AGU PUBLICATIONS

Geochemistry, Geophysics, Geosystems

TECHNICAL REPORTS: METHODS

10.1002/2016GC006723

The GMT/MATLAB Toolbox

Paul Wessel[1] **and Joaquim F. Luis**[2]

[1]Department of Geology and Geophysics, University of Hawaii at Mānoa, Honolulu, Hawaii, USA, [2]Universidade do Algarve, IDL, Faro, Portugal

Key Points:
- Toolbox allows complete access to all GMT modules from MATLAB or Octave
- Improves interoperability between GMT and MATLAB/Octave
- Encourages development of user-friendly front ends to GMT

Correspondence to:
P. Wessel,
pwessel@hawaii.edu

Citation:
Wessel, P., and J. F. Luis (2017), The GMT/MATLAB Toolbox, *Geochem. Geophys. Geosyst.*, *18*, 811–823, doi:10.1002/2016GC006723.

Received 8 NOV 2016
Accepted 18 JAN 2017
Accepted article online 28 JAN 2017
Published online 28 FEB 2017

Abstract The GMT/MATLAB toolbox is a basic interface between MATLAB® (or Octave) and GMT, the Generic Mapping Tools, which allows MATLAB users full access to all GMT modules. Data may be passed between the two programs using intermediate MATLAB structures that organize the metadata needed; these are produced when GMT modules are run. In addition, standard MATLAB matrix data can be used directly as input to GMT modules. The toolbox improves interoperability between two widely used tools in the geosciences and extends the capability of both tools: GMT gains access to the powerful computational capabilities of MATLAB while the latter gains the ability to access specialized gridding algorithms and can produce publication-quality PostScript-based illustrations. The toolbox is available on all platforms and may be downloaded from the GMT website.

1. Introduction

The Generic Mapping Tools (GMT; gmt.soest.hawaii.edu) is a widely used set of software for analyzing and displaying geoscience data [*Wessel and Smith*, 1991, 1995, 1998]. Its power to analyze and process data and produce publication-quality graphics has made it one of several standard processing toolsets used by a large segment of the Earth and Ocean Sciences communities. GMT's strengths lie in superior publication-quality vector graphics (from page-size to wall-size), geodetic-quality map projections, robust data processing algorithms scalable to enormous data sets and grid sizes (e.g., making global bathymetry maps from 440 million randomly distributed soundings constraining grids with sizes up to 43,200 by 86,400, Olson *et al*. [2014]), its ability to run under all operating systems and being Open Source and freely available under a flexible license (Lesser GNU Public License). The GMT toolbox offers ∼140 modules sharing a common set of command options, file structures, and documentation. GMT modules are designed as UNIX filters, i.e., they accept input and write output, which allows users to write shell scripts where one module's output becomes another module's input, creating highly customized GMT workflows.

Since its initiation, GMT has been a UNIX and DOS command-line tool set and greatly benefitted from scripting. However, the release of GMT 5 [Wessel *et al*., 2013] introduced three key changes: (1) There is now a fully documented C Application Program Interface (API) for building new modules and libraries, containing basic functionality for handling GMT data objects (i.e., the input/output of data), manipulating module options, reporting errors and warnings, and accessing any of ∼140 modules via a flexible GMT_Call_Module function. Unlike previous GMT versions, in GMT 5 these modules are no longer stand-alone programs but have been implemented as high-level API functions; (2) The "namespace pollution," i.e., how to manage lots of different program executables in a standard installation directory (such as/usr/bin) when there might be many other tools with exactly the same names, have been eliminated. Since there are already other software packages that distribute programs called "surface" or "triangulate," this means they are competing for installation in the same directory as gmt via standard software package managers. By building just a single executable called **gmt** (that can access all modules) and requiring users to run "gmt triangulate" that conflict is now avoided, while a few shell functions implement backwards compatibility with older GMT 4 scripts; (3) The notions of input sources and output destinations have been generalized. In addition to the familiar mechanism of passing file names or using standard input/output, developers using the API can specify sources and destinations in several different ways, including memory locations, file pointers to open files or standard streams, and file descriptors. The GMT modules themselves are unaware of these distinctions as this flexibility is implemented in the API input/output layer.

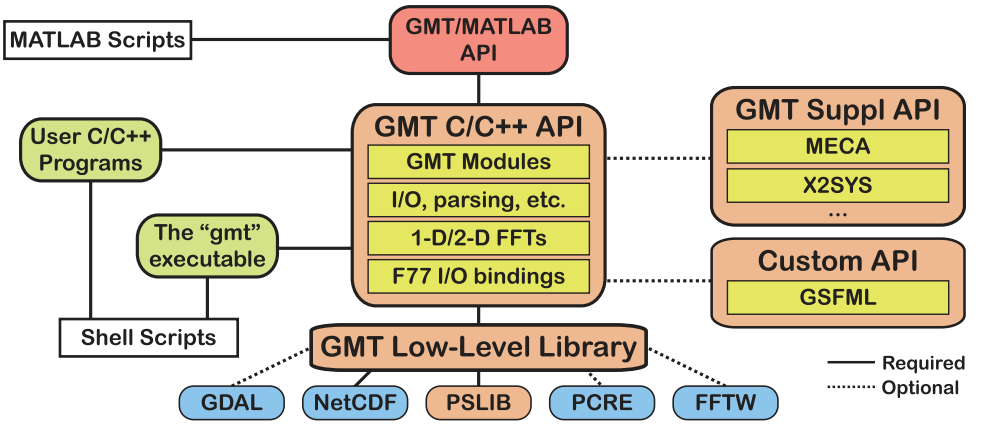

Figure 1. Conceptual block diagram of GMT dependencies. In GMT5, the high-level functionality resides in the API and any module may be called via the single gmt executable. Supplemental and custom APIs may also be accessed this way. The GMT/MATLAB toolbox allows direct access to all core, supplemental, and custom GMT modules, similar to how UNIX shell scripts access modules via the gmt executable (modified from Wessel *et al*. [2013]).

Consequently, the new GMT API has made it possible for developers to build additional modules or external APIs on top of the core C API, allowing for rapid development of new and complex functionality, including discipline-specific processing not presently available in the main GMT toolbox [Wessel *et al*., 2015]. Here we report on the development of a Foreign Function Interface (FFI) between the GMT API and external programs via the C programming language. While this FFI is known to work with other languages (an experimental Julia—http://julialang.org - is also being developed and work on a Python interface will start soon), we present here the GMT/MATLAB toolbox (http://gmt.soest.hawaii.edu/projects/gmt-matlab-octave-api/wiki). This toolbox represents a layer (an API) between the GMT API and the MATLAB® and Octave environments and allows full access to GMT and its modules. We will discuss the passing of data, show a series of examples of the symbiotic relationships of these two tools in producing elaborate results, present installation procedures for Windows, OS X, and Linux, and discuss any limitations in the toolbox relative to the standalone command-line GMT version.

