: #Reload the grid generating tool (in case you change something in the tool and want it
         imp.reload(ogg)

         #Generate the Mercator grid
         lamMerc,phiMerc = ogg.generate_mercator_grid(Ni,phi_s_Merc,phi_n_Merc,lon0,lenlon,ensur

         #The phi resolution in the last row of Mercator grid along the symmetry meridian
         DeltaPhiMerc_so = phiMerc[ 1,Ni//4]-phiMerc[ 0,Ni//4]
         DeltaPhiMerc_no = phiMerc[-1,Ni//4]-phiMerc[-2,Ni//4]

         #Generate the Bipolar grid.
         lat0_bp = phiMerc[-1,Ni//4] + DeltaPhiMerc_no #Start from a lattitude to smooth out dy.

         Nj_ncap=inverse_nominal_resolution* 120 - 1   #MIDAS has refineS*( 240 for 1/4 degree,
         print("MIDAS number of bipolar cap js= ", Nj_ncap)

         Nj_ncap = int(0.5+(90.-lat0_bp)/DeltaPhiMerc_no) #Impose boundary condition for smooth
         print("New   number of bipolar cap js= ", Nj_ncap)

         lamBP,phiBP = ogg.generate_bipolar_cap_grid(Ni,Nj_ncap,lat0_bp,lon0,lenlon)

         #ogg.plot_mesh_in_xyz(lamBP, phiBP, stride=20)

         print("Last  lattitude of Mercator",phiMerc[-1,0])
         print("First lattitude of Bipolar ",phiBP[0,0])
         print("DM  ",phiMerc[-1,Ni//4]-phiMerc[-2,Ni//4])
         print("DMB ",phiBP[0,Ni//4]-phiMerc[-1,Ni//4])
         print("DB  ",phiBP[1,Ni//4]-phiBP[0,Ni//4])
```

```
#Generate the Southern Ocean grid
lat0_SO=-78.0
lenlat_SO = phiMerc[0,Ni//4] - DeltaPhiMerc_so - lat0_SO #Start from a lattitude to smo
Nj_SO = int(0.5 + lenlat_SO/DeltaPhiMerc_so) #nearest integer
lamSO,phiSO = ogg.generate_latlon_grid(Ni,Nj_SO,lon0,lenlon,lat0_SO,lenlat_SO)

print("First lattitude of Mercator",phiMerc[0,0])
print("Last  lattitude of SO       ",phiSO[-1,0])
print("DM  ",phiMerc[1,Ni//4]-phiMerc[0,Ni//4])
print("DMSO",phiMerc[0,Ni//4]-phiSO[-1,Ni//4])
print("DSO ",phiSO[-1,Ni//4]-phiSO[-2,Ni//4])
print("DSO ",phiSO[1,Ni//4]-phiSO[0,Ni//4])


#Generate the southern cap grid
lon_dp=100.0   # longitude of the displaced pole
r_dp =0.2
doughnut=0.0
deltaPhiSO = phiSO[1,Ni//4]-phiSO[0,Ni//4]
lat0_SC=phiSO[0,Ni//4]-deltaPhiSO
fullArc = lat0_SC+90.
halfArc = fullArc/2
#halfArc = lat0_SC - 0.5*(lat0_SC-90.)
nparts=8

Nj_scap=inverse_nominal_resolution*  40   #MIDAS has refineS*(  80 for 1/4 degree, ???
print("MIDAS number of southern cap js= ", Nj_scap)

Nj_scap = int((nparts/(nparts-1))*halfArc/deltaPhiSO) #Impose boundary condition for sm
print("new   number of southern cap js= ", Nj_scap)

lamc_DP,phic_DP = ogg.generate_displaced_pole_grid(Ni,Nj_scap,lon0,lenlon,lon_dp,r_dp,l

print("First lattitude of SO ",phiSO[0,0])
print("Last  lattitude of DP ",phic_DP[-1,0])
print("DSO ",phiSO[1,Ni//4]-phiSO[0,Ni//4])
print("DSS ",phiSO[0,Ni//4]-phic_DP[-1,Ni//4])
print("DSC ",phic_DP[-1,Ni//4]-phic_DP[-2,Ni//4])
print("DSC ",phic_DP[1,Ni//4]-phic_DP[0,Ni//4])

lat_dp = -90 + np.arctan(r_dp*np.tan((90+lat0_SC)*np.pi/180))*180/np.pi
print("The geographical location of the displaced pole=",lon_dp,lat_dp)

