Regridding Data: A Package to Interpolate, Extrapolate, and Fitt a Curve

M. Abouali and Jose E. Castillo

March 16, 2014

Publication Number: CSRCR2014-02

Computational Science & Engineering Faculty and Students Research Articles

Database Powered by the Computational Science Research Center Computing Group & Visualization Lab

COMPUTATIONAL SCIENCE & ENGINEERING

SAN DIEGO STATE UNIVERSITY

Computational Science Research Center
College of Sciences
5500 Campanile Drive
San Diego, CA 92182-1245
(619) 594-3430

Regridding Data: A package to interpolate, extrapolate, and Fitt a Curve

M. Abouali *[1] and Jose E. Castillo[2]

[1]Ph.D. Candidate at San Diego State University
[2]Director, Computational Science Research Center, San Diego State University

March 16, 2014

1 Introduction

The idea of representing a data field via parameterization of a function is not new. In fact, if the functions or basis are sine and cosine, the procedure is known as the Fourier transformation. Many meshless methods make frequent use of radial basis functions (RBF) such as the Gaussian function, and polynomial functions are also used frequently. If the degree of the polynomial is equal to one, then the procedure is called a linear/bilinear/trilinear interpolation in a 1D/2D/3D domain. In geophysical fluid models and many other numerical models, the data is often stored in one location, but its value is needed at multiple locations. For example, in the A-grid approach, all variables, including the velocity and pressure, are stored at the cell centers; however, in order to calculate the fluxes, they are also needed at the cell faces. Therefore, some type of interpolation must be performed. Depending on the curvature of the boundary in curvilinear grids, one might need to use extrapolation. Another approach is to fit a surface to the data points and evaluate it at the desired locations. Hereafter, all of these methods will be used interchangeably (although mathematically they are not the same procedure), or they may simply referred to as "regridding" or "reprojecting".

The choice of interpolating functions can also affect the stability of the numerical scheme in use. As mentioned above, there are numerous choices available; however, these procedures are quite time-consuming (e.g., in fluid models, the same procedure has to be performed millions of times). In some methods, including Kriging's, the values of the data must be known before beginning the method. However, in most approaches, the most time-consuming aspect of the method depends on the locations of the points, and not on the value of the data. (The polynomial approach is one such method.) Programs such as SCRIP scr (n.d.), NCL ncl (n.d.), and ESMF esm (n.d.), pre-calculate the "interpolation weights". Once the weights are known, the re-projecting procedure would be reduced to a single matrix multiplication. While ESMF and NCL allow for scattered data interpolation, SCRIP only works on structured data. In addition, all of these packages support only interpolation, and not extrapolation or a curve-fitting approach.

Due to their importance, a MATLAB package was developed to calculate these interpolation weights. They can now be interpolated, extrapolated, or fit into a local surface of the data. This package can be used for structured, non-structured, and even scattered grids. The following section describes the equations, how the matrix is formed, and explains important results.

2 Governing Equations

Let us assume f_j is a measured or computed parameter, such as pressure or temperature, at locations denoted by (x_j, y_j) for 2D distribution and (x_j, y_j, z_j) for 3D distribution, where $j \in [1, n]$ and n is the number of points. These points, with a known value, are called source points or the source grid. Note that there is no assumption made as to how these points are distributed; therefore, the source grid can be structured, unstructured, or even scattered. In polynomial presentation, the aim is to describe the spatial variation of the parameter with a polynomial functions; This function can be written as follows:

*maboualiedu@gmail.com

$$f = f(x,y) = \sum_{p_1=0}^{p_d} \sum_{p_2=0}^{p_d} a_{p_1,p_2} x^{p_1} y^{p_2}, \tag{1}$$

for 2D grids and in 3D grids as:

$$f = f(x,y,z) = \sum_{p_1=0}^{p_d} \sum_{p_2=0}^{p_d} \sum_{p_3=0}^{p_d} a_{p_1,p_2,p_3} x^{p_1} y^{p_2} z^{p_3}, \tag{2}$$

where p_d is the polynomial degree, and a_{p_1,p_2} or a_{p_1,p_2,p_3} are the polynomial coefficients that must be determined. These coefficients are also called free parameters. There are various well-known approaches to determine these coefficients, including Lagrange, Newton, and Barycentric interpolations. Once these coefficients are determined, one can replace (x_i^*, y_i^*, z_i^*) at the point, where there is no measurement or computed value, in order to estimate a value for the parameter, i.e. f_i^*, where $i \in [1, m]$ and m is the number of points. These points, whose values are unknown, are called the destination grid.