2. GMT/MATLAB Overview

The intertwined use of GMT and MATLAB® encapsulates the working environments of major groups of scientists, especially those operating within the Earth and Ocean disciplines. It is important to note that while GMT 4 offered basic support for MATLAB going back more than a decade, this limited support only included the ability to read and write GMT grids from the MATLAB command line; no access was provided to the GMT modules themselves except via cumbersome system calls and separate input/output processing.

Both scientists and engineers rely on MATLAB for technical work. As shown in Figure 1, GMT can now be accessed from MATLAB via a MEX (MATLAB Executable) function called **gmt**. The gmt MEX function provides access to all of GMT's modules as well as import, formatting, and export of GMT data objects. Internally, the GMT/MATLAB C API defines six high-level data structures that handle input and output of data via GMT modules. These are data tables (representing one or more sets of points, lines, or polygons), grids (2-D equidistant data matrices), raster images (with 1–4 color bands), raw PostScript code, text tables (free-form text/data mixed records), and color palette tables (i.e., color maps). Correspondingly, we have defined six data structures that we use at the interface between GMT and MATLAB via the **gmt** MEX function. The GMT/MATLAB API is responsible for translating between the GMT structures and native MATLAB objects. The six MATLAB objects (Table 1) are:

Segments. GMT considers point, line, and polygon data to be organized in one or more segments in a data table. Modules that return segments use a native MATLAB segment structure that holds the segment data, which may be either numerical, text, or both; it also holds a segment header string which GMT uses to pass metadata. Thus, GMT modules returning segments will typically produce arrays of segments and you may pass these to any other module expecting points, lines, or polygons or use them directly in MATLAB. Since a

Table 1. The GMT/MATLAB Data Structures

(a) Data/Text Segment Structure and its Attributes	
data	Matrix with segment data
text	Cell array with any text after data coordinates
header	String with segment header
comment	Cell array with any data set comments [empty after first segment]
proj4	Projection string in PROJ4 syntax (optional)
wkt	Projection string in WKT syntax (optional)
(b) Grid Structure and its Attributes	
z	Grid matrix
x	Vector with *x* coordinates
y	Vector with *y* coordinates
range	Vector with range [x_min x_max y_min y_max z_min z_max]
inc	Vector with grid spacing [x_inc y_inc]
registration	Registration type: 0 = Grid registration; 1 = Pixel registration
nodata	The value used to indicate "no data"
title	Title (optional)
comment	Comment (optional)
command	Command used to create the grid (optional)
datatype	Data type: "float" or "double"
x_unit	Units of *x* axis (optional)
y_unit	Units of *y* axis (optional)
z_unit	Units of *z* axis (optional)
layout	A three chars code describing the memory layout (optional)
proj4	Projection string in PROJ4 syntax (optional)
wkt	Projection string in WKT syntax (optional)
(c) Image Structure and its Attributes	
image	Image matrix
X	Vector with *x* coordinates
y	Vector with *y* coordinates
range	Vector with range [x_min x_max y_min y_max z_min z_max]
inc	Vector with grid spacing [x_inc y_inc]
registration	Registration type: 0 = Grid registration; 1 = Pixel registration
nodata	The value used to indicate "no data"
title	Title (optional)
comment	Comment (optional)
command	Command used to create the image (optional)
datatype	datatype: "uint8" or "int8"
x_unit	Units of *x* axis (optional)
y_unit	Units of *y* axis (optional)
z_unit	Units of *z* axis (optional)
colormap	Color palette structure
alpha	Array with alpha (transparency) values (empty if no alpha)
layout	A four chars code describing the memory layout
proj4	Projection string in PROJ4 syntax (optional)
wkt	Projection string in WKT syntax (optional)
(d) Color Palette Table Structure and its Attributes	
colormap	Matrix with R/G/B values for all color intervals
alpha	Vector with alpha (transparency) values
range	Matrix with [z_low z_high] for each color interval
minmax	Vector with [z_min z_max] for entire range
bfn	Matrix with R/G/B for back-, fore-ground, and NaNs
depth	Color depth as 1 (black/white), 8 (gray) or 24 (colors)
hinge	Hinge separating two parts of the CPT [NaN means no hinge]
cpt	Full GMT cpt array for communicating with GMT
model	Color model used: "rgb," "hsv," or "cmyk"
comment	Cell array with any data set comments [empty after first segment]
(e) PostScript Structure and its Attributes	
postscript	Text array with all PostScript code
length	Number of bytes in the string
mode	Status mode: 1 = Has header, 2 = Has trailer. 3 = Has both.
comment	Comment (optional)

matrix is the fundamental data type in MATLAB you can also pass any data type matrix directly to GMT modules as well. Consequently, it is very easy to pass data from MATLAB into GMT modules that process data tables as well as to receive data segments from GMT modules that process and produce data tables as output. The toolbox also provides a function for concatenating segments into a single segment.

Grids. Many tools consider equidistant grids a particular data type and numerous file formats exist for saving such data. Because GMT relies on GDAL (Geospatial Data Abstraction Layer; www.gdal.org) we are able to read and write almost all such formats in addition to a native netCDF4 format that complies with both the Cooperative Ocean/Atmosphere Research Data Service (COARDS) and the more recent Climate and Forecast (CF) netCDF conventions (www.unidata.ucar.edu/software/netcdf/conventions.html). We have designed a native MATLAB grid structure that holds header information from the GMT grid as well as the data matrix representing the gridded values. These structures may be passed to GMT modules that expect grids and are returned from GMT modules that produce such grids. In addition, we supply a function to convert a MATLAB matrix and some metadata into a grid structure.

Images. The raster image shares many characteristics with the grid structure except the bytes representing each node reflect gray shade, color bands (1, 3, or 4 for indexed, RGB and RGBA, respectively), and possibly transparency values. We therefore represent images in another native MATLAB structure that among other items contains three components: the image matrix, a color map (present for indexed images only), and an alpha matrix (for images specifying transparency on a per-pixel level). As for grids, a wrapper function creating the correct structure is provided in the gmt function.

