Manual of the matlab scripts of `Roms_Interpolation`

Mathieu Dutout Sikirić

February 25, 2009

The set of scripts is concerned with interpolation for the ROMS model. Issues concerned are:

1. Interpolating from a roms grid to another roms grid. This is **R2R_Interpolation**.
2. Interpolating from ROMS to a set of ADCP or CTD locations, this **R2A_Interpolation**.
3. Interpolating from ROMS to the boundary of a nested grid.

1 Main features of the scripts

The main features of the existing script are:

1. Relative simplicity of use. No special requirements on the grids.
2. There is an initial computation of interpolation data and then the interpolation is done fast. If very fast speed is needed then there is a sparse matrix mode.
3. The interpolation is 2D + 1D, i.e. first interpolate on the horizontal then on the vertical.
4. No mexcdf files is used, which means that the scripts are quite slow at least when creating the initial array.

1.1 Interpolation method

For **2D array** (free surface, sea surface temperature, ...):

1. The interpolation at a point $p_{fine,\rho}$ is done linearly by selecting 4 points $p_{coarse,\rho,i}$ for 1‘$i \leq 4$. If no such quadruples can be found then $P_{fine,\rho}$ is outside the coarse grid and we cannot interpolate.
2. If some of $p_{coarse,\rho,i}$ are land then they are excluded.
3. If all points are excluded then we select the nearest wet one.

For **2Duv array** (u/v barotropic, u/v surface, ...):

1. The angles of the grids are interpolated linearly.
2. The same strategy as for 2D array is used.
3. No ρ points are used.

For **3D array** (temperature, salinity, ...):

1. The ROMS parametrization (originally from Song & Haidvogel) is used for finding the vertical positions in terms of θ_b, θ_s, ...
2. For a point $p_{fine,\rho}$ at a depth d_{fine} the 4 neighbors $p_{coarse,\rho,i}$ are determined. Points in the land are excluded.
3. The 3D array is interpolated at the coarse point at the depth d_{fine}. If such an interpolation is impossible, then the point is excluded.
4. If all points have been excluded then we interpolate to the nearest point.

For **3Duv array** (momentum, ...):

1. A combination of the method for 3D array and 2Duv array is used.

2 R2R_Interpolation use

There are two kinds of functions:

1. **InterpolSpMat_R2R_***: which uses sparse matrix that one has to compute in advance. The speed is very high but the problem is one of memory.
2. **InterpolMemEff_R2R_***: which uses simple iteration on the precomputed data. The speed is less than for sparse matrix but still reasonable.

In the following, we use the variable names **ZETA**, **TEMP**, **UBAR, VBAR**, and **U, V** because they are typical of the four classes (2D, 2D u/v, 3D, 3D u/v) of problems to be considered. We do not imply by that that the function do not work differently for salinity or any other field.

The array **AddiRecordInfo** contains the vertical discretization of the ROMS model for the two grids. So typically, one has:

```
AddiRecordInfo.BigThetaS=3;
AddiRecordInfo.BigThetaB=0.35;
AddiRecordInfo.Nbig=20;
AddiRecordInfo.Bighc=3;

AddiRecordInfo.SmaThetaS=3;
AddiRecordInfo.SmaThetaB=0.35;
AddiRecordInfo.Nsma=20;
AddiRecordInfo.Smahc=3;
```

If one uses the Memory efficient functions, then one has an initial call to get the array used thereafter:

```
TotalArray=InterpolGetTotalArray_R2R_fields(BigGridFile, SmallGridFile);
```

This call is then followed by interpolation for the various model fields:

```
ZETAsma=InterpolMemEff_R2R_2Dfield(TotalArray, ZETAbig);
TEMPsma=InterpolMemEff_R2R_3Dfield(TotalArray, AddiRecordInfo, TEMPbig);
[UBARsma, VBARsma]=InterpolMemEff_R2R_2Duvfield(TotalArray, UBARbig, VBARbig);
[Usma, Vsma]=InterpolMemEff_R2R_3Duvfield(...
   TotalArray, AddiRecordInfo, Ubig, Vbig);
```