In 2D there are $n_s = (p_d + 1)^2$ free parameters and in 3D there are $n_s = (p_d + 1)^3$; by substituting the value and location of the points on the source grid we can form a system of equations to determine the free parameters and uniquely define the polynomial. If exactly n_s points from the source grid are chosen to form the system of the equation, the resulting polynomial passes exactly through the points on the source grid. Therefore, it would either be interpolation (once the destination point falls within the convex hull of the source grid) or extrapolation (once the destination point falls outside). If more points than the minimum required are used we are actually fitting a curve to the source grid.

If all points at the source grid are used to determine the polynomial coefficients, the resulting polynomial is considered "global"; however, if only a subset of points on the source grid are used to form the polynomial for each point on the destination grid, then the polynomial is termed "local."

This entire procedure can be thought of as a transform function, which projects/transforms/regrids data from the source grid onto the destination grid. Note that no assumptions on any of the grids are made, so they can be structured, unstructured, or even scattered. The software package introduced here uses the local polynomial approach by inverting the resulting Vandermonde matrices. Vandermonde matrices are known to be poorly conditioned for large polynomial degrees. Another issue regarding high p_d is the Runge phenomenon and the oscillation arising at the boundaries. Both of these issues requires the user to choose p_d, carefully.

This entire procedure can be simplified as a sparse matrix, $P_{m\times n}$, which can then be used to re-project data from the source grid onto the destination grid with a single matrix multiplication, i.e.:

$$f^*_{m\times 1} = P_{m\times n} f_{n\times 1}, \tag{3}$$

where $f_{n\times 1} = [f_1, f_2, \cdots, f_n]^T$ and $f^*_{m\times 1} = [f_1^*, f_2^*, \cdots, f_m^*]^T$. The following outlines how this matrix was constructed; for the simplicity's sake, the procedure is described for 2D source grids with $p_d = 1$.

Let us assume s_i points were used on the source grid to project data onto the i^{th} point on the destination grid. Note that s_i could be a different value for each point; however, $s_i \geqslant n_s$. Let us also assume $S_i = [f_{j1}, f_{j2}, \cdots, f_{js_i}]^T$ are the s_i closest points to the i^{th} point on the destination grid. We then determine

$$f_i^* = P_i \begin{pmatrix} f_{j1} \\ f_{j2} \\ \vdots \\ f_{js_i} \end{pmatrix}, \tag{4}$$

where P_i is a $1 \times s_i$ matrix computed as follows:

$$P_i = A^* \left(A^T A\right)^{-1} A^T, \tag{5}$$

and:

$$A_{s_i\times n_s} = \begin{pmatrix} 1 & x_{j1} & y_{j1} & x_{j1}y_{j1} \\ 1 & x_{j2} & y_{j2} & x_{j2}y_{j2} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_{js_i} & y_{js_i} & x_{js_i}y_{js_i} \end{pmatrix}, \tag{6}$$

and:

$$A^*_{1\times n_s} = \begin{pmatrix} 1 & x^*_i & y^*_i & x^*_i y^*_i \end{pmatrix}. \tag{7}$$

P_i must be calculated for each point on the destination grid. Each element of P_i, i.e. $< P_i >_{1,c}$ is then substituted at $< P >_{i,jc}$, or:

$$\begin{aligned} < P_i >_{1,1} &\rightarrow < P >_{i,j1}, \\ < P_i >_{1,2} &\rightarrow < P >_{i,j2}, \\ &\vdots \\ < P_i >_{1,s_i} &\rightarrow < P >_{i,js_i}, \end{aligned} \tag{8}$$

and all other elements of $P_{m\times n}$ are set to zero. Note that P is completely independent of the data and is only a function of the locations. This means that, as long as the points on both the destination and source grids have not moved relative to each other, P will not change. Again, this movement is relative, i.e. points on both grids can move, but not relative to each other. Hence, the grids can be translated or even rotated, as long as the same rotation or translation is applied to both grids. Under certain conditions, the grids can alse be scaled, while continuing to use the same weights.

In many cases, the fact that P stays constant can be used to reduce computational need. In our case, i.e., fluid mechanics, the grid cells stay constant throughout the entire simulation; with the only changes being in the values of the velocity, pressure, and other variables. Therefore, we can calculate P once, then reuse it as many time as needed at a cost of a single, sparse matrix multiplication. Although this procedure was explained for polynomials, the author has successfully applied the same concept to RBF, the Gradient plus Inverse Distance Squared (GIDS) method, and the Inverse Distance Weighted (IDW) method (paper in preparation).

3 MATLAB Command

The command to create a 2D projector is as follows:

```
P=ConstructProjector2D(xs,ys,xd,yd,nPoly,nInterp),
```

where:

- **xs** and **ys** are the source grid points in the physical domain,
- **xd** and **yd** are the destination grid points in the physical domain,
- **nPoly** defines the degree of the polynomial to use, and
- **nInterp** determines how many points from the source grid must be used to determine the free parameters.