Color Palettes. GMT uses its flexible Color Palette Table (CPT) format to describe how the color (or pattern) of symbols, lines, polygons, or grids should vary as a function of a state variable. In MATLAB, this information is provided in another MATLAB structure that holds the color map as well as an optional alpha array for transparency values. Like grids, these structures may be passed to GMT modules that expect CPTs and will be returned from GMT modules that normally would produce CPT files. The content of the CPT can also be used to change color maps in MATLAB.

PostScript. While most users of the GMT/MATLAB toolbox are unlikely to manipulate PostScript directly, the API allows for the passing of PostScript via another data structure.

Text Data. Because of their variable record lengths and mixed numeric/text contents, we chose to represent text tables by data segments (as introduced above) containing a cell array. All GMT modules expecting text tables (or producing them) will use such segments to communicate with MATLAB. If the leading columns of data records contain numerical data, then those components will be decoded into a data matrix instead. You may also pass text data to GMT using text cell arrays, and if only a single text record is involved you can pass that string as is.

Interaction between GMT and MATLAB is done via the MEX-function gmt. This function is invoked with the syntax

$$[\text{output objects}] = \text{gmt}\ (\textit{modulename}, \textit{optionstring}[,\ \text{input objects}]);$$

where *modulename* is a string with the name of a GMT module (e.g., "surface," "grdimage," "psmeca," or even a custom extension), while the *optionstring* is a text string with the options passed to this module. If the module requires data inputs from the MATLAB environment, then these are provided as optional comma-separated arguments following the option string. Should the module produce output(s), then these are captured in standard MATLAB parlance by assigning the result of gmt to one or more comma-separated variables. Some modules do not require option strings or input objects, while other modules do not produce any output objects.

As on the command line, the gmt function has access to all modules in the core and supplemental GMT package, as well as any custom extensions installed, such as the modules developed for the Global Seafloor Fabric and Magnetic Lineation project [Wessel *et al*., 2015]. In addition, it can also use two i/o modules that are irrelevant on the command line: the *read* and *write* modules. These modules allow the toolbox to import and export any of the GMT data types to and from external files. For instance, to import a grid from the file relief.nc we run

```
G = gmt ('read', '-Tg relief.nc');
```

We use the -T option to specify grid (g), image (i), PostScript (p), color palette (c), data set (d), or textset (t). Results kept in MATLAB can be written out at any time via the write module, e.g., to save the grid G to a file we use

```
gmt ('write', 'model_surface.nc', G);
```

Because GMT data tables often contain headers followed by many segments, each with their individual segment headers, it is best to read such data using the read module since native MATLAB import via load is unable to parse such headers.

3. How Input and Output Are Assigned

Each GMT module knows what its primary input and output objects should be. Some modules only produce output (e.g., psbasemap makes a basemap plot with axes annotations) while other modules only expect input and do not return any items back to MATLAB (e.g., the write module writes the data object it is given to a file). Typically, (i.e., on the command line) users must carefully specify the input filenames and sometimes give these via a module option. Because users of the toolbox will want to provide input from data already in memory and likewise wish to assign results to variables, the syntax between the command line and toolbox commands necessarily must differ. For example, here is a basic GMT command that reads the time series raw_data.txt and filters it using a 15 unit full-width (6σ) median filter:

```
gmt filter1d raw_data.txt Fm15 > filtered_data.txt
```

Here the input file is given on the command line but input could instead come via the shell's standard input stream via piping. Most GMT modules that write tables will write these to the shell's output stream and users will typically redirect these streams to a file (as in our example) or pipe the output into another process. When using the GMT/MATLAB toolbox there are no shell redirections available. Instead, we wish to pass data to and from the MATLAB environment. If we assume that the content in raw_data.txt exists in a MATLAB array named raw_data (perhaps we ran "load raw_data.txt" or used the gmt read module) and we wish to obtain the filtered result as a segment array named filtered, we would run the toolbox command

```
filtered = gmt ('filter1d', '-Fm15', raw_data);
```

Here the filter1d module is used to perform the Gaussian filter. This illustrates the main difference between command line and toolbox usage: instead of redirecting output to a file we return it to an internal object (here a segment array) using standard MATLAB assignments of output.

For data types where piping and redirection of output streams are inappropriate (including most grid file formats) the GMT modules use option flags to specify where grids should be written. Consider a GMT command that reads (x, y, z) triplets from the file depths.txt and produces an equidistant grid using a Green's function-based spline-in-tension gridding routine:

```
gmt greenspline depths.txt -R-50/300/200/600 -I5 -D1 -St0.3 -Gbathy.nc
```

Here the result of gridding Cartesian data (-D1) within the specified region (an equidistant lattice from x from −50 to 300 and y from 200 to 600, both with increments of 5) using moderately tensioned cubic splines (-St0.3) is written to the netCDF file bathy.nc. When using the GMT/MATLAB toolbox, we do not want to write a file but wish to receive the resulting grid as a new MATLAB variable. Again, assuming we already loaded in the input data, the equivalent toolbox command is

```
bathy = gmt ('greenspline', '-R-50/300/200/600 -I5 -D1 -St0.3', depths);
```

Note that since we are returning the result back to MATLAB (i.e., via the bathy variable), -G is no longer specified among the options. In this case, the toolbox uses the GMT API to determine that the *primary* output of greenspline is a grid and that this is normally specified via the -G option. If no such option is given (or given without specifying an output filename), then we instead return the grid via memory (as here), provided a left-side assignment is specified. GMT only allows this behavior when called via an external API such as this toolbox: not specifying the primary output option (here -G) on the command line would result in an error message. However, it is perfectly fine to specify the option -Gbathy.nc in the toolbox—it simply means you are saving the result to a file instead of returning it to MATLAB.

Some GMT modules can produce more than one output (here called *secondary* outputs) or can read more than one input type (i.e., secondary inputs). Secondary inputs or outputs are always specified by explicit module options on the command line, e.g., "-Fpolygon.txt." In these cases, the toolbox enforces the

following rules: When a secondary input is passed as an object then we must specify the corresponding option flag but provide no file argument (e.g., just '-F' in the above case). Likewise, for secondary output we supply the option flag and add additional objects to the left-hand side of the assignment. All secondary items, whether input or output, must appear after all primary items, and if more than one secondary item is given then their order must match the order of the corresponding options in *optionstring*. Here are two examples contrasting the GMT command line versus GMT/MATLAB toolbox usage. In the first example we wish to determine all the data points in the file all_points.txt that happen to be located inside the polygon specified in the file polygon.txt. On the command line this would be achieved by

```
gmt select points.txt -Fpolygon.txt > points_inside.txt
```

while in the toolbox (assuming the points and polygon already reside in memory) we would run

```
inside = gmt ('select', '-F', points, polygon);
```

Here the points object must be listed first since it is the primary data expected.