#print("cut the grid below row 81")
#lamc_DP = lamc_DP[80:,:]
#phic_DP = phic_DP[80:,:]

print("Starting grid lattitude =", phic_DP[0,0])
```

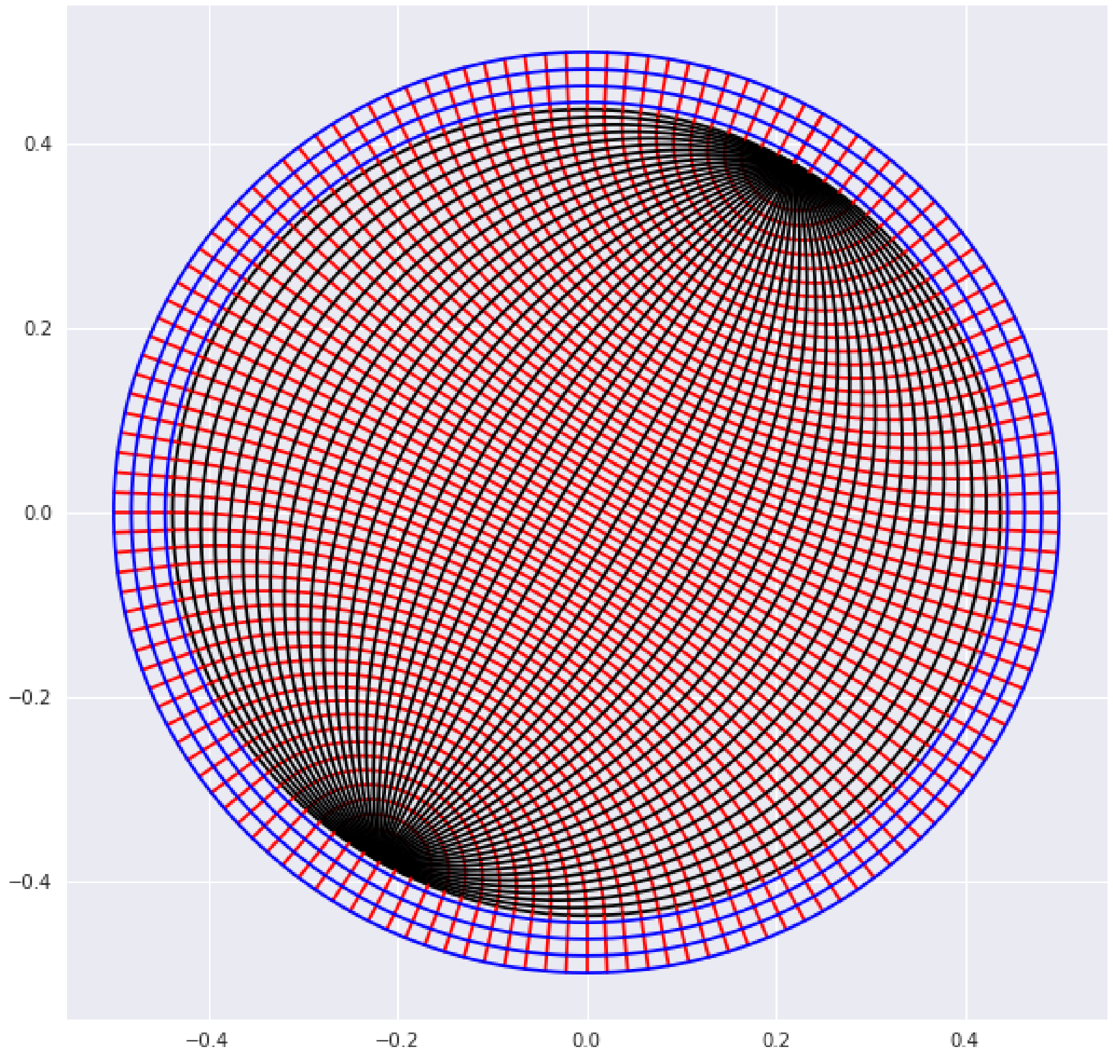
```
Requesting Mercator grid with phi range: phi_s,phi_n= -66.85954724706843 64.0589597296948
   y*= [-727  673] nj= 1401
   Supergrid has an odd number of area cells!
   Fixing this by shifting the y_star[1]
   y*= [-727  672] nj= 1400
   Generating Mercator grid with phi range: phi_s,phi_n= [-66.85954725  64.00422531]
   Equator is at j= 727
   number of js= 1400
MIDAS number of bipolar cap js=  479
New   number of bipolar cap js=  473
Generating bipolar grid bounded at latitude  64.0590671656
   number of js= 474
Last  lattitude of Mercator 64.004225313
First lattitude of Bipolar  64.0590671656
DM   0.0548418526513
DMB  0.0548418526513
DB   0.0548434097979
Generating regular lat-lon grid between latitudes  -78.0 -66.9087198534