If nInterp is set to n_s, i.e. the minimum required, the resulting matrix, i.e. P would perform true interpolation or extrapolation. If nInterp is set to a larger number, for each point on the destination grid, the program finds nInterp-closest points on the source grid, fits a surface with a degree given by nPoly and calculates the weights. In this package the points are selected only based on the distance and closeness; however, we can harness a more robust approach in selecting the points. As long as this selection is not depending on the data, the same procedure can be used.

The command for 3D grids is the same as for 2D Grids:

```
P=ConstructProjector3D(xs,ys,zs,xd,yd,zd,nPoly,nInterp)
```

where **zs** and **zd** are the third dimension coordinates for points on both the source grid and the destination grid. In the geosciences, inverse distance weighted (IDW) is very commonly used; the IDW can be formulated as:

$$f^* = \frac{\Sigma \frac{1}{d_i^n} f_i}{\Sigma \frac{1}{d_i^n}} \tag{9}$$

Most of the computation time in IDW is spent locating the nInterp closest points. Using a similar approach, the IDW can also be simplified into a single sparse matrix. Since IDW is very common in the geosciences, a similar function was developed for IDW interpolation for in both 2D and 3D.

Once P is calculated, we can re-grid data from the source grid onto the destination grid, as follows:

$$f^*_{m\times 1} = P_{m\times n} f_{n\times n_f}, \tag{10}$$

where n_f is the number of data fields; i.e., if there are multiple data fields, we can either re-grid them one by one, or each data field can be represented as a column and all re-gridded at once.

4 Results

To test the code a structured grid was generated where $(x, y) \in [0, 2\pi] \times [0, 2\pi]$. Four data fields were developed, as follows:

$$F_1(x, y) = \sin\left(\sqrt{x^2 + y^2}\right), \tag{11}$$

$$F_2(x, y) = \sin(x)\cos(y), \tag{12}$$

$$F_3(x, y) = e^{-\sqrt{x^2+y^2}}, \tag{13}$$

$$F_4(x, y) = e^{-\sqrt{(x-x_0)^2+(y-y_0)^2}}. \tag{14}$$

Using the developed MATLAB package, a Projector, P, was computed, which interpolates from the nodal grid to the cell centers. P was computed only once, and then the same matrix was applied to all four functions. The results are shown in Figure 1. To test the package for scattered grids, two sets of random data were generated, Figure 2. The projector P was constructed only once and the above mentioned functions were evaluated on source grid points to produce a data field. Using the P generated above, the data field was projected to the destination grid and its accuracy was computed. The accuracy is shown in Figure 3. Note that since both the destination and source grids are randomly generated, each execution of the code will generate a different graph.

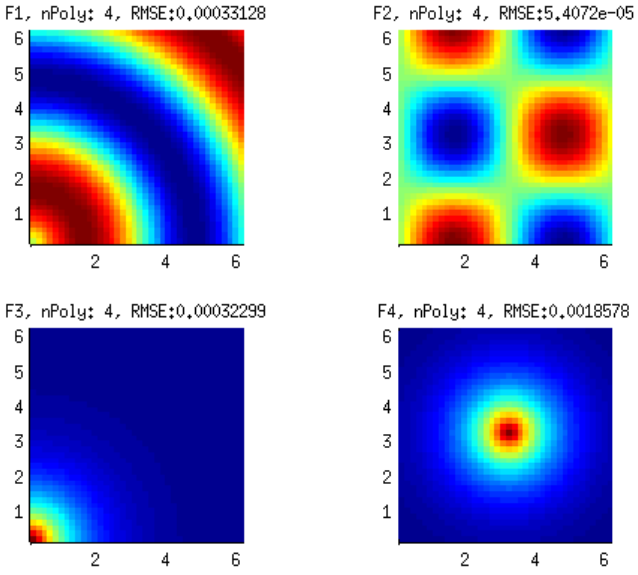

Figure 1: Interpolating from cell corners to cell centers.