Our second example considers the joining of line segments into closed polygons. We wish to create one file with all closed polygons and another file with any remaining disjointed lines. Not expecting perfection, we allow segment end-points closer than 0.1 units to be connected. On the command line, we would run

```
gmt connect all_segments.txt -Cclosed.txt -T0.1 > rest.txt
```

where all_segments.txt are the input lines, closed.txt is the file that will hold closed polygons made from the relevant lines, while any remaining lines (i.e., open polygons) are written to standard output and redirected to the file rest.txt. Equivalent toolbox usage would be

```
all = gmt ('read -Td all_segments.txt');
[ rest, closed] = gmt ('connect', '-T0.1 -C', all);
```

Note the primary output (here rest) must be listed before any secondary outputs (here closed) in the left-hand side of the assignment.

So far, the toolbox has been able to understand where inputs and outputs objects should be inserted, provided we follow the rules introduced above. However, there are two situations where more information must be provided. The first situation involves two GMT modules that allow complete freedom in how arguments are passed. These are gmtmath and grdmath, our reverse polish notation calculators for tables and grids, respectively. While the command-line versions require placement of arguments in the right order among the desired operators, the toolbox necessarily expects all inputs at the end of the function call. Hence, we must assist the toolbox command by placing *markers* where the input arguments should be used; the marker we chose is the question mark (?). We will demonstrate this need using an example of grdmath. Imagine that we have created two separate grids: kei.nc contains an evaluation of the radial $z = \mathrm{bei}(r)$ Kelvin-Bessel function (not available in MATLAB) while cos.nc contains a cylindrical undulation in the x direction. We create these two grids on the command line by

```
gmt grdmath -R-4/4/-4/4 -I256+ X Y HYPOT KEI = kei.nc
gmt grdmath -R -I256+ X COS = cos.nc
```

Later, we decide we need π plus the product of these two grids, so we compute

```
gmt grdmath kei.nc cos.nc MUL PI ADD = answer.nc
```

In the toolbox, the first two commands are straightforward:

```
kei = gmt ('grdmath', '-R-4/4/-4/4 -I256+ X Y HYPOT KEI');
C = gmt ('grdmath', '-R -I256+ X COS');
```

but when time comes to perform the final calculation we cannot simply do

```
answer = gmt ('grdmath', 'MUL PI ADD', kei, C);
```

since grdmath would not know where kei and C should be put in the context of the operators MUL and ADD. We could probably teach grdmath to discover the only possible solution since the MUL operator requires two operands but none are listed on the command line. The logical choice then is to take kei and C as operands. However, in the general case it may not be possible to determine a unique layout, but more

importantly it is simply too confusing to separate all operators from their operands (other than constants) as we would lose track of the mathematical operation we are performing. For this reason, we will assist the module by inserting question marks where we wish the module to use the next unused input object in the list. Hence, the valid toolbox command actually becomes

```
answer = gmt ('grdmath', '? ? MUL PI ADD', kei, C);
```

Of course, all these calculations could have been done at once with no input objects but often we reuse results in different contexts and then the markers are required.

The second situation arises if you wish to use a grid as argument to the -R option (i.e., to set the current region to that of the grid). On the command line this may look like

```
gmt pscoast -Reurope.nc -JM5i P -Baf -Gred > map.ps
```

However, in the toolbox we cannot simply supply -R with no argument since that is already an established shorthand for selecting the *previously* specified region. The solution is to supply –R?. Assuming our grid is called Europe then the toolbox command would become

```
map = gmt ('pscoast', '-R? -JM5i -P -Baf -Gred', europe);
```

4. Examples of Toolbox Use

The GMT and MATLAB combination is extremely flexible, letting the user harvest the general numerical and graphical capabilities of both systems. The GMT/MATLAB toolbox represents a giant step forward in interoperability between GMT and other software packages (i.e., beyond mere file format compatibility). It is difficult to do justice to the range of operations possible with this well-matched combination. Also, by keeping the interface low-level and simple we will have the resources to maintain it, while developers can build more user-friendly GUI layers on top of the basic interface. Here we will present four typical examples of GMT/MATLAB toolbox use, ranging from the simple to the complex. Each script shows a mix of basic MATLAB language constructs interspersed with calls to the toolbox. We only give general comments on these scripts here; for details on the syntax used for either MATLAB or GMT commands we refer you to the relevant documentation.

4.1. Example 1: Gridding

Our first example illustrates gridding of ship track bathymetry near the Geologists seamounts southwest of Hawaii via robust, median-based averaging followed by gridding using a minimum curvature spline in tension algorithm. The result is visualized in Octave using contour and surf. While this is a simple example, we note that the blockmedian and surface combination powers the creation of many global data sets [Becker *et al*., 2009] and that our gridding module surface is widely used across all sciences (citations of our splines-in-tension algorithm, *Smith and Wessel* [1990] exceed 1000). The script is shown below while Figure 2 shows a screenshot of the toolbox in action in Octave.

```
geo = gmt ('read' '-Td geologists.txt'); % Read in the point data
% Decimate data using median spatial averaging on a 1 arc min lattice
ave = gmt ('blockmedian' '-R158:00W/156:40W/18:00N/19:40N -I1m' geo);
% Grid the data using splines in tension
G = gmt ('surface' '-R -I1m -T0.2 -N1000 -C1e-6' ave);
% Plot the result in MATLAB
figure(2); clf; subplot (2,1,1)
contour (G.x, G.y, G.z)
hold on; axis equal
plot (ave(:,1), ave(:,2), 'g.' 'MarkerSize' 2)
axis (G.range(1:4)); subplot (2,1,2)
surf (G.x, G.y, -G.z); title ('Geologists Seamounts')
```

Here GMT was used for various calculations not easily done in Octave while we display the contents of the structures ave and G using standard Octave graphics.

