Tripolar grid generation code: user guide and code overview

Alistair Sellar

July 10, 2007

1 Introduction

The NEMO ocean modelling framework uses tripolar grids for global models. In contrast to traditional latitude-longitude grids, which suffer from a singularity at the geographical north pole, tripolar grids have a single southern pole over Antarctica and two northern poles, generally over Russia and Canada, thus moving all grid singularities out of the ocean model domain. The motivation for using such a grid, as well as the merits of other solutions, are discussed in Madec and Imbard (1996).

Code to generate the tripolar 'ORCA' grids currently used in NEMO was written primarily by Gurvan Madec (LOCEAN-IPSL, Paris). Originally this code was designed to generate a grid with a single displaced north pole (Madec and Imbard, 1996), but in 1999 was modified to generate tripolar grids. In 2005, I added some code to generate higher meridional resolution in the tropics and converted from Fortran 77 to Fortran 90.

This document is designed to enable a new user to run the code and make changes to the main grid parameters without the need to understand the code itself. It is also intended to give an introduction to the code for anyone who needs to develop the it further.

2 The grid

The tripolar grid produced by this code consists of 3 parts. The northern part stretches from the geographical north pole to a given line of latitude in the northern hemisphere (around 20N for the standard ORCA grids). The 'zonal' gridlines, or 'parallels' of the northern grid are [confocal?] ellipses, evolving from a circle at the southern join, to more eccentric ellipses further north, and finally to a line joining the two north poles.

The 'southern' grid stretches from the join with the northern grid latitude down to the southern limit of the grid, generally at around 80S. This southern grid is a Mercator grid: the zonal spacing is regular, while the meridional spacing decreases polewards as the cosine of latitude to maintain isotropic grid boxes.

The tropical grid lies inside this southern part, between xS and xN, where x is generally between 15 and 20. Within the tropical grid, the zonal grid lines are identical to those of the external southern grid, while the meridional spacing is gradually decreased towards the equator to achieve higher equatorial resolution.

The 3 grids are matched where they join each other, so that the meridional and zonal spacings are continuous over the whole grid.

3 Compilation

The code is intended to compile under any compliant Fortran 90 compiler. It must be compiled with a flag that promotes reals to 8 bytes. It must be linked with the NetCDF F90 library.

4 Run

4.1 Parameters

The tropical stretching is stamped on top of the global grid once it has been calculated. The following parameters specify the underlying grid, which does not have tropical stretching. If tropical stretching is included, the final grid will have more rows than indicated here.

- `jpi`: This specifies the number of points in the zonal direction, and hence the zonal spacing of the grid. The grid is automatically isotropic outside of the northern part, so this also controls the meridional spacing [find out where]. For reasons which will become clear in section 5.2.1, `jpi` is actually the number of zonal points plus 1: *eg.* a 1-degree grid uses `jpi=361`.

- `jpeqt`: This specifies the number of rows between the geographic equator and the southern limit of the grid. `jpeqt` includes both the southern most row and the equatorial row. The meridional spacing in the southern hemisphere is controlled by `jpi`, so this parameter effectively sets the southern-most latitude of the grid.

- `jpeqn`: This specifies the number of rows between the geographical equator and the join to the northern grid. This effectively sets the latitude of this join. `jpeqn` includes the joining row, but not the equatorial row.

- `jpnord`: This specifies the number of rows in the (stretched) northern portion of the grid. Since the southern extent of this is effectively controlled by `jpi` and `jpeqn`, this controls the meridional spacing in the northern portion. As a rule of thumb, we have taken the value 218 used for ORCA05 (whose join is at around 20N) and scaled linearly. This parameter includes the joining row, so the total number of rows in the grid is `jpj = jpeqt+jpeqn+jpnord-1` (before tropical stretching is added).