The effect of nInterp is interesting. By choosing an nInterp bigger than n_s, we can actually fitt a local surface on the source grid and evaluate the resulting function on the destination grid to get a value. This means that,

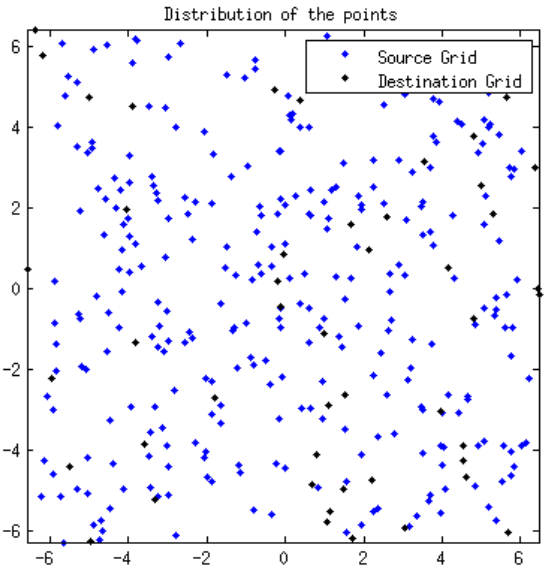

Figure 2: Scattered points.

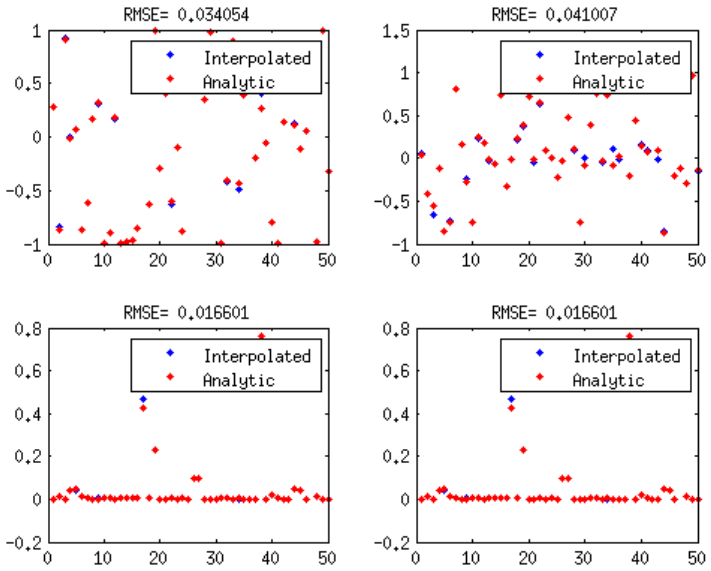

Figure 3: Interpolation results for scattered points.

unlike the interpolation, the function representing the surface does not necessarily match the values of the source grid. To check the effect of this approach on the error, a series of tests were performed. Two scattered grids were randomly generated; the degree of the polynomial was set to 4, i.e. $p_d = 4$; and multiple projector, P, was generated using different nInterps ranging from $n_s = 25$ to 45. Figure 4 shows how the root mean squared error (RMSE) changes with nInterp. The grid is generated randomly each time the code is executed; and, depending on how the points in the source and destination grids are distributed, different errors will be achieved. However, in all cases, RMSE shows the same behavior relative to nInterp. As shown in the graph, the RMSE is decreased by increasing the nInterp. This behavior was consistent in all the executations that we tried. We have shown that how the accuracy of the numerical solution to Poisson's equation changes by changing the nInterp in a separate publication (submitted, under review).

Once nInterp is set to the minimum required value, i.e. n_s, the system is forced to have zero errors at the source grids, i.e. true interpolation. Therefore, all errors are forced to be distributed only on the destination grid points. However, by setting nInterp to a number larger that of the minimum, we can relax the distribution of the error; thereby, compensating for some of the errors on the source grid. This explanation has been confirmed by subsequent tests.

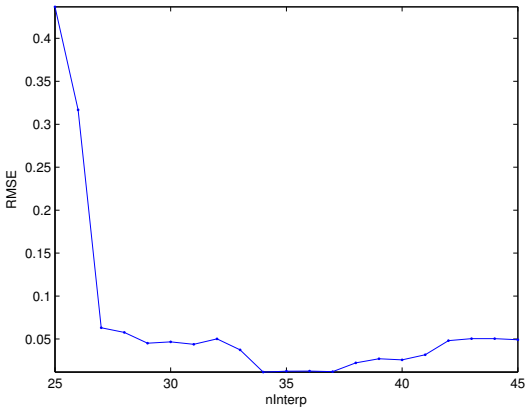

Figure 4: Effect of nInterp on interpolation accuracy.

5 Software Availability

This MATLAB package is available for download at:

http://www.mathworks.com/matlabcentral/fileexchange/41669-interpolantextrapolant-2d3d-data

We recommend beginning with one of the test functions to get acquainted with the code.

References

Earth System Modelling Framework (ESFM), http://www.earthsystemmodeling.org/.

The NCAR Command Language (Version 6.1.2) [Software]. (2013). Boulder, Colorado: UCAR/NCAR/CISL/VETS. http://dx.doi.org/10.5065/D6WD3XH5.

Spherical Coordinate Remapping and Interpolation Package, http://climate.lanl.gov/Software/SCRIP/index.shtml.