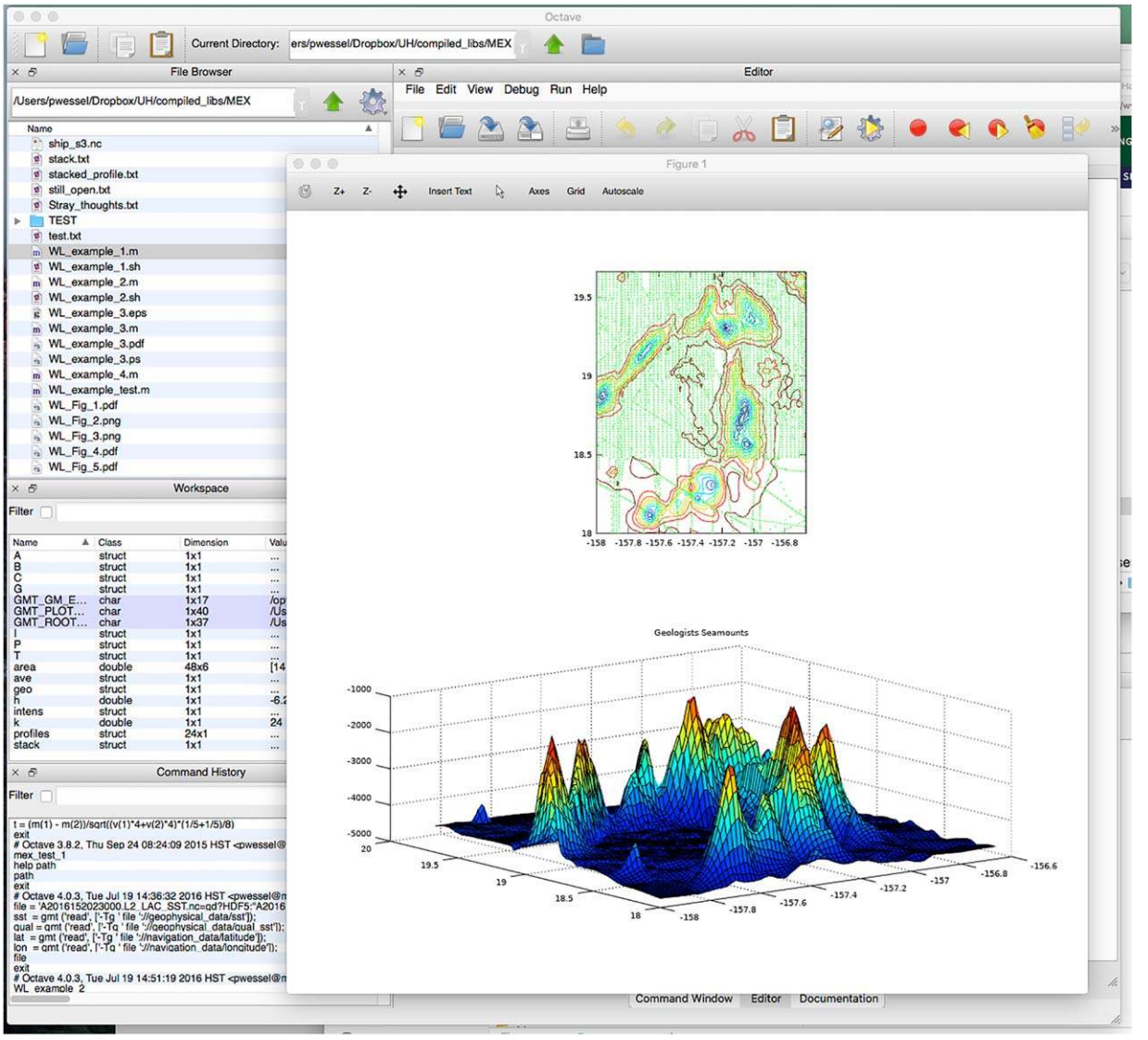

Figure 2. Screen capture of situation once our first script has completed. Octave displays the illustration, while we see the Workspace window indicate we have three structures containing the data we read or produced.

4.2. Example 2: Profiling

Our next example is more ambitious and imports a relief grid for an area off Japan as well as a line data set reflecting the location of the Japan Trench. We wish to create profiles normal to the trench every 25 km, with each profile being 300 km long and sampling the grid every one km along the profiles. Finally, these profiles, which all have distance = 0 at the trench axis, are stacked and a median profile is computed. Both the set of profiles and the median profile are returned to MATLAB for further analysis. The script is shown below while Figure 3 shows a screen capture of the toolbox in action, where two figure windows were created: one displays a PNG image created by GMT and another displays the individual profiles and highlights the stacked median profile.

```
% Read in relief grid and Japan trench location
G = gmt ('read -Tg JP.nc');
T = gmt ('read -Td JP.txt');
% Take gradient in N45E direction to be used for illumination
intens = gmt ('grdgradient -Nt0.8 -A90' G);
% Sample grid along profiles normal to the trench
```