- `rlampo`: This specifies the longitude of the second pole. The first longitude of the final grid will be `rlampo-180`, at least for `rphipo=90`.
- `rphipo`: As far as we have worked out, this sets the maximum latitude through which the line between the poles passes. We have found no reason to change this from 90.
- `rphi1`: The latitude of the first pole.
- `rphi2`: This controls the latitude of the second pole, which is given by `180-rphi2`.

At present the integer parameters, `jp*`, are set in `param.f90`, while the real parameters, `r*` are calculated at run-time in the routine `set_coefficients`. It may be that many of the real parameters could be set in `param.f90`.

Once the global grid has been calculated, it is a Mercator grid everywhere outside of the northern part. If tropical stretching is added this must not encroach into the northern part of the grid; otherwise there will be a mismatch in the southern hemisphere. Tropical stretching is set using the logical `l_trop_stretch` in module `param`.

The shape of the tropical stretching is controlled in the module `trop.f90`. The function `dphidj` specifies the desired meridional grid spacing, $\frac{\mathrm{d}\phi}{\mathrm{d}j}$, as a function of ϕ. The function currently used is

$$\frac{\mathrm{d}\phi}{\mathrm{d}j}(\phi) = R\left[\cos(\phi) - 0.5(1 - \frac{R_{min}}{R})\left(e^{-(\frac{c\phi}{a})^4} + e^{-(\frac{c\phi}{b})^4}\right)\right]. \tag{1}$$

In this function,

- R is the nominal resolution of the grid in degrees (`ppres` in the code),
- R_{min} is the minimum meridional resolution desired in the tropics (`ppresmin` in the code),
- $\cos(\phi)$ is the meridional spacing of the underlying mercator grid,
- a and b alter the shape of 'bi-gausian', controlling the flatness across the equator, the steepness of the sides and the rate at which the tropical stretching decays to the underlying grid (*ppa* and *ppb* in the code),
- c alters the 'width' of the function and is varied to match the tropical grid to the external grid (subroutine argument *pc* in the code). A starting value of $c = 1$ is used. The code tries to match the grids at the external grid point closest to the latitude specified with `pphi_join`.

The grid parameters `ppres`, `ppresmin` and `pphi_join` are set in the header of the module `trop`, while the function parameters `ppa` and `ppb` are set in the header of the routine `dphipj` inside that module. To alter the form of the function itself, see section **??**.

4.2 Outputs

The principal output of the grid generation code is the `coordinates.nc` NetCDF file which contains the grid details needed by NEMO. The variables in this file are firstly the geographical coordinates for each of the T, U, V and F grids (*eg.* glamt, gphit on the T grid), and secondly the zonal and meridional scale factors for each grid (*eg.* e1t, e2t for the T grid).

The scale factors are explained in detail in the NEMO manual, but put simply, if λ and ϕ are considered as continuous functions of continuous indices i and j, then the coordinates are the values of λ and ϕ at integer values of i and j and the scale factors are based on the first derivatives of λ and ϕ with respect to i and j, evaluated at these points. The scale factors are calculated in units of meters and are approximately equal to the arc length of each grid box.

Figure **??** shows the CDL representation of such a file. In addition to the coordinates and scale factors, the grid generation code also calculates the anisotropy of the grid ($= e_1/e_2$) for each of the 4 grids. They are stored in a single 3D array, of which the 3rd index specifies the grid (in the order T, U, V, F).

In addition to the main coordinates file, the code also produces two files which both contain the coordinates and scale factors for the north half-grid. These are the NetCDF file `coordinates_north.nc` (and the legacy binary file `hgrnth` if `l_write_binary` is set). The file `coordinates_north.nc` can be used to quickly generate a new global grid if no changes are required to the northern portion.

4.3 Faster running

This is one example of how the code can be speeded up if there is not a need to calculate all the grid details from scratch. To run using an existing `coordinates_north.nc` file, set `l_nth_calc` in `param.f90`.

The most useful efficiency is helpful when experimenting with the details of the northern grid and only the coordinates are of interest, not the scale factors. In this case, setting `l_ech_calc=.false.` skips the calculation of the scale factors, which speeds up the code by a factor of 5.