   The number of j's is not even. Fixing this by cutting one row at south.
   generated regular lat-lon grid between latitudes  -77.9509235392 -66.9087198534
   number of js= 226
First lattitude of Mercator -66.8595472471
Last  lattitude of SO       -66.9087198534
DM   0.04917260638
DMSO 0.04917260638
DSO  0.0490764608254
DSO  0.0490764608254
MIDAS number of southern cap js=  160
new   number of southern cap js=  139
Generating displaced pole grid bounded at latitude  -78.0
   rdp= 0.2  , doughnut= 0.0
   The number of j's is not even. Fixing this by cutting one row at south.
   number of js= 138
First lattitude of SO  -77.9509235392
Last  lattitude of DP  -78.0
DSO  0.0490764608254
DSS  0.0490764608254
DSC  0.0466578121629
DSC  0.0675195684747
The geographical location of the displaced pole= 100.0 -87.5657469192
Starting grid lattitude = -87.2489631053
```

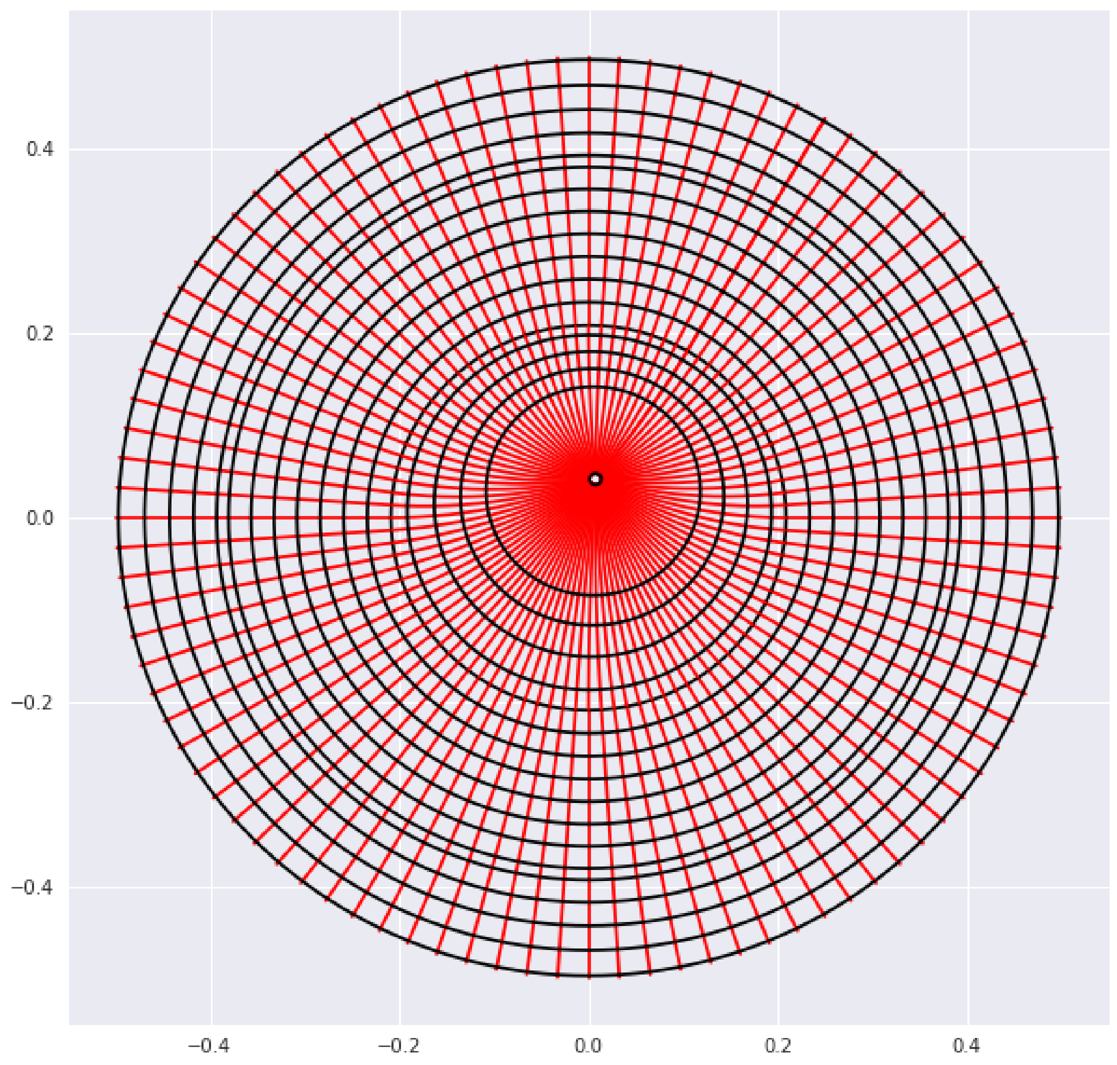
8.2 Visualize