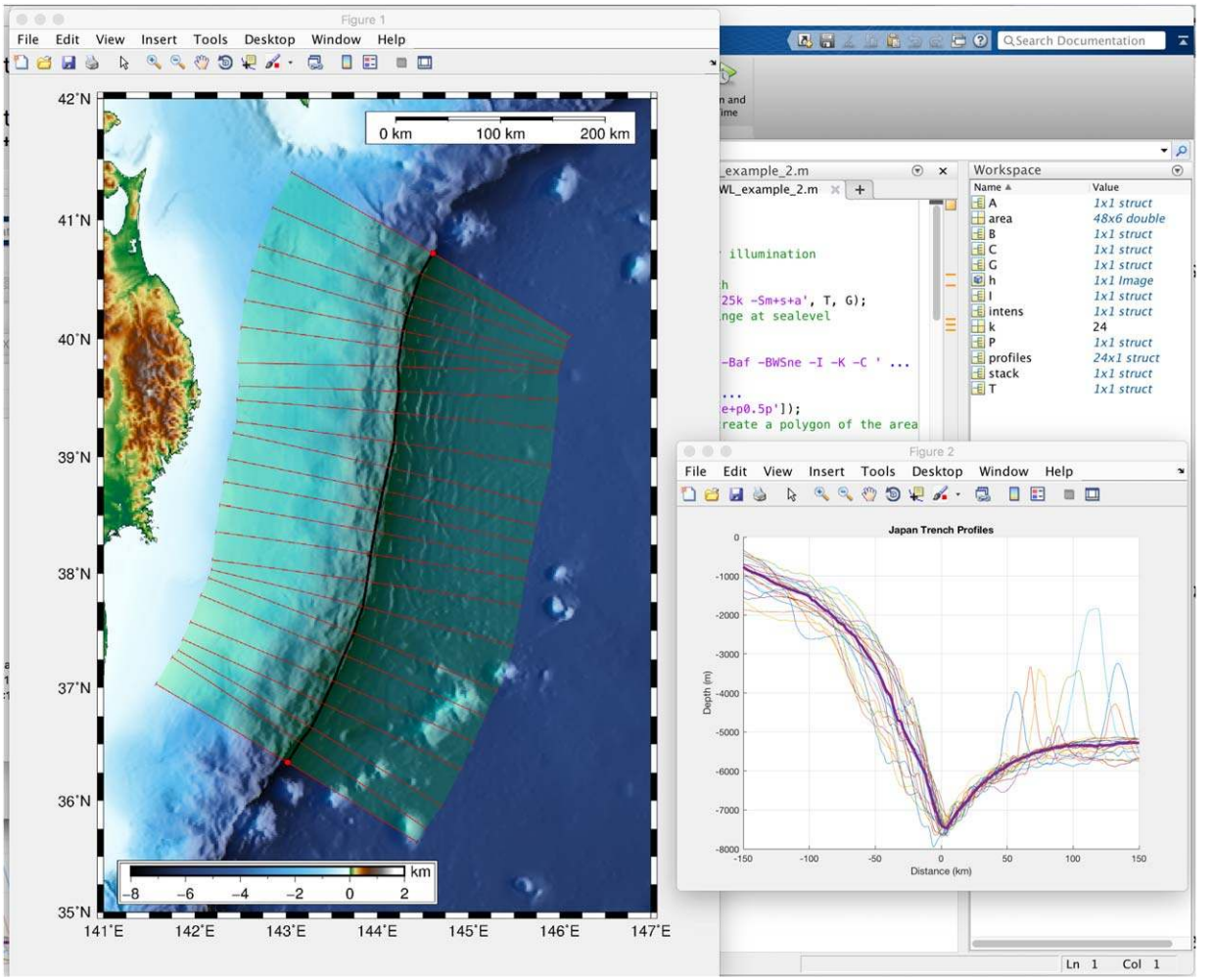

Figure 3. Screen capture of situation once our second script has completed. MATLAB displays two illustrations: A PNG image produced by GMT but loaded into MATLAB and a MATLAB graph of the individual profiles, with the stacked profile drawn in a heavier line.

```
[ profiles, stack] = gmt ('grdtrack -G -C300k/1k/25k -Sm+s+a' T, G);
% Evaluate an asymmetrical color palette with hinge at sealevel
C = gmt ('makecpt' '-Cgeo -T-8000/2000');
% Make the GMT plot
P = gmt ('grdimage' [ '-R141/147/35/42 -JM6i -P -Baf -BWSne -I -K -C '...
  '--FORMAT_GEO_MAP=dddF'] , G, intens, C);
P = gmt ('pscoast' [ '-R -J -O -K -W0.25p -Dh '...
  '-LjTR+w200k+u+f+c38:30N+o0.5i/0.2i -F+gwhite+p0.5p'] );
% Use first and last point of each profile and create a polygon of the area
A = gmt('convert' '-Ef -T' profiles);
B = gmt('convert' '-El -T -Is' profiles);
area = gmt ('catsegment' [ A B] ); % Join the two segments
P = gmt ('psxy' '-R -J -O -K -Ggreen@85' area);
P = gmt ('psxy -R -J -O -K -W2p+v0.3c+gred+p0.25p+bc+ec' T);
P = gmt ('psxy -R -J -O -K -W0.5p,red+v0.25c+p0.25p+bt+et' profiles);
P = gmt ('psscale' [ '-R -J -O -DjBL+w3i/0.1i+h+o0.3i/0.4i -C -W0.001 '...
  '-F+gwhite+p0.5p+i0.25p -Bxaf -By+l"km"'] , C);
% Convert plot to a PNG and display in MATLAB as Figure 1.
I = gmt ('psconvert -TG -P -E300 -A' P);
```

```
figure(1); clf
h = imshow(I.image); set(h, 'AlphaData' I.alpha)
% Figure 2 shows the stacked relief profiles across the trench
figure(2); clf; hold on
for k=1:length(profiles)
  plot(profiles(k).data(:,3), profiles(k).data(:,5))
end
plot(stack.data(:,1), stack.data(:,2), 'LineWidth' 3)
xlabel('Distance (km)'); ylabel('Depth (m)');
title('Japan Trench Profiles'); xlim([-150 150]); grid on
```

Note we are here mixing GMT and MATLAB graphics: the map is made as a GMT PostScript plot but converted to PNG, then read and displayed by MATLAB, whereas the profiles are displayed directly in MATLAB. Note the use of an asymmetrical color table with different data ranges for sea and land. We maintain the PostScript code in MATLAB and this structure auto-appends each new overlay added.

4.3. Example 3: Data Conversion

Our third example illustrates reading in satellite measurements of sea surface temperature stored as several individual arrays inside a single HDF5 data set. We read these separate arrays, exclude low-quality data points, grid the data stored in satellite sensor coordinates using a nearest neighbor algorithm, and mask the grid away from the data constraints. Finally, we generate a PDF illustration (Figure 4) which we display within MATLAB via its open command.

```
% Import sea surface temperature grids from several HDF5 layers
file = 'A2016152023000.L2_LAC_SST.nc=gd?HDF5:"A2016152023000.L2_LAC_SST.nc"';
sst = gmt('read' ['-Tg' file '://geophysical_data/sst']);
qual = gmt('read' ['-Tg' file '://geophysical_data/qual_sst']);
lat = gmt('read' ['-Tg' file '://navigation_data/latitude']);
lon = gmt('read' ['-Tg' file '://navigation_data/longitude']);
% Ignore points of low quality that is stored in the quality flags array
sst.z(qual.z > 0) = [];
lon.z(qual.z > 0) = [];
lat.z(qual.z > 0) = [];
% Perform nearest neighbor gridding
G = gmt('nearneighbor' '-R-12/-1/33/43 -I0.01 -S0.05' [lon.z' lat.z'
sst.z']);
% Create a mask that is NaN where no data exist
mask = gmt('grdmask' '-R -I0.01 -NNaN/1/1 -S0.02' [lon.z' lat.z']);
% Apply the mask to limit the plot that follows
G.z = G.z .* mask.z;
% Select color table and make PDF illustration and display it in MATLAB
cpt_s = gmt('grd2cpt' '-Cjet -E -M --COLOR_NAN=255' G);
cpt_t = gmt('makecpt' '-Celevation -T0/3000');
gmt('grdimage' '-JM15c -C -Ba -BWSne -P -K > WL_example_3.ps' G, cpt_s)
arg = ['-TdjTR+w3c+o1c+f3+l --FONT_TITLE=12 --MAP_TITLE_OFFSET=5p'];
% Clip topography to only plot over land, using shading
gmt('pscoast' '-Di -Gc -A1000 -J -R -O -K >> WL_example_3.ps')
int = gmt('grdgradient' '-R etopo1m.nc -A0 -Nt0.7');
gmt('grdimage' '-R -J -O -K -C etopo1m.nc -I -t50 >> WL_example_3.ps' ...
  cpt_t, int)
gmt('pscoast' '-Q -O -K >> WL_example_3.ps')
gmt('pscoast' ['-Di -W0.5p -A1000 -N1/0.5p -J -R' arg '-O -K' ...
  '>> WL_example_3.ps'])
gmt('psscale' ['-R -J -C -DJBC+w14c/0.25c+o0/1c+h+ef -Baf+u" m" ' ...
  '-O -K >> WL_example_3.ps'], cpt_t)
```