5 Code overview

5.1 In-line comments

The most extensive comments in the code are in French. In places I have added English comments to clarify the most tricky points. In general these are not attempts at translations of the existing comments, but rather an indication of what the code is doing, as gleaned from my understanding of the existing comments and from the code itself.

When reading the code the following explanation of terms may be useful:

- *grille* = grid
- The *terrestrial equator* is the row of the grid which lies on the true geographical equator.
- The *equator of the grid* is the line along which the (stretched) northern grid joins the rest of the grid. It is a line of constant latitude. It is to the north of the terrestrial equator.
- The *demi-grille nord* is half of the northern part of the grid, covering a longitude range of only 180°. It is twice the resolution of the final grid and is used in the calculation of the northern portion of the grid (see section **??**).
- *facteurs d echelle* = scale factors
- The grid lines which run from the south pole to either of the two north poles are referred to as meridiens of the grid and the grid lines orthogonal to these which ring the globe are referred to as parallels of the grid.
- *Controlles* are checks, eg. for errors or inconsistencies.

5.2 Grid generation method

As mentioned previously, the code computes each section of the grid in turn. The north section with the double poles is computed first, such that it will match with a mercator grid at its southern limit (around 20N for the standard ORCA grids). Next the 'southern' part is computed which covers all of the globe from the southern limit (around 78S for the ORCA grids) to the join with the northern part. Hence the southern part of the grid extends into the northern hemisphere. The two grids are joined together in a single array. Finally, the tropical part is calculated (if desired), and the relevent section of the existing grid is replaced by the new tropical rows.

5.2.1 The north half-grid

The north grid is computed for only half of the full longitude range, from one pole to the other, and the symmetry of the grid is used to deduce the remainder. All four sub-grids (T, U, V, F grids) are calculated at the same time by using a grid which is twice the resolution desired. The sub-grids are extracted later. The result of these two facts is that the north half-grid has twice the number of meridional points requested (`2 * jpnord`) and the same number of desired zonal points, plus one to include both the meridien running up through the first pole and that running through the second (*eg.* for ORCA1, `jpi=360+1`).

The method used to generate the north grid is very similar to that described in Madec and Imbard (1996). The difference is that in that paper, there is a single north pole over Siberia, so the grid parallels are non-concentric circles, whereas here the are two north poles over Russia and Canada, so the grid parallels are [confocal?] ellipses. The underlying method remains the same, so only a basic overview is given here. There is also a documentation paper written in french covering this part of the code (by Christophe Bagot, a copy of which I was given by Gurvan Madec).

Firstly, the grid parallels are defined by analytical functions which define the semi-major and semi-minor axes of the ellipses as a function of the continuous [meridional?] index j [possibly an independent index t?]. Some checks are performed which ensure that the ellipses will grow monotonically with this index. If the grid parameters are changed a lot, it may be that the new ellipses will fail these checks. For example, when we tried to move the southern limit of the north grid from 20N to 30N, these checks could only be passed by moving both poles to the same latitude of 55N. It could be that such problems may be avoided by playing with other parameters (such as `jpnord`) or by altering the functions themselves. Some guidance on the original choice of these functions is given in the Bagot documentation.

These ellipses are not explicitly computed. Instead the functions defining the elipses are used to define the gradient of the meridiens (which must be orthogonal to the ellipses) at any point within the domain of the grid. Thus the path of the meridiens from the grid equator to the poles can be found by numerical integration of this first order ordinary differential equation.

A leap-frog scheme is used to perform this integration. Each meridien is integrated using a very small step size (50 times finer than the main grid resolution). To avoid errors near the geographical north pole, this integration is performed in stereographic coordinates and translated to latitude and longitude coordinates afterwards.