```
In [16]: #Visualize


         import matplotlib.pyplot as plt
         import seaborn as sns; sns.set()

         def plot_mesh_in_latlon(lam, phi, stride=1, phi_color='k', lam_color='r', newfig=True,
             if (phi.shape != lam.shape): raise Exception('Ooops: lam and phi should have same s
             nj,ni = lam.shape
             if(newfig):
                 plt.figure(figsize=(10,10))
             for i in range(0,ni,stride):
                 plt.plot(lam[:,i],phi[:,i],lam_color)
             for j in range(0,nj,stride):
                 plt.plot(lam[j,:],phi[j,:],phi_color)
             if title is not None:
                 plt.title(title)

         def plot_mesh_in_xyz(lam, phi, stride=1, phi_color='k', lam_color='r', lowerlat=None, u
             PI_180 = np.pi/180.
             if lowerlat is not None:
                 lam,phi = cut_below(lam,phi,lowerlat=lowerlat)
             if upperlat is not None:
                 lam,phi = cut_above(lam,phi,upperlat=upperlat)
             x = np.cos(phi*PI_180) * np.cos(lam*PI_180)
             y = np.cos(phi*PI_180) * np.sin(lam*PI_180)
             z = np.sin(phi*PI_180)
             plot_mesh_in_latlon(x, y, stride=stride, phi_color=phi_color, lam_color=lam_color,

         def cut_below(lam,phi,lowerlat):
             nj,ni = lam.shape
             for j in range(0,nj):
                 if(phi[j,0]>lowerlat):
                     break
             jmin=j
         #    print("jmin",jmin)
             return lam[jmin:,:], phi[jmin:,:]

         def cut_above(lam,phi,upperlat):
             nj,ni = lam.shape
             for j in range(0,nj):
                 if(phi[j,0]>upperlat):
                     break
             jmax=j
         #    print("jmax",jmax)
             return lam[0:jmax,:], phi[0:jmax,:]
```

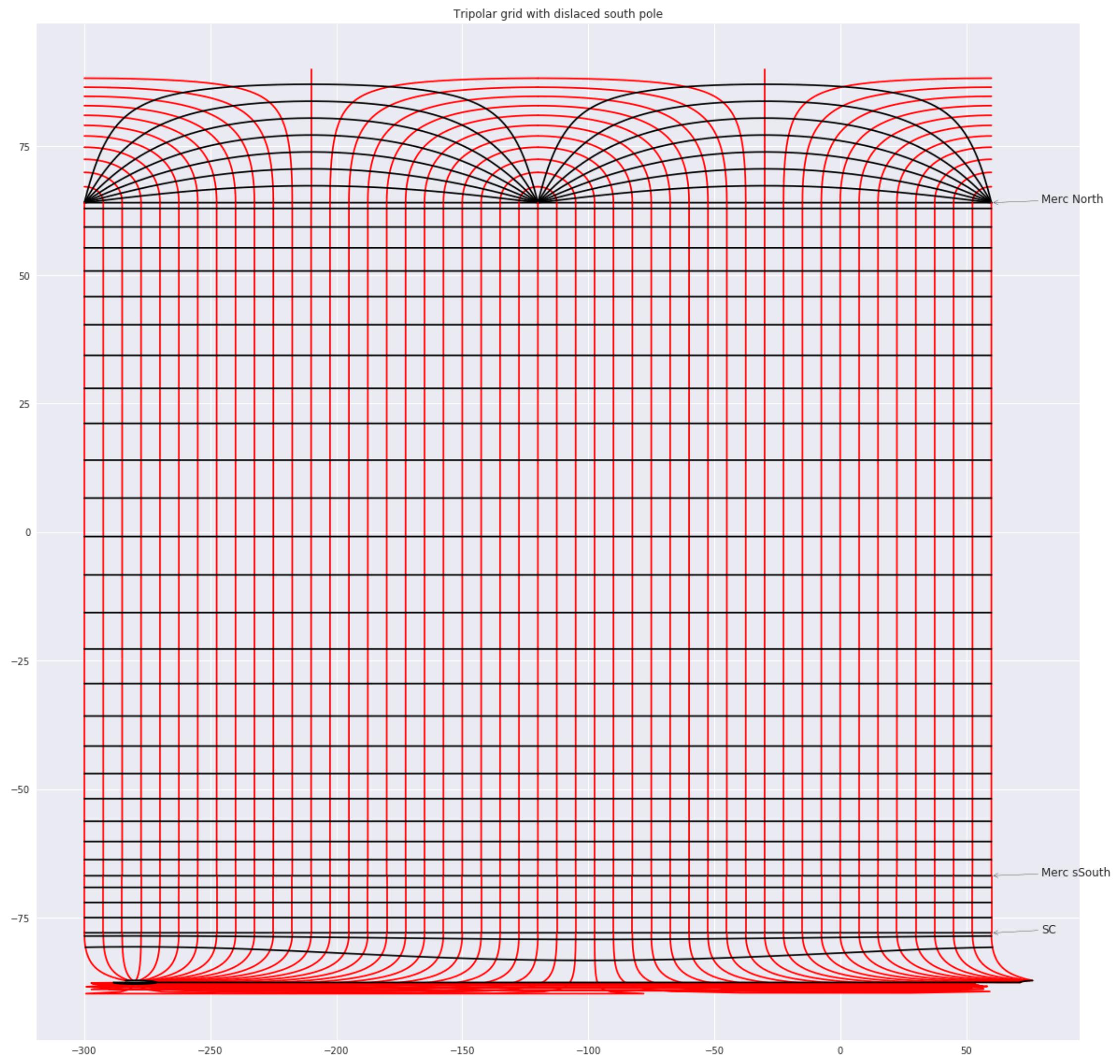
```
In [17]: #plot grids
         plt.figure(figsize=(10,10))
         plot_mesh_in_xyz(lamBP, phiBP, stride=20,newfig=False)
         plot_mesh_in_xyz(lamMerc,phiMerc, phi_color='b', stride=20,lowerlat=60,newfig=False)
```