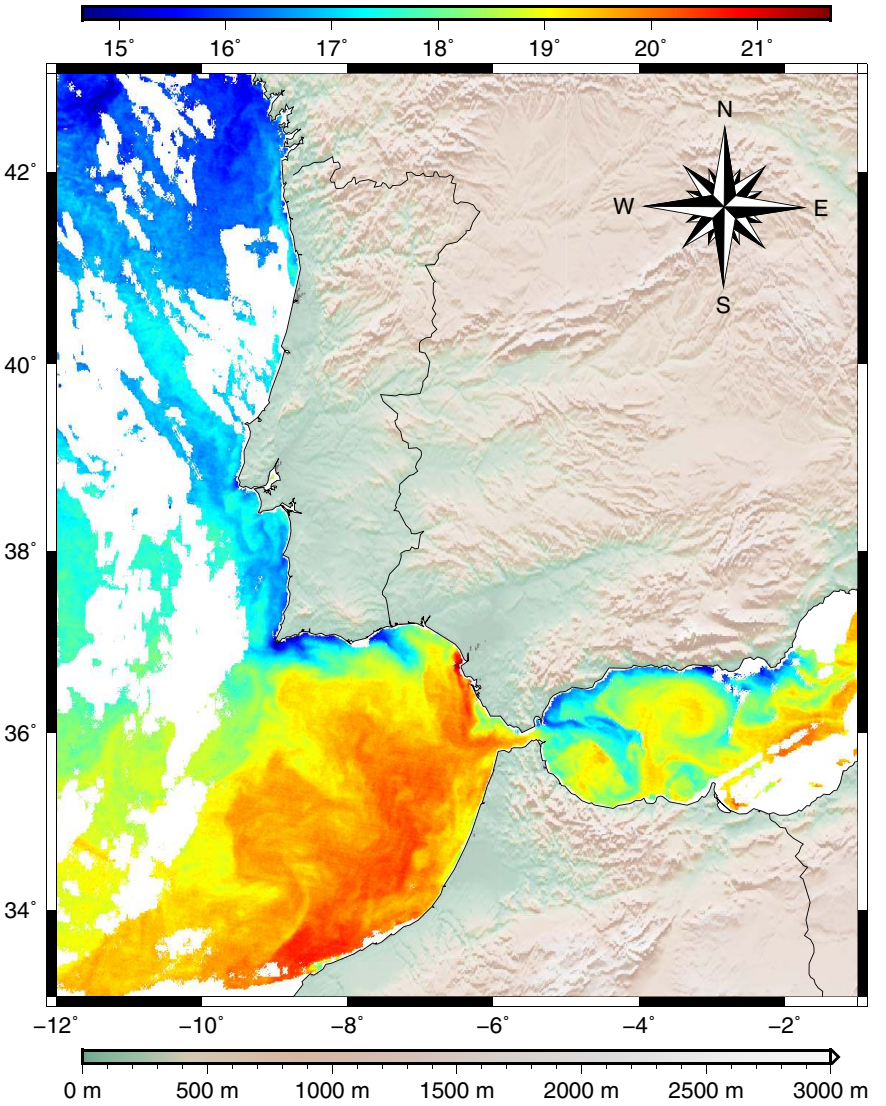

Figure 4. Sea-surface temperatures from a MODIS Level 2 scene (best resolution of all provided products at the OceanColor site, ~1 km) for the day 1 June 2016 near the Straits of Gibraltar from NASA OceanColor (http://oceancolor.gsfc.nasa.gov/cms) site. We use ETOPO1 to paint the land and overlay political boundaries and color bars. MATLAB assists with data processing while GMT makes a publication-ready PDF map.

```
gmt ('psscale' [ '-R -J -C -DJTC+w14c/0.25c+o0/1c+h -Ba1f+u\232 '...
  '-O >> WL_example_3.ps'], cpt_s)
gmt ('psconvert' '-Tef -P -A WL_example_3.ps')
open ('WL_example_3.pdf');
```

Here we focus on the sea surface temperature grids over the oceans by washing out the land topography using 50% transparency. The two color-bars relate colors to data values, while national borders and a direction rose add distinction to the map.

4.4. Example 4: Crossover Error Analysis

Our final example highlights GMT's capability of determining intersections between arbitrary lines and to estimate any mismatch in observed along-track quantities at these intersections; another task not easily done in MATLAB. Such information is useful when determining systematic biases that affect certain instruments, requiring further data processing. We let MATLAB display a map showing the worst offenders and a histogram of the crossover error distribution; this is then saved as a PDF illustration (Figure 5).