If one uses the sparse matrix formulation, then one has to choose which array is interpolated. The choice has to be made for 2D array, 2D u/v array, 3D, 3D u/v. Assuming one uses sparse matrix for everything, the computation begins in the same way.

```
TotalArray=InterpolGetTotalArray_R2R_fields(BigGridFile, SmallGridFile);
```

then the sparse matrix are computed:

```
ArrayBigSma2D=InterpolGetSpMat_R2R_2Dfield(TotalArray);
ArrayBigSma2Duv=InterpolGetSpMat_R2R_2Duvfield(TotalArray);
ArrayBigSma3D=InterpolGetSpMat_R2R_3Dfield(TotalArray, AddiRecordInfo);
ArrayBigSma3Duv=InterpolGetSpMat_R2R_3Duvfield(TotalArray, AddiRecordInfo);
```

and the interpolation is done:

```
ZETAsma=InterpolSpMat_R2R_2Dfield(ArrayBigSma2D, ZETAbig);
TEMPsma=InterpolSpMat_R2R_3Dfield(ArrayBigSma3D, TEMPbig);
[UBARsma, VBARsma_sm]=InterpolSpMat_R2R_2Duvfield(...
    ArrayBigSma2Duv, UBARbig, VBARbig);
[Usma, Vsma]=InterpolSpMat_R2R_3Duvfield(...
    ArrayBigSma3Duv, Ubig, Vbig);
```

The example **CreateInitial.m** is an illustration of the use of those functions. If one uses the sparse matrix formulation, then we can achive high speed since only array operations are done but we are limited by memory.

The memory efficient formulations are to be prefered for normal use and they are reasonable in size and speed. We recommend to use the sparse matrix only if speed is really the critical point of the computation. This can be the case for 2D computation of plots.

What `R2R_Interpolation` is not

1. It does not deal with the problem of dynamic adjustment. After the interpolation from the coarse grid, you can expect artificial currents in the fine grid.

2. It is not a substitute for spinning-up the model.

3. It is not a general interpolation program, it applies only to the ROMS model setting.

4. It does not use other interpolation methods like nearest neighbors or bicubic method, though one may want to do just that.

3 R2A_Interpolation functions

There are two differences between the R2R functions and the R2A functions:

- Roms uses **rho**, **u** and **v** points, which means that the temperature, u component of speed and v component of speed are at different points. Typically, an ADCP gives you the u and v components at the same point.

- The vertical spacing of an ADCP or CTD is of course not the same as the ROMS vertical discretization and so as to be passed as argument.

ADCP and CTD are different physical instrument but for interpolation they behave in the same way. So, in the following we use the word ADCP, which is supposed to mean ADCP or CTD.

The first call to the function is

```
TotalArray=ADCP_R2A_GetTotalArray(...
    GridFile, AddiRecordVertDisc, ...
    ListLonADCP, ListLatADCP, ADCPverticalLevels)
```

with

1. **GridFile** the ROMS grid of the model.

2. **AddiRecordVertDisc** is the description of the vertical discretization of the ROMS grid. For example:

   ```
   AddiRecordVertDisc.ThetaS=3;
   AddiRecordVertDisc.ThetaB=0.35;
   AddiRecordVertDisc.N=20;
   AddiRecordVertDisc.hc=3;
   ```

3. **ListLonADCP**, **ListLatADCP** are the longitude and latitude in degrees in a 1D array $(N, 1)$, with N the number of ADCP.

4. **ADCPverticalLevels** is a (N, m) matrix with m the maximum number of vertical levels. In it you should put the depth of the ADCP. The depth are negative numbers. If an ADCP has less than m vertical levels, then put NaN for the unused entry of the matrix.

Then the interpolation is done in the simple way:

```
ZETAadcp=ADCP_R2A_Interpolation2Dfield(TotalArray, ZETAroms);
[UVbar_east, UVbar_north]=ADCP_R2A_Interpolation2Duvfield(...
   TotalArray, UBARroms, VBARroms);
TEMPadcp=ADCP_R2A_Interpolation3Dfield(TotalArray, TEMProms);
[UV_east, UV_north]=ADCP_R2A_Interpolation3Duvfield(...
   TotalArray, Uroms, Vroms);
```

For u/v field the interpolation is always to the eastern and northern components. For $3d$ array the output `TEMPadcp`, `UV_east`, `UV_north` are (N, m) matrices with NaN entries when there was a NaN in the matrix ADCPverticalLevels.