```
plt.figure(figsize=(10,10))
plot_mesh_in_xyz(lamSO,phiSO, stride=30,newfig=False)
plot_mesh_in_xyz(lamc_DP,phic_DP, stride=30,newfig=False)
plot_mesh_in_xyz(lamMerc,phiMerc, stride=30, upperlat=-60,newfig=False)
```





```
In [18]: plt.figure(figsize=(20,20))
         plt.title("Tripolar grid with dislaced south pole")
         plot_mesh_in_latlon(lamBP, phiBP, stride=60,newfig=False)
         plot_mesh_in_latlon(lamMerc,phiMerc, stride=60,newfig=False);
         plot_mesh_in_latlon(lamSO,phiSO, stride=60,newfig=False)
         plot_mesh_in_latlon(lamc_DP,phic_DP, stride=60,newfig=False)
         #plt.arrow(70,-75,10,10)
         plt.annotate("Merc North",xy=(60,phi_n_Merc), xytext=(80, phi_n_Merc),arrowprops=dict(a
         plt.annotate("Merc sSouth",xy=(60,phi_s_Merc), xytext=(80, phi_s_Merc),arrowprops=dict(
         plt.annotate("SC",xy=(60,lat0_SO), xytext=(80, lat0_SO),arrowprops=dict(arrowstyle="->"