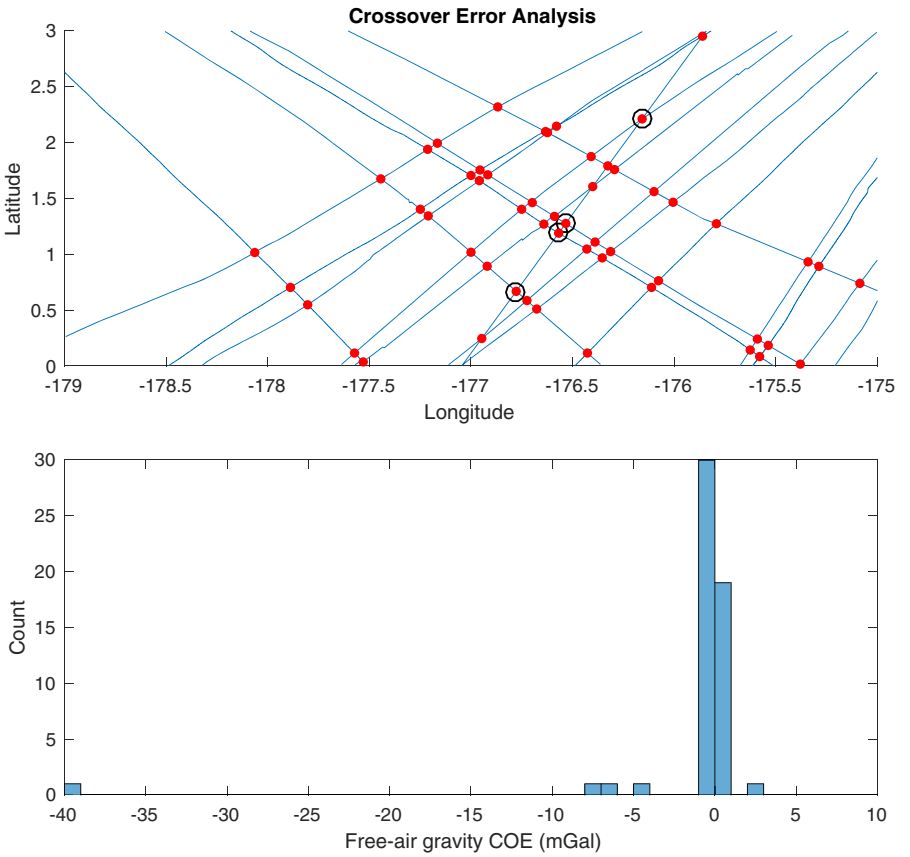

Figure 5. MATLAB illustration of the crossing lines, their intersections (red circles) and the subset of large crossover errors (circles). The crossover error distribution is presented as a histogram, identifying outliers as those exceeding ±4 mGal.

```
% Read in all tracks
all = gmt ('read -Td all_tracks.txt');
% Initiate the crossover system
here = pwd;
setenv ('X2SYS_HOME' here);
gmt ('x2sys_init' 'TEST -Dgeoz -Etxt -F -Gd -R-180/-175/0/5')
% Compute internal crossovers
cross = gmt ('x2sys_cross' '-TTEST all_tracks.txt -Qi -Ia');
% Make plot showing crossovers and a histogram of their values
% Find which crossings exceed dz = 4.
bad = find (abs(cross.data(:,11)) > 4);
figure (4); clf; subplot (2,1,1); hold on
xlabel ('Longitude'); ylabel('Latitude');
tracks = gmt ('catsegment' all, 1);
plot (tracks(:,1), tracks(:,2)); hold on
plot (cross.data(bad,1), cross.data(bad,2), 'ko' 'MarkerSize' 10)
title ('Crossover Error Analysis')
% Plot tracks and lcoations of crossings
plot (cross.data(:,1), cross.data(:,2), 'r.' 'MarkerSize' 15)
subplot (2,1,2)
histogram (cross.data(:,11), -40:1:10)
xlabel ('Free-air gravity COE (mGal)'); ylabel('Count');
```

Here GMT relies on the x2sys supplemental tool for the analysis and MATLAB for the display.

5. Installation and Limitations

The core GMT package is available from the GMT site; several Linux distributions also provide this package. However, installing the GMT/MATLAB toolbox varies between various platforms, mostly due to the dependency on the user's MATLAB installation (and version) and possible complications arising from shared library conflicts. We will there discuss these procedures separately; please see our Wiki for updates.

5.1. Windows

The standard GMT installer for Windows already comes with the optional GMT/MATLAB toolbox that is expected to work with a wide range of MATLAB versions. Given a valid MATLAB version then all that is required is to run the GMT installer and add the GMT bin directory to the MATLAB path.

5.2. OS X

MATLAB users on OS X can install a prebuilt GMT distribution with the toolbox from the project website. Alternatively, they may build the toolbox using their installed version of MATLAB and the GMT OSX Bundle distributed from the GMT website. Furthermore, users who wish to gain access to the latest GMT version via subversion must first build and install their own GMT bundle. Instructions for doing these steps can be found on the GMT/MATLAB Toolbox wiki page.

5.3. Linux

As for OS X, MATLAB users on Linux must build the toolbox themselves, using the instructions found on the GMT/MATLAB Toolbox wiki page. We have not yet been able to avoid a shared library conflict between MATLAB and GMT's GDAL dependency so for now we recommend building GMT without the GDAL library. This means the GMT/MATLAB Toolbox can only open files that the core GMT API understands. We expect this limitation to be lifted with more experience building the toolbox. We note that the shared library conflicts do not affect the toolbox built for Octave.

Acknowledgments

P. W. acknowledges support from the US National Science Foundation via grant OCE-1029874. Reviews by Dennis Harry, Sumant Jha, and an anonymous reviewer greatly improved the paper. This is SOEST contribution 9908.

References

Becker, J. J., et al. (2009), Global bathymetry and elevation data at 30 Arc Seconds resolution: SRTM30_PLUS, *Mar. Geod.*, *32*, 355–371.

Olson, C. L., J. J. Becker, and D. T. Sandwell (2014), A new global bathymetry map at 15 arcsecond resolution for resolving seafloor fabric: SRTM15_PLUS, Abstract OS34A-03 presented at 2014 Fall Meeting, AGU, San Francisco, Calif., 15–19 Dec.

Smith, W. H. F., and P. Wessel (1990), Gridding with continuous curvature splines in tension, *Geophysics*, *55*, 293–305.

Wessel, P., and W. H. F. Smith (1991), Free software helps map and display data, *EOS Trans. AGU*, *72*, 441–446.

Wessel, P., and W. H. F. Smith (1995), New version of the Generic Mapping Tools released, *EOS Trans. AGU*, *76*, 329–336.

Wessel, P., and W. H. F. Smith (1998), New, improved version of Generic Mapping Tools released, *Eos Trans, AGU*, *79*, 579.

Wessel, P., W. H. F. Smith, R. Scharroo, J. F. Luis, and F. Wobbe (2013), Generic Mapping Tools: Improved version released, *EOS Trans. AGU*, *94*, 409–410.

Wessel, P., K. J. Matthews, R. D. Müller, A. Mazzoni, J. M. Whittaker, R. Myhill, and M. T. Chandler (2015), Semiautomatic fracture zone tracking, *Geochem. Geophys. Geosyst.*, *16*, 2462–2472, doi:10.1002/2015GC005853.