4 Interpolation to the boundary

We consider here 1way nesting, running two grid, one of coarse resolution and a small one of fine resolution and using the big grid to interpolate to the small grid.

The method requires the creation of boundary file for the small grid. Two methods are used classically: station file and history file. We provide methods for both situation. Note that all codes creates boundary for zeta, u, v, ubar, vbar, temp, salt. If only a subset of those files are needed then you will need to hack it by yourself. Also, we do not provide functionalities for different time discretizations in the boundary: the timings are the same as in the station file or the history file.

Below the array **AddiRecordInfo** described the needed informations on the vertical discretization and which boundary of the small grid are opened $(= 1)$ or closed $(= 0)$. The East, West, South, North refer to the ROMS conventions and correspond to the classic definition if the angle of the grid is 0.

```
AddiRecordInfo.BigThetaS=3;
AddiRecordInfo.BigThetaB=0.35;
AddiRecordInfo.Nbig=20;
AddiRecordInfo.Bighc=3;

AddiRecordInfo.SmaThetaS=3;
AddiRecordInfo.SmaThetaB=0.35;
AddiRecordInfo.Nsma=20;
AddiRecordInfo.Smahc=3;

AddiRecordInfo.DoEast=0;
AddiRecordInfo.DoWest=1;
AddiRecordInfo.DoNorth=1;
AddiRecordInfo.DoSouth=1;
```

Station file method: The first step is the creation of the station file input for the roms model.

```
TheRecord=OW_RecordBoundaryPoints(SmaGridFile);
OW_OutputStations(...
    StationFile, ...
    TheRecord.LonCoord, TheRecord.LatCoord, ...
    TheRecord.ListComments);
```

The boundary itself is created by the following command.

```
NEST_CreateBoundaryFromStation(...
    StationFileName, TheOffset, ...
    SmallGridFile, AddiRecordInfo, ...
    BeginTimeSec, EndTimeSec, TheBoundaryFile);
```

with

- **StationFileName** is the netcdf file of the stations.
- **TheOffset=0** is the offset to apply in the statio file.
- **SmallGridFile** is the netcdf file of the nested grid.
- **BeginTimeSec, EndTimeSec** are the beginning and ending time of the time frame considered.
- **TheBoundaryFile** is the output boundary file to be used by the roms model run on the nested grid.

Stations can be used for multiple purposes. In that case, the Lon, Lat array of **OW_OutputStations** can be modified and the variable **TheOffset** has to be adjusted accordingly.

One defect of the station method is that sometimes the file created by ROMS contains some non-identified points for no real reason other than wrong arithmetic. In that case, the program **NEST_CreateBoundaryFromStation** takes the nearest valid station point, not the greatest strategy. Another defect is that you need to have your station file correct before running, while the history file allows you to do after the run.

History file method: The first step is the creation of the interpolating array.

```
NestingTotalArray=NEST_CreateNestingTotalArray(...
    BigGridFile, SmallGridFile, AddiRecordInfo, UseSpMat);
```

The variable **BigGridFile**, **SmallGridFile** and **AddiRecordInfo** are clear. **UseSpMat** trigger the use of sparse matrix for the interpolation something, which we recommend since the boundary is small compared to the rest of the domain and many record have to be treated a priori.

The creation of the boundary file is then done by

```
NEST_CreateBoundaryFromHistory(...
    PrefixHis, NestingTotalArray, ...
    BeginTimeSec, EndTimeSec, TheBoundaryFile);
```

with **PrefixHis** the prefix to the history file (for example 'Output/ocean_his_0001.nc' has prefix 'Output/ocean_his_'). **BeginTimeSec** and **EndTimeSec** are the beginning and endtime chosen.

Creation of grid of nested domain: For that you need the package `LP_Bathymetry` for doing smoothing with linear programming while having some point fixed.

5 Availability

The source of the program is available from http://www.liga.ens.fr/~dutour/Roms_Interpolation/index.html