The meridiens are integrated up from the 'grid equator' at each integer and half-integer value of the zonal index i (the integer values to compute the meridiens through the T and V points and the half-integer values for the U-F meridiens). Once the paths of the meridiens have been integrated, interpolation is used to find the values of λ and ϕ at integer and half-integer values of the meridional index j. Thus the coordinates of all the grid points in the northern grid have been computed.

In fact, for the calculation of the scale factors, it is a little more complicated than this. For e_1 we require gradients of λ and ϕ with respect to i. To compute this, 2 additional meridiens are computed to each side of the meridien itself (5 meridiens in all) and the gradients calculated using [4th order?] finite difference across the 5 points corresponding to the same value of j. For e_2, we require gradients with respect to j, so on each main meridien 2 additional points are found above and below each grid point to provide 5 points for a finite difference calculation.

[A figure of the above would be useful here.]

Most of the above is performed by the subroutine `nthcar`, which is called by `nthctl` for each meridien of the north half-grid.

5.2.2 The southern grid

The southern grid, which covers all of the domain not covered by the north grid, is computed very quickly. Spacing in the zonal direction is set by the number of zonal points (`jpi-1`). The meridional spacing is that of a mercator grid ($R\cos(\phi)$, where R is the zonal resolution).

This grid is calculated and joined with the north grid in the subroutine `mshsth`.

5.2.3 The tropical grid

The tropical grid is the final part to be calculated. This is performed by the subroutine `mshtrp`, using the information and subprograms in the module `trop`, along with the root-finding routines in the module `shoots`. This is only calculated if `l_trop_stretch` is true, otherwise the tropics are left as calculated by `msh_sth`.

In `trop`, the user specifes the function, $d\phi/dj$, which defines the desired meridional grid spacing as a function of ϕ, and also specifies the latitude, ϕ_j, at which this function is to be matched to the underlying grid. The code finds the grid point of the underlying grid which is closest to this latitude, and stores this latitude as ϕ_c. This is done by looking only at the grid points in the northern hemisphere: it is assumed that the grid is symmetric. Thus ϕ_j must not lie inside the north grid.

The routine `msh_trp` then takes the reciprocal of $d\phi/dj(\phi)$ to yield $dj/d\phi(\phi)$. A fine grid of ϕ is constructed from the equator up to ϕ_c, and $dj/d\phi(\phi)$ is integrated starting from the equator to yield $j(phi)$. In general, $j(\phi_c)$ will not be an integer and so the grids will not match. The code uses a Netwon-Raphson root-finder with repeated integrations to alter the function 'width factor' c until $j(\phi_c)$ is an integer and the grids match.

If a user wished to change the function $d\phi/dj(\phi)$ used for the tropical stretching, this should simply be a case of editing the function `dphidj` in module `trop`. It is important to retain some dependence on the 'width factor' `pc` to allow matching to the external grid.

The global arrays are deallocated and re-allocated inside `msh_trp` with an increased size to accomodate the extra rows in the tropics. The data for the south and north parts are stored in temporary arrays and read into the appropriate parts of the new global arrays.

5.2.4 Wrap points

Wrap points are added by `msh_sth` (which also performs the extraction of the full north sub-grids from the half-grid using symmetry). There are 2 wrap columns added to the [eastern edge?] of the grid to make the grid east-west cyclic. In addition to this there is a wrap row added to the top edge of the grid. This caters for the connection between the two sides of the top row across what is known as the north fold (the line connecting the two north poles).

In ORCA2, which was generated using an older version of this code, the north fold is a T-U row and so there are two V-F rows and one T-U row above this (since there are same number of rows on each grid and the V and F grids sit above the T and U grids) which act as wrap rows. Such grids correspond to the parameter setting `nperio=4` in NEMO. In this version of the code, the north fold is a V-F row, so there is one T-U row and one V-F row above this which act as wrap rows. ORCA1 and ORCA05 are examples of such grids and correspond to `nperio=6` in NEMO. Note that ORCA025 is derived from ORCA05 by quadrature and is of type `nperio=4`, like ORCA2.