Out[18]: <matplotlib.text.Annotation at 0x2ba644b6db38>
```

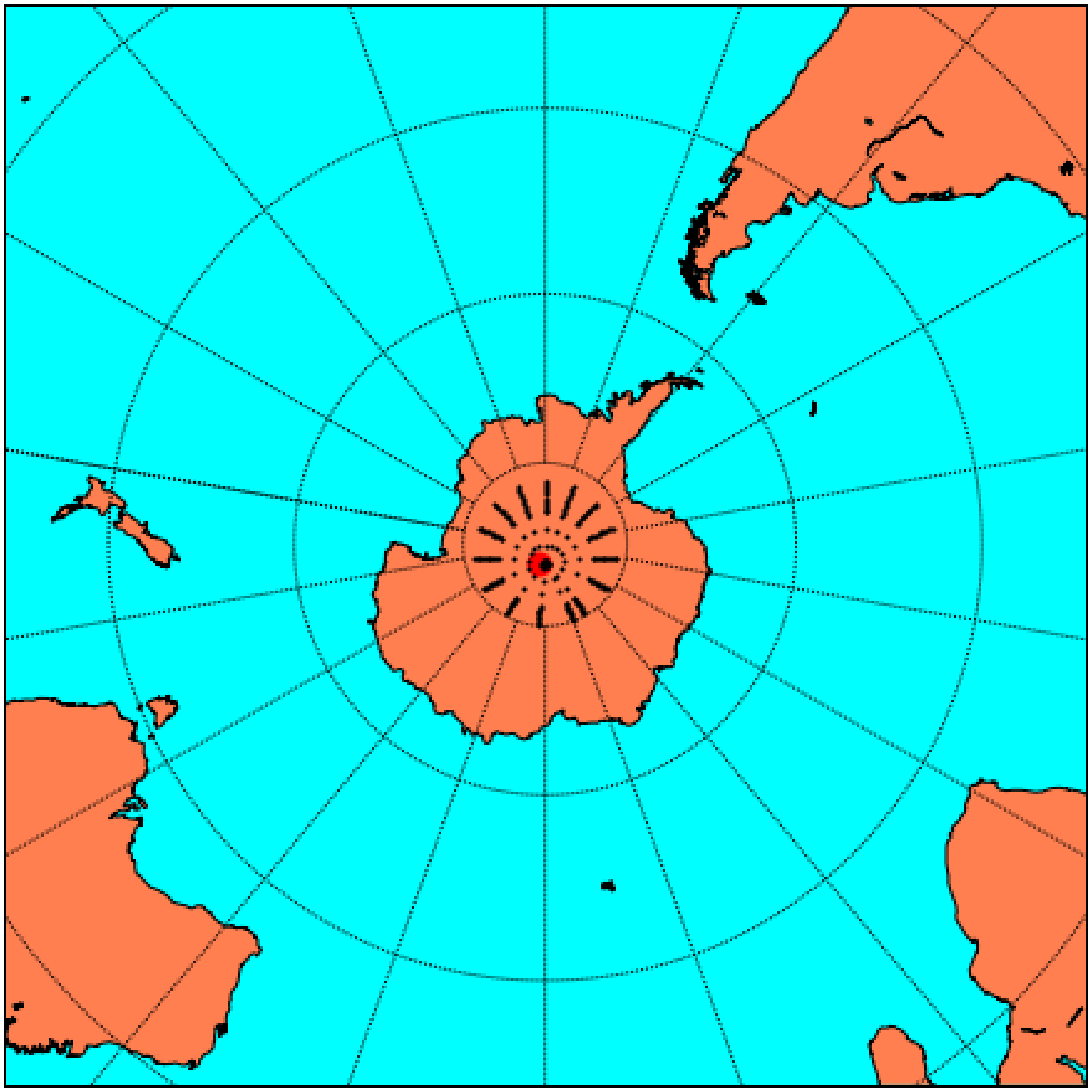


Why the particular values $r_{dp} = 0.2$ and $\lambda_{dp} = 100$?

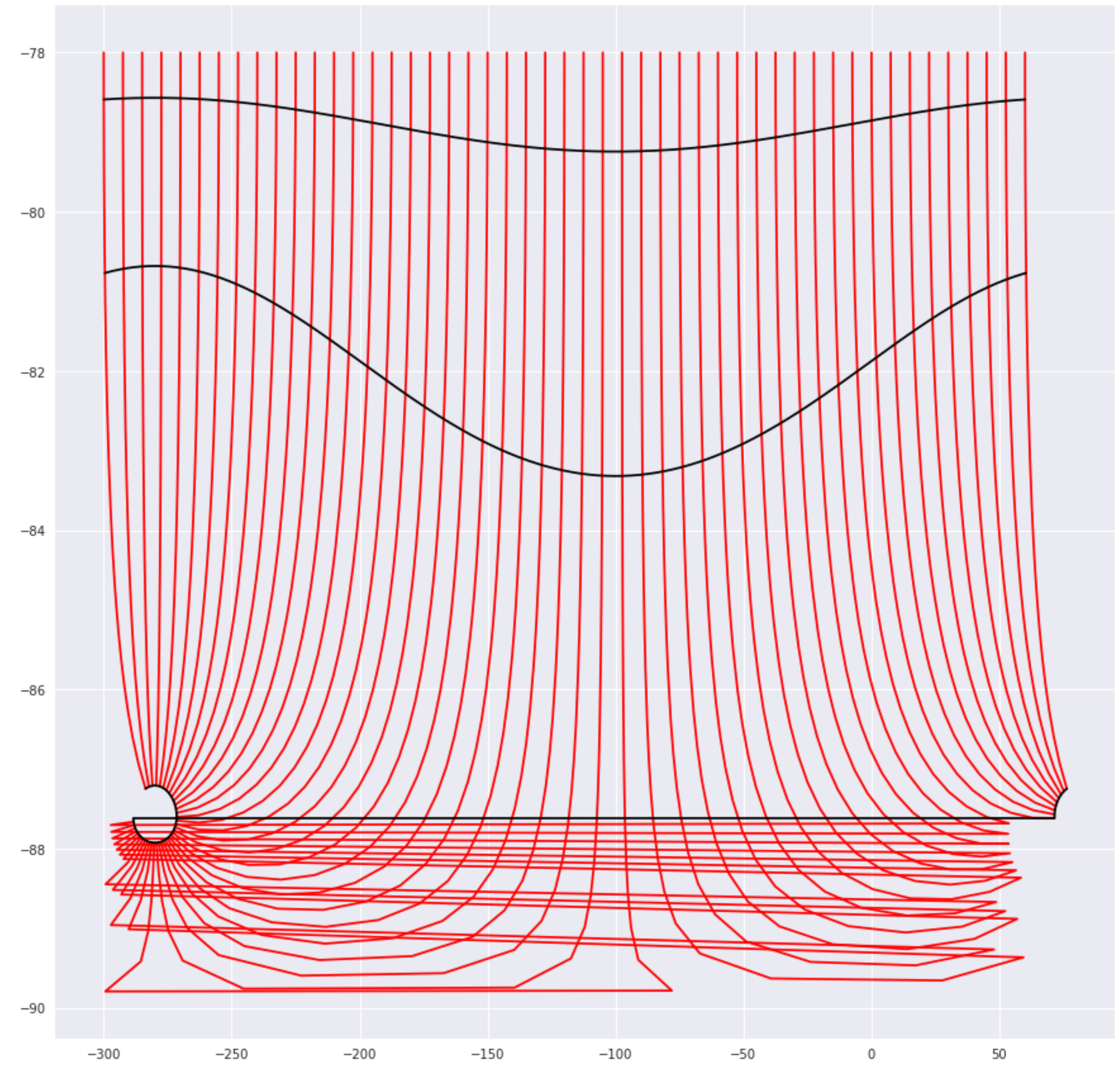
The figure below shows that we could optimize these values to cut more land points out of the ocean grid.

```
In [19]: from mpl_toolkits.basemap import Basemap
         plt.figure(figsize=(8, 8))
         # setup south polar stereographic basemap.
         # The longitude lon_0 is at 6-o'clock, and the
         # latitude circle boundinglat is tangent to the edge
         # of the map at lon_0. Default value of lat_ts
         # (latitude of true scale) is pole.
         m = Basemap(projection='spstere',boundinglat=-30,lon_0=80,resolution='l')
         m.drawcoastlines()
         m.fillcontinents(color='coral',lake_color='aqua')
         m.drawparallels(np.arange(-80.,81.,20.))
```

```
m.drawmeridians(np.arange(-180.,181.,20.))
m.drawmapboundary(fill_color='aqua')
x, y = m(lon_dp, lat_dp)
plt.plot(x, y, 'ok', markersize=10,color='r')
#plt.text(x, y, ' DP', fontsize=12);
x, y = m(lamc_DP[:80:5,::200],phic_DP[:80:5,::200])
plt.plot(x, y, 'ok', markersize=2)
plt.show()
```

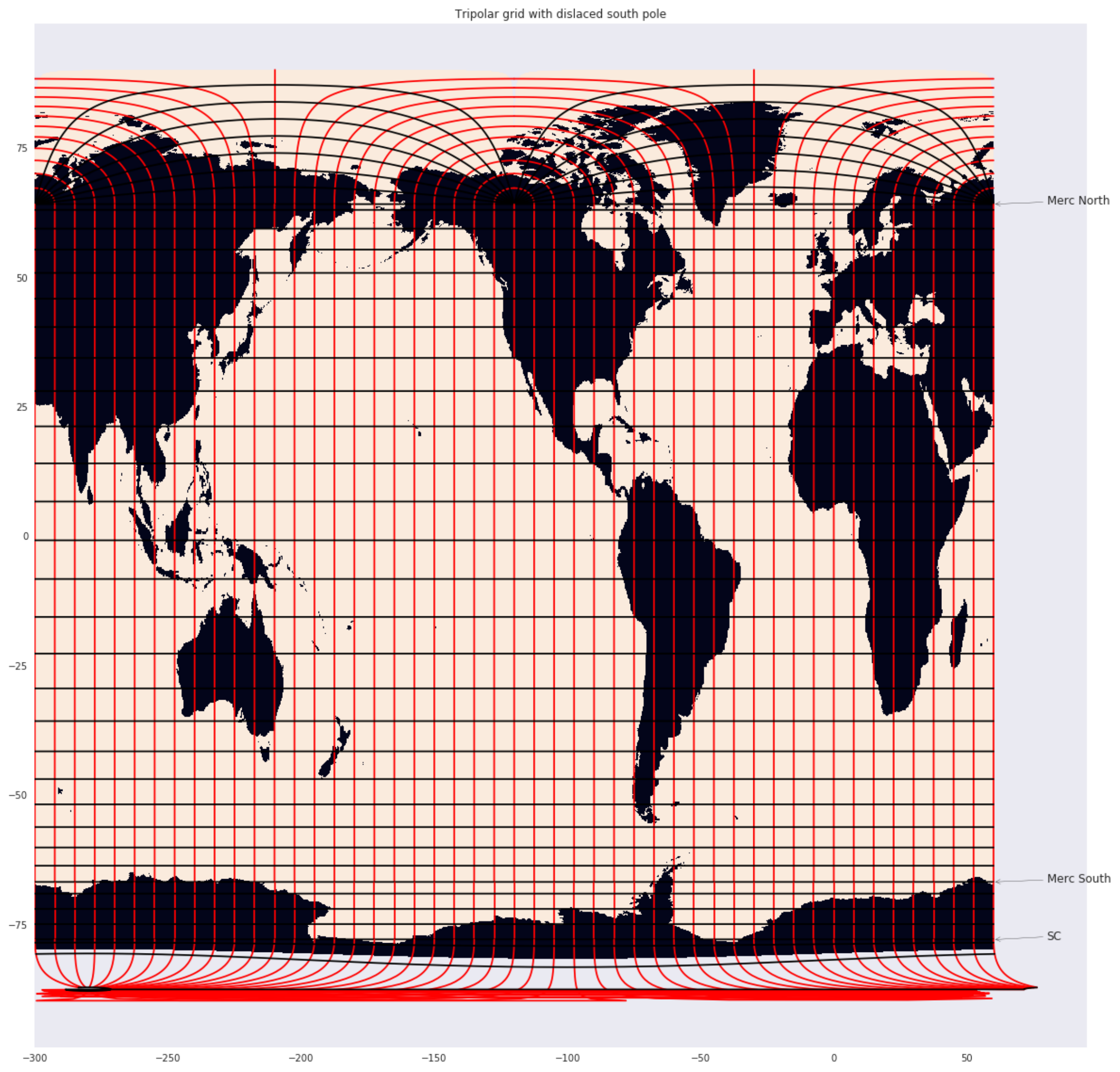


```
In [20]: plt.figure(figsize=(15,15))
         plot_mesh_in_latlon(lamc_DP,phic_DP, stride=60,newfig=False)
```



```
In [21]: import netCDF4 as nc
         ocean_static = nc.Dataset('/ptmp/oar.gfdl.ogrp-account/archive/oar.gfdl.ogrp-account/CM
         #for var in ocean_static.variables: print(var,end=' ')
         # xh yh time xq yq areacello deptho hfgeou sftof Coriolis geolon geolat geolon_c geolat
         geolon = np.array(ocean_static.variables['geolon'])
         geolat = np.array(ocean_static.variables['geolat'])
         wet = np.array(ocean_static.variables['wet'])
         plt.figure(figsize=(20,20))
         plt.title("Tripolar grid with dislaced south pole")

         #plt.pcolormesh(wet); plt.colorbar()

         plot_mesh_in_latlon(lamBP, phiBP, stride=60,newfig=False)
         plot_mesh_in_latlon(lamMerc,phiMerc, stride=60,newfig=False);
         plot_mesh_in_latlon(lamSO,phiSO, stride=60,newfig=False)
         plot_mesh_in_latlon(lamc_DP,phic_DP, stride=60,newfig=False)

         plt.annotate("Merc North",xy=(60,phi_n_Merc), xytext=(80, phi_n_Merc),arrowprops=dict(a
         plt.annotate("Merc South",xy=(60,phi_s_Merc), xytext=(80, phi_s_Merc),arrowprops=dict(a
         plt.annotate("SC",        xy=(60,lat0_SO),    xytext=(80, lat0_SO),   arrowprops=dict(a


         plt.pcolormesh(geolon,geolat,wet)
```

Out[21]: `<matplotlib.collections.QuadMesh at 0x2ba635f32f60>`



In [ ]: