The evoLS library

Documentation for the implementation of level set evolution by (generalized) radius of curvature

Author:
Aaron Berk

Supervisor:
Dr. Nicholas Kevlahan

Department of Mathematics & Statistics
McMaster University

September 10, 2013

1 Introduction

Each of the number of tsunamis that have occurred in the recent past has demonstrated the awesome amount of destruction of which these natural disasters are capable. The ability to use realistic numerical simulations to understand better the myriad factors affecting tidal flow could help in the design of effective countermeasures against tsunamis. However, the construction of a realistic numerical simulation is entirely non-trivial. Several tools are required to construct such a simulation of tidal flow on the surface of the Earth (*i.e.,* tidal flow subject to realistic bottom bathymetry and coastline data). The method will be based on solving an incarnation of the shallow water equations using an adaptive wavelet collocation scheme, with boundary conditions implemented via Brinkman penalization (refer to [5] for a more thorough summary of these methods).

Subject to the methods mentioned above, working with realistic bottom bathymetry and coastline data of the Earth is another non-trivial aspect of constructing a realistic numerical simulation. Because of the ability of adaptive wavelet methods to "zoom in" on regions of the domain that require higher resolution, it is necessary to be able to control how finely, how accurately and how smoothly the coastline can be discretized at arbitrary regions throughout the grid. When fine scale features must be resolved near the coastline, the coastline itself must be resolved finely too, so that boundary conditions (and more generally, local geometric information) may be approximated accurately. Moreover, this information should be able to be calculated to within an arbitrary tolerance, and in such a way that the method does not attempt to resolve more detail than is possible for the scale given. Namely, this means that the curvature of the coastline must be small enough that it can be effectively resolved by the size of the grid, lest the propagation of numerical error result.

For the sake of an example, we refer the reader to Figure 1. It is clear that many small islands are represented by the "default" map of the Earth. With the desired numerical implementation, it is impractical to represent such features, and make the simulation significantly more prone to numerical error. Moreover, highly curved and irregular regions of the coastline are visible. Such regions cannot be well-represented, nor well-accounted for by boundary penalization methods. Thus, it is necessary to come up with a well-defined way of altering this map, such that prominent features remain in tact, while problematic features are eliminated.

In this document, we elucidate the formation of a method for obtaining a "smooth" version of the Earth's coastlines, using a mathematical characterization of what constitutes "smoothness". This method accepts an input map and its corresponding domain, and outputs a version of the coastal data where small features (*i.e.,* those features that are deemed too small to be represented properly by the grid, or where boundary penalization methods will not work correctly) are "removed" and the detail of coastal features is smoothed and scaled according to the grid size (or pre-defined user settings). Moreover, so that our methods may work in conjunction with an adaptive wavelet scheme, we introduce a method of approximating to a finer detail a given space curve, where this space curve is represented implicitly by a matrix, and the output is given as a list of values representing points in the domain through which that space curve passes. The algorithm is designed so that the space curve need not be topologically closed or connected.

Methods

2 Data acquisition

Bottom bathymetry data were obtained from the National Oceanic and Atmospheric Administration's (NOAA) National Geophysical Data Center. These data were taken from ETOPO1 [1] using the WCS Grid Extract Tool. These data were used in the production of all relevant images, below.

3 Level set methods

Solving the shallow water equations on a Cartesian domain where the boundary conditions are irregularly-shaped requires knowledge of the vectors normal and tangent to the boundary at each point in the domain. As the number of domain points (*i.e.,* grid points) is adaptive, so too, must be the process by which the normal and tangent vector information is calculated. Moreover, given that the spacing of the domain is a limiting factor in the resolution of detail, it is necessary that the boundary data contain no more detail than that which can still be resolved by the grid. These factors thus impose the following conditions on the coastline data:

1. the interface must be manipulated in such a way where geometric information (such as normal and tangent information) is preserved;
2. the interface can be resolved to different scales, according to the size of the local grid cells;
3. the smoothness of the interface can be controlled in a well-defined way, so that only features that can resolved by the domain, or by the penalization methods, are represented.

To this end, we use a *level set method* to perturb the interface. "Level set methods are a collection of numerical algorithms for solving a particular class of differential equations" [3], which are capable of satisfying requirements 1 and 3 above, and easily adaptable to 2. These methods implicitly represent the interface using a *level surface* or *level set function* $\phi_0(x, y) \subseteq \mathbb{R}^3$ so that the interface is the zero isocontour of the level surface:[1]

$$\Gamma(x, y) := \{(x, y) \ : \ \phi_0(x, y) = 0\} \subseteq \mathbb{R}^2.$$

We then *evolve* a level surface $\phi = \phi(x, y, t) \subseteq \mathbb{R}^3$ (and thus, the interface defined by $\phi = 0$) through time, using $\phi(x, y, 0) = \phi_o(x, y)$ as the initial conditions, and defining the evolution by some equation(s) of motion, thereby obtaining a new level surface (and new interface) at a later time value. We define the equation of motion of the level surface by a Hamilton-Jacobi partial differential equation (HJ PDE), consequently defining the motion of the interface, also. For a more thorough description of level set methods, as well as motivation for the use of HJ PDEs, we refer the reader to the first several chapters (esp. Ch. 3) of [4].

As per [4], the equation of motion will be of the form

$$\phi_t + \vec{V} \cdot \nabla\phi = 0, \tag{1}$$

where $\phi = \phi(x, y, t)$, the t subscript denotes a temporal partial derivative with respect to the time variable t, and ∇ represents the gradient operator (so that $\nabla\phi$ is a row vector whose elements are the spatial derivatives of ϕ). Here, $\vec{V}(\vec{x}, t)$ is some velocity field which depends on $\vec{x} = (x, y, z) \in \mathbb{R}^3$ and the time variable, $t \in \{x \in \mathbb{R} \ : \ x \geq 0\}$. To satisfy requirement 3 above we must choose a suitable vector field $\vec{V} = \vec{V}(x, y, z, t)$ by which to evolve ϕ. Now, as the level surface is going to evolve in such a way where the "smoothness" of the surface is scaled according to grid size, we know that $\vec{V}$ should correspond to a self-generated velocity field.

By saying that ϕ solves a PDE, we necessitate that ϕ is, in fact, differentiable in time and space. Due to further constraints that we impose later, it does little more to impose that ϕ be twice differentiable in space. With this assumption, curvature is now an important geometric property made available to use by the use of level set methods.

[1] In fact, it is precisely for this reason that we refer to the coastline data using the word, "interface" — namely, that the coastline data represents the land-water interface, and that it is represented as the interface between ϕ_0 and the zero function, $z = 0$.

3.1 Derivation of level set equation for motion by mean curvature

Let $\phi = \phi(x, y, t)$ be a level surface. For any time $t^* \geq 0$, the unit vector that is normal to the interface defined by $\phi(x, y, t^*) = 0$, at a point (x, y) lying on the interface, is defined by

$$\vec{N} = \frac{\nabla\phi}{|\nabla\phi|}. \tag{2}$$

Moreover, the mean *curvature*, κ is defined as the divergence of the unit normal,

$$\kappa = \nabla \cdot \vec{N} = \nabla \cdot \left(\frac{\nabla\phi}{|\nabla\phi|} \right). \tag{3}$$

Thus, given the level set equation as defined by Equation 1, let $\vec{V} = -b\kappa\vec{N}$, where $b > 0$ is a constant, and where κ and $\vec{N}$ are as previously defined. Equation 1 now becomes

$$\phi_t - b\kappa\vec{N} \cdot \nabla\phi = 0$$

With a simple manipulation [4], this equation takes the form

$$\phi_t - b\kappa\,|\nabla\phi| = 0 \tag{4}$$

Our choice of $\vec{V}$ implies that the level surface moves in the direction of the curvature of the level surface, with a velocity proportional to the magnitude of the curvature. Thus, highly curved areas become less curved. Therefore, we have described a level set equation which roughly satisfies requirement 3, above.

In practice this equation has shown to be insufficient for smoothing the interface while also retaining characteristic features of the interface. In particular, in order for regions of high curvature to become smooth enough, it is necessary to increase b and/or the duration of the evolution; however, this serves to smoothen lower-curvature regions of the level surface too greatly. Therefore, we require an implementation of the above level set method which is able to "slow" the evolution of the level surface in regions where the local curvature is small enough, while still evolving regions of high curvature. Thus, we generalize Equation 4 (equation (4.5) of [4]) by introducing an additional multiplier to serve as a scaling function.

Let $\psi_\beta(\kappa) = \mathbb{R} \to [0, 1]$ be a piecewise continuous function from the domain of ϕ to the unit interval, subject to some parameter $\beta > 0$ and such that $\psi_\beta|_{\kappa \geq 0}$ is monotonic, as is $\psi_\beta|_{\kappa \leq 0}$. β will be chosen in a such a way that allows for the relative smoothness of the final interface to be controlled. The task of choosing a specific function for ψ will be left until later. Redefine $\vec{V}$ by

$$\vec{V} = -b\kappa\psi_\beta(\kappa(x, y, t), t)\vec{N}, \tag{5}$$

so that Equation 4 becomes

$$\phi_t - b\kappa\psi_\beta\,|\nabla\phi| = 0. \tag{6}$$

3.1.1 Caveats

Earlier when deriving the evolution of a surface by mean curvature, it was stated that the parameter b in Equation 6 must be positive. In fact, if $b < 0$, then the solution to the level set equation is ill-posed [4].

3.2 Signed distance functions and the re-initialization equation

Up until now, we have given no constraints on what kinds of functions ϕ we can use to embed a zero isocontour. As the zero isocontour is the only part of the level surface that will be used after the level surface evolution, there is quite a bit of choice in how to select the function ϕ. One such choice that will simplify matters greatly, and help to minimize the propagation of numerical error is to ensure that ϕ is a *signed distance function*. That is, we mandate that the magnitude of the gradient of the level surface be everywhere unity:

$$|\nabla\phi| = 1. \tag{7}$$

By placing such a constraint on ϕ, we see that Equation 6 in fact simplifies to [4]:

$$\phi_t - b\psi_\beta\Delta\phi = 0, \tag{8}$$

where Δ is the Laplacian operator so that $\Delta\phi = \phi_{xx} + \phi_{yy}$. Moreover, ensuring that $\phi_0(x, y)$ is a signed distance helps to prevent large gradient values and other effects that scream, "numerical instability".

There is no unique process by which one calculates a signed distance function. The most straightforward approach is perhaps the following.

3.2.1 Straightforward numerical calculation of signed distance function

Let $\phi : \mathbb{R}^2 \to \mathbb{R}$ be a level surface so that the zero isocontour of ϕ represents a finite number of closed curves in the plane — call this set Γ. Regions bounded by these curves are said to be interior to the zero isocontour, and are thus deemed "inside" regions, Ω^-. Analogously, exterior regions are said to be "outside" regions, Ω^+. One can then implicitly construct the signed distance function ϕ^* corresponding to the zero isocontour according to the equation

$$\phi^*(x, y) = \text{sgn}(x, y)\left(\inf_{(g,h)\in\Gamma} \text{d}((x, y), (g, h))\right),$$

where d is the Euclidean distance function, $\text{sgn}(x, y) = 1$ if $(x, y) \in \Omega^+$ and $\text{sgn}(x, y) = -1$ if $(x, y) \in \Omega^-$. If $(x, y) \in \Gamma$, then $\phi^*(x, y) = 0$.[2]

3.2.2 Re-initialization equation

Note that a signed distance function for the given isocontour Γ can also be found as the solution to a level set equation. Indeed, the level surface of Γ which is also a signed distance function is, in fact, the steady state solution to the *re-initialization equation*

$$\phi_t + |\nabla\phi| = 1, \tag{9}$$

where $\phi(x, y)$ is any level surface containing the zero isocontour of interest, such that $\phi < 0$ for regions inside Γ and $\phi > 0$ for regions exterior to Γ.

The result follows, because when $\phi_t = 0$, the above equation reduces to $|\nabla\phi| = 1$, which is precisely the definition of a signed distance function.

[2] For a more thorough summary of signed distance functions, we refer the reader to Chapter 2 of [4].

While the original zero isocontour is not, by default, preserved by the solution of this equation, simple steps can be taken to ensure that the zero isocontour is preserved. Note that for the purposes of this paper, the purpose is to evolve the zero level set, and thus we do not take such precautions in the numerical implementation.

This method is often preferable to the straightforward calculation seen above, because the straightforward calculation can be computationally slow. Indeed, one is often able to approximate a solution to Equation 9 at regular intervals during the process of another level set evolution, thereby preserving the accuracy and computational efficiency of that evolution. Note however, that in order to be able to take advantage of computational speed gains, it is necessary to solve for the steady state solution locally in a narrow band about the zero contour, and by using accelerated iteration methods. In such schemes, it would be necessary to be more clever about how the re-initializations are implemented, since in such schemes it *would* be desirable to preserve the geometry of the zero isocontour.[3]

Numerical implementation

We now discuss how the above theory was implemented in MATLAB. Namely, we describe the scripts and methods belonging to the `evoLS` library that was created, which is used to take an input matrix representing bathymetry data and output a "smoothed" version of the coastline represented implicitly by those data for the purpose of implementing effective and "problem-free" boundary penalization methods.

4 The level set toolbox

Fortunately, an implementation of many level set methods was coded as a MATLAB toolbox by Ian Mitchell from UBC [2, 3]. `toolboxLS-1.1` includes not only many tools for solving level set equations, but numerous scripts exemplifying the capabilities of the toolbox, many of which are seen as examples in [4]. Please refer to any of the aforementioned references for a more detailed discussion on the numerical algorithms available for use with these methods (*e.g.,* derivative approximations, ODE solvers).

5 Evolution by radius of curvature

For the sake of an example, we use a sample map taken from the NOAA database [1] and use it as the input matrix for the level set method. The implementation of the level set method is loosely based on a framework for an example of evolution by mean curvature, provided by Mitchell [2, 3]. We have also modified other files that are used in conjunction with this framework. The details of these files are discussed below.

The framework of the process of evolving a level set by radius of curvature is coded in `evoLS/evoByRoC/radiusOfCurvatureLSevo.m` (variations of this file exist in the same parent directory, and are named in a similar way, with a summary of differences contained in the header).

[3] For more detail on the re-initialization equation, see Chapter 7 of [4].

5.1 Description of radiusOfCurvatureLSevo.m

As the framework of the overall method, the purpose of the script file is to load in a geoTIFF file that is stored in the `evoLS/images` directory, to convert it into a format appropriate for level set evolution, to carry out a re-initialization process similar to that described above (using level set evolution), to run the level set evolution(s) by radius of curvature, and then to output the grid structure that was used, in addition to the result of the level set evolution itself (formatted as a matrix).

```
%% radiusOfCurvatureLSevo.m

%% This function will act as a framework to consolidate other
% methods in the implementation of a level set method.  By
% default, the level set method is written to use evolution by
% radius of curvature. By default, this script runs method using
% 'medium' accuracy, and the "large" etopo1 geotiff file.
%
% Note that many parts of this file can be modified to add or
% remove functionality.
```

```
%-------------------------------------------------------------
% Add paths.
%
% This script was written with the use of rsync in mind so that
% the entire evoLS folder can be mirrored across different
% systems. Consequently, it is expected that there is a .mat file
% containing a single string object which is located in the same
% directory as the evoLS folder. The string object contains the
% absolute path to the "Kernel" folder containing the methods of
% the LStoolbox. Note that this string object can also be used as
% an absolute reference to the location of the evoLS folder.
load('../../parentPath.mat');
addpath(genpath([parentPath, 'LStoolbox/Kernel']));

%-----------------------------------------
% add path to custom methods
addpath(genpath([parentPath, 'evoLS/methods']));
```

To read in the image file, we have to load the directory in which the `evoLS` folder resides. The file `parentPath.mat` in the `evoLS/..` folder contains the path to the `LStoolbox` itself. This is loaded so that the path to the `LStoolbox` itself can be loaded.

It is important to note that the script file expects this `.mat` file containing the "parent" path (and that its location resides in the directory of the `evoLS` folder). This is done for the purpose of mobility, so that the entire library can be mirrored on different systems (*e.g.,* using `rsync`).

```
%-----------------------------------------
% Script parameters

% size of map to evolve:
mapSize = 'Large'; % {'Small', 'Large'}.

% relative location of save directory in which to save output
savedir = 'resdata/';

% The savefile prefix is used for the series of images generated
% by evolving the surface of the (_mapSize_) (map) of Northern
% South America extending up to Floridian Coast using curvature
% (L)evel (S)et methods that evolve the surface at a rate
% proportional to the local curvature (inversely proportional to
% the radius of curvature).  To this prefix will be appended the
% duration of surface evolution, the curvature parameter value,
% and appropriate file extension (.mat, .png)
%
% if doSave then this is the name of the file to save; if
% ~doSave, then this is the base name of the file to load.
savefileprefix = ['map', mapSize(1),'_RoCLS'];
```

Next, script parameters are set. As there are principally two images that have been used as test examples, we use the `mapSize` variable to differentiate between the two, and set the save directory and save-file prefix for the output matrix.

```
% ------------------------------------------------------------
% Import and scale GeoTIFF

geotiffFN = [parentPath, 'evoLS/images/etopo1', mapSize, '.tif'];
try
  fprintf('Trying to read in geotiff file using geotiffread...');
  [geoA, geoR] = geotiffread(geotiffFN);
  fprintf('success!\n');
catch ME
  fprintf('failed!\nIt''s likely that this version of MATLAB does not have ↩
     geotiffread (see explanation below...)\n\n');
  display(ME);

  fprintf(['\n\nTrying alternative read in; requires geoR',...
     mapSize, '.mat...\n']);
  if exist(['../images/geoR', mapSize, '.mat'])==2
    fprintf(['    found geoR', mapSize, '.mat\n']);

    fprintf(['    reading geoR', mapSize, '.mat\n'])
    load(['../images/geoR', mapSize, '.mat']);

    fprintf('    reading TIFF image using imread...');
    geoA = imread(geotiffFN);
    fprintf('Success!\n\n');
  else % geoR file not found.
    error(['geoR', mapSize, '.mat not found. Ensure that geoR', ...
```

```
      mapSize, '.mat is located in ../images and try again']);
  end
end

% Convert geoA to proper format
geoA = double(geoA);
d_init = geoA./max(abs(geoA(:))); % in range (-1, 1);
         % zero contour remains unshifted

%-------------------------------------------------------------
% Make grid

g.dim = 2;
g.min = [geoR.Lonlim(1) + geoR.DeltaLon/2;
   geoR.Latlim(1) - geoR.DeltaLat/2];
g.max = [geoR.Lonlim(2) - geoR.DeltaLon/2;
   geoR.Latlim(2) + geoR.DeltaLat/2];
g.N = size(geoA).';
g.bdry = @addGhostExtrapolate;

g = processGrid(g);

clear geoA; % geoA not needed anymore
```

The geoTIFF file is either loaded using `geotiffread`, or using `imread` depending on whether or not the version of MATLAB being used has the geotiffread function. If the version of MATLAB does not have the geotiffread function, then a second file, `geoR.mat` must also be loaded, which contains a `struct` object with fields `Lonlim, Latlim, DeltaLat` and `DeltaLon`. The first two are 1×2 row vectors representing the bounds of the map as latitude and longitude, while the latter two are scalars representing the grid size.

The map image object (hereafter referred to as the input matrix) is converted to a type double matrix and scaled within a ball of unit radius, but not translated so that zero-value elements remain unchanged.

The grid is formed from the `georaster` object (or the `geoR.mat` object). The grid represents the domain of the input matrix. The `processGrid` function is then used to complete the remaining fields of the `grid` object. The `grid` object is one that is specific to the `toolboxLS` package, and so similarly, too, is the `processGrid` function.

```
%-------------------------------------------------------------
% Evolve Level Surfaces proportional to local curvature

fprintf('Attempting reinitialization...\n');

% medium accuracy: 516.74 s on 949 x 1562 = 1.48 M pixels
d_reinit = evoLS_reinit(d_init, g);
clear d_init;
fprintf('Reinitialization complete.\n');

fprintf('Starting level set evolution(s) by RoC...\n');
fprintf('--parametrizing runs...\n');
% note that bVal and tMax can be vectors.
bVal = 0.04;
```

```
tMax = 2.3;
parMat = combvec(tMax, bVal);
clear bVal tMax; % not needed anymore.

% check multiparm (if so, expect long run?)
nRuns = size(parMat, 2);

fprintf('--total number of runs is %02d\n', nRuns);

for k = 1:nRuns
  fprintf(['--beginning curvature evolution %02d of %02d;\n' ...
     '----tMax=%5.3f, bVal=%4.3f...\n'], ...
    k, nRuns, parMat(1, k), parMat(2,k));
  % medium accuracy: 354.45 s on 949 x 1562 = 1.48 M pixels
  d_curv = evoLS_curvature(d_reinit, g, parMat(1,k), ...
         parMat(2,k), 'low', 1);
  fprintf('---Complete!\n');

  fprintf('--Saving result...');

  description = ['An attempt at evolving etopo1', mapSize, ...
     ' using evoLS_curvature --- a function' ...
     ' that attempts to evolve the surface' ...
     ' according to local radius of curvature' ...
     ' (if the surface is ''too curved'',' ...
     ' then the surface evolves more quickly;' ...
     ' if surface curvature is within' ...
     ' appropriate bounds, then evolution is' ...
     ' locally suppressed). Evolution was run' ...
     ' via radiusOfCurvatureLSevo.m. The simulation' ...
     ' duration of the run was', num2str(parMat(1,k)),...
     ' with a bMltplr of', num2str(parMat(2,k)), ...
     '. Note that d_init can be obtained by' ...
     ' rescaling the original image found in the' ...
     ' associated tiff file: ', geotiffFN, ').'];

  % define name of savefile.
  savefile = [savedir, savefileprefix, ...
        '.tMax', num2str(parMat(1,k)), ...
        '.b', num2str(parMat(2,k)), '.mat'];
  save(savefile, 'description', 'g', 'd_curv');

  clear description d_curv; % remove before next run.

  fprintf('success!\n');

end
```

The above code chunk is effectively pseudo code. This section of the code is where the important computations of the level set method are performed (and the details hidden within other methods). Namely, the input matrix is first put through a re-initialization function. As described previously, the purpose of this is to return an intermediate matrix such that the absolute value of the gradient of this matrix is everywhere unity.

The purpose for this re-initialization is to lower computational complexity and increase accuracy in the radius of curvature evolution. Note that as this method does not affect the zero isocontour of the input matrix, this function does not adversely affect the accuracy of level set methods that are used afterward.

Next, we define parameters that will be passed to the main method that executes the evolution by radius of curvature — `evoLS_curvature`. These parameters are the b value(s), and the duration(s) for which the evolution will be run. These parameters are passed iteratively to the `evoLS_curvature` method using a `for` loop so that different parameter combinations can be analyzed in batch.

The b value is a constant multiplier to the velocity of each point on the interface, where the interface moves in the normal direction with velocity proportional to its curvature. As stated above, the second parameter represents the duration of the evolution.

After each output matrix is returned, a description with the relevant parameter values is created, and then the grid object, the description and the output matrix, `d_curv` are saved to an appropriately named .mat file of the form

`save_directory/savefileprefix.tMax##.b##.mat`

where `##` represents the numeric value of each input parameter.

```
fprintf('Level set evolutions complete.\n\n');

% Stuff for remote runs to quit the matlab once run is completed.
fprintf('\nexiting script...\n');
W = what;
if ~isempty(strfind(W.path, '/1/home/berkas/'));
  exit;
end
```

This last part of the code chunk writes to `stdout` that the simulation has finished, and checks to see for a token indicating that the script is running remotely (*e.g.,* on Anatolius — one McMaster's Math Department's super computers). If this is the case, then `exit` function is called to close MATLAB. Note that if this is statement is true when the script is not being run remotely, then the current MATLAB session will necessarily close. Also note that if this code chunk is not present when the script is run remotely, then the MATLAB session will not exit upon completion (thereby possibly causing the user a bit of a headache).

5.2 Description of `term` functions

During the calculation of the level set methods, it is necessary to compute the value of the term that is equal to ϕ_t (*e.g.,* $b\psi\kappa|\nabla\phi|$ in Equation 6). For each iteration, these values are computed by the Term functions, found in the `LStoolbox/Kernel/ExplicitIntegration/Term` folder of the parent path. In other methods (such as `evoLS_curvature` [see below]) the `term` functions are invoked as *function handles*, as they primarily serve as "helper" functions or component functions to other methods to solve for ϕ at the next time step. A basic description of how these work can be found in [3].

5.2.1 termCurvature.m

This function is discussed in more detail in [3]. Any gaps between that document and the usage of this function in the context of the `evoLS` library can be filled in by the description of `termRadiusOfCurvature.m`, below.

5.2.2 termRadiusOfCurvature.m

```
function [ ydot, stepBound, schemeData ] = termRadiusOfCurvature(t, y, ↩
    schemeData)
% termRadiusOfCurvature: approximate a "generalized" motion by mean
% curvature term in an HJ PDE.
%
% [ ydot, stepBound, schemeData ] =
%                             termRadiusOfCurvature(t, y, schemeData)
%
% Computes an approximation of generalized motion by mean
% curvature for a Hamilton-Jacobi PDE.  This is a second order
% equation that simplifies to a heat equation if the function is a
% signed distance function and teh scaling function (see below) is
% unity.  Specifically:
%
%        D_t \phi - b(x,t) \psi(x,t) \kappa(x) \| \grad \phi \| = 0.
%
```

The function `termRadiusOfCurvature` is used to calculate the value of $b\psi(\kappa)\kappa(x,y)\left|\nabla\phi\right|$. It takes the current time (given by `t`), the current values of ϕ (given by `y`) and `schemeData` as arguments and outputs: the `schemeData`, the maximum time step to add to the value `t` (stored as `stepBound`, this value is used in the CFL condition for the ODE solver), and the change in the function values, `ydot` (equivalent to ϕ_t).

```
  %---------------------------------------------------------
  % According to O&F equation (4.5).
  [ curvature, gradMag ] = feval(thisSchemeData.curvatureFunc, grid, data);

  %---------------------------------------------------------
  % Get multiplier
  if(isa(thisSchemeData.b, 'double'))

    checkStructureFields(thisSchemeData, 'tSwitch');

    if (t <= schemeData.tSwitch)
      b = thisSchemeData.b;
    else
      tolRoC = 3;
      b = thisSchemeData.b.*scalingFn(curvature, tolRoC);
    end
      ⋮
```

While the framework of this function is very similar to that of `termCurvature`, it differs in its implementation in the calculation of the curvature term of the level set equation (Equation 4). Namely, the curvature and gradient magnitude are calculated before the multiplier is retrieved from `schemeData` and formatted. The curvature is then used in the calculation of the multiplier, as outlined in subsection 3.1. It should be noted that the remainder of the code in the conditional statements is not used in the current implementation and as such, it is omitted.

```
%-----------------------------------------------------------
  % As above, according to O&F equation (4.5)
  delta = -b .* curvature .* gradMag;

 %-----------------------------------------------------------
  % According to O&F equation (4.7).
  stepBound = 1 / (2 * max(b(:)) * sum(grid.dx .^ -2));

  % Reshape output into vector format and negate for RHS of ODE.
  ydot = -delta(:);
```

After MATLAB has successfully navigated the set of conditional statements, it calculates the "curvature term", where the value of `b` is equal to $b\psi$, `curvature` is equal to κ, and `gradMag` is equal to $|\nabla\phi|$. This value is stored as `delta` and, after the calculation of `stepBound` (which is used as a parameter for the CFL solution method (see [3, 4]), the negative value of `delta` is returned as `ydot`.

```
%-----------------------------------------------------------
% The scaling function
%    is used as a multiplier to the curvature term in the level
%    set equation. It is denoted by \psi_\beta(\kappa), where
%    where \beta is a parameter of \psi defined by the user,
%    and \kappa is the curvature of \phi.
function ret = scalingFn(kappa, beta)
  ret = exp(-beta./kappa.^2);
% The choice of the scaling function is not unique; however, we
% recommend a function similar to the one used above, because
% it is smooth, even, monotonic on each side of zero, and
% contained in the interval [0,1].
```

Again, the multiplier, `b`, is defined as the product of the b value stored in `schemeData` and the value of the scaling function evaluated at the given curvature at the current time. The scaling function is defined at the end of `termRadiusOfCurvature.m` (though for stylistic reasons, this could be changed easily), and takes two arguments — the first is the independent variable of the scaling function, curvature (`curvature`), while the second is the parameter mentioned in subsection 3.1 which controls how much smoothing will be applied to the zero isocontour (this parameter is stored as `tolRoC` and passed to `scalingFn` as `beta`). In the current regime, higher values of `tolRoC` correspond to smoother (*i.e.,* more circular) connected component subsets of the zero isocontour.

For the sake of the current implementation, the scaling function ψ_β is defined as

$$\psi_\beta(\kappa(x,y)) := \exp\left(-\frac{\beta}{\kappa(x,y)}\right), \tag{10}$$

where $\kappa(x,y)$ is the curvature of ϕ at the point (x,y). Though there exist other choices for ψ_β, Equation 10 was used because:

1. it's C^∞
2. $\lim_{\kappa \to 0} \psi_\beta = 0$
3. $\lim_{\kappa \to \pm\infty} \psi_\beta = 1$
4. it's monotonic on each side of 0 (*i.e.,* $\psi_\beta|_{\kappa \geq 0}$ is monotonic; likewise for $\kappa \leq 0$).

Consequently, as a region of the level surface becomes less curved, its evolution slows, while highly curved regions continue to evolve at a rate that is nearly equal to $b\kappa|\nabla\phi|$. The parameter value β can be chosen to control how quickly the evolution slows. For example, it could be chosen by the criteria that when $\kappa = 3$ the rate of evolution is slowed by a factor of 2.

5.3 Description of evoLS_curvature.m

```
function data = evoLS_curvature(data, g, tMax, bVal, accuracy, evoByRoC)
% evoLS_curvature: method used to evolve a given level surface by
% mean curvature (by default) or "generalized" mean curvature
% (i.e., subject to some scaling multiplier).
%
%   [ data ] = evoLS_curvature(data, g, tMax, bVal, accuracy, evoByRoC)
%
% Parameters:
%   accuracy: Controls the order of approximations.  Note that the spatial
%   approximation is always second order.
%
%                 'low'         Use odeCFL1.
%                 'medium'      Use odeCFL2 (default).
%                 'high'        Use odeCFL3.
%
%   evoByRoC: Boolean.  Use the radius of curvature-dependent
%   version of the flow field, which adjusts the relative b value
%   coefficient in proportion to the local radius of curvature (in
%   fact, by the local curvature, itself).
%   g: struct.  The domain on which the surface is to be
%   evolved. All grids should be passed _after_ passing them
%   through the function processGrid from the LStoolbox.
%   data: double.  A matrix object that represents the height
%   of the initial level surface at each point in space (thus
%   assuming a regular [Cartesian] grid cell structure).
%   bVal: scalar. this value is used as the constant multiplier
%   to the curvature term in the level set equation. Good
%   results have been obtained with this value in the range of
%   0.02--0.04.
%   tMax: scalar. This represents the duration of the level set
%   evolution. Good results have been obtained for tMax in the
%   range of 1--2.5.
%
% Output Parameters:
%   data: Implicit surface function at tMax.
```

This section contains a description of the method `evoLS_curvature`, found in `evoLS/methods`. This method is used by `radiusOfCurvatureLSevo.m` to evolve a given level surface by (generalized) mean curvature. The function takes in the input data formatted as a matrix, the domain of the data as a grid `struct` object, the duration of the simulation, `tMax`, the b value which serves as a coefficient multiplier to the curvature term of the level set equation, `bVal`, the accuracy (medium by default) and whether or not to use "generalized" evolution by mean curvature, given as the Boolean value `evoByRoc`.

```
%-------------------------------------------------------------
% Setting parameter values
% Duration parameter.
if(nargin < 3)
  tMax = 1.75;
end

% Curvature speed parameter.
if(nargin < 4)
  bValue = 0.02;
else
  bValue = bVal;
end

if(nargin < 5)
  accuracy = 'medium';
end

% Use the time dependent motion?
if(nargin < 6)
  useRoCDependent = 0;
else
  useRoCDependent = evoByRoC;
end

%-------------------------------------------------------------
% Integration parameters.

% Can change this vector if desired.
plot_points = 0:tMax/10:tMax; % length = 16
%tMax = max(plot_points);                  % End time.
t0 = 0;                                   % Start time.

% How close (relative) do we need to get to tMax to be considered
% finished?
small = 100 * eps;
```

First the parameters of the simulation are defined (if these have been omitted by the user). Then the number of sections in which to perform the integrations is set. This variable is poorly named as `plot_points` as an artifact of the previous code. The purpose of this variable is output how many integration steps it took to advance to the next time value stored in `plot_points`. For example, if this variable is the vector `[ 0 1 2 ]`, then the code will print messages of the form

```
### steps in ### seconds from 0 to 1
### steps in ### seconds from 1 to 2
```

to be used as feedback for how the algorithm is progressing and to approximate how long the algorithm will take to complete.

```
%---------------------------------------------------------------
% Set up functions used for motion by mean curvature

schemeData.grid = g;
schemeData.curvatureFunc = @curvatureSecond;
```

```
if(useRoCDependent)
  % Time dependent flow field
  % Use modified termCurvature function to increase speed (and
  % hopefully accuracy, somehow?)
  schemeFunc = @termRadiusOfCurvature;
  schemeData.b = bValue;
  schemeData.tSwitch = 0.5 * tMax;
else
  schemeFunc = @termCurvature;
  % Time independent flow field is constant.
  schemeData.b = bValue;
end
```

This code chunk checks whether to use evolution by mean curvature, or generalized mean curvature. In the latter case, the scheme function, `schemeFunc`, is set to use `termRadiusOfCurvature` to calculate the value of ϕ_t. The parameter `tSwitch` is used to tell the method when to "turn on" evolution by *generalized* mean curvature. That is, if `tSwitch` is equal to `0.5*tMax`, then the surface will evolve by mean curvature until half of the total duration of the simulation, after which the surface will continue evolve subject to the scaling function (described above). If `useRoCDependent` is false, then regular evolution by mean curvature occurs (cf. section 2.3.1 of [3]).

```
%---------------------------------------------------------------
% Set up time approximation scheme.
integratorOptions = odeCFLset('factorCFL', 0.9, 'stats', 'on');

% Choose approximations at appropriate level of accuracy.
switch(accuracy)
 case 'low'
  integratorFunc = @odeCFL1;
 case 'medium'
  integratorFunc = @odeCFL2;
 case 'high'
  integratorFunc = @odeCFL3;
 otherwise
  error('Unknown accuracy level %s', accuracy);
end

%---------------------------------------------------------------
% Initialize time stepping (variables are poorly named)
plot_count = 1;

if(plot_points(1) == t0)
  plot_count = plot_count + 1;
```

```
end
```

This set of lines is used to set up the integration parameters of the simulation and is described in further detail in [3].

```
%-------------------------------------------------------------
% Loop until tMax (up to allowed roundoff).
tNow = t0;
startTime = cputime;
while(tMax - tNow > small * tMax)

  % Reshape data array into column vector for ode solver call.
  y0 = data(:);

  % How far to step?
  tSpan = [ tNow, plot_points(plot_count) ];

  % Take a timestep.
  [ t y ] = feval(integratorFunc, schemeFunc, tSpan, y0,...
                  integratorOptions, schemeData);
  tNow = t(end);

  % Get back the correctly shaped data array
  data = reshape(y, g.shape);

  plot_count = plot_count + 1;

end

endTime = cputime;
fprintf('Total execution time %g seconds\n', endTime - startTime);
end
```

In this section, the initial time of the simulation is stored in `tNow`. The evolution process is then initialized by `while` loop so that the simulation will continue as long as the difference in the total duration and the current time is larger than some arbitrary threshold value (set to 100ε multiplied to the total simulation duration, where ε is the machine ε value of the system).

The loop stores the data matrix as a column vector, `y0`. It also creates a 1×2 column vector whose first entry is the current time value and whose second entry is the next time value. When the integration is performed, it will integrate (in smaller steps) until it reaches this second time value. This operation is performed by the `integratorFunc` which is defined according to the accuracy level used. This function uses the `schemeFunc`, the time bounds `tSpan`, the input data formatted as a column matrix `y0`, the options passed to the integrator for alternative functionality `integratorOptions` (these are unchanged from the default script, and the `schemeData` which represents the parameter values of the simulation, as well as the various approximation methods used during the integration (namely, differencing methods). The integration returns the new level surface data represented as a column matrix, and the corresponding time vector `t`. This data is reshaped, and the iterative variable `plot_count` is increased by one so that the integration using the updated surface can be performed over the next time interval.

When the integration has finished, the output data representing the evolved level surface is returned as a matrix, as well as the grid `struct` object representing the domain of the function.

5.4 Simulations

All simulations were performed either on one of McMaster's supercomputers, Anatolius (see details here) or on a 2011 MacBook Pro with a 2.6 GHz Intel Core i7 and 8 GB RAM with solid state drive. Thus, the `cputime` values in the log files of the library correspond to run times on one of these systems.

6 Numerical results

Here we highlight some of the results of the level set evolutions that were performed using data retrieved from the NOAA database. We include examples from level set evolutions using a constant multiplier for the curvature term of the level set equation (Equation 4) and use these results to motivate the use of level set evolution by generalized mean curvature. We then highlight preliminary results of evolution by generalized mean curvature and include notable *caveats* when implementing generalized mean curvature to evolve a level surface function.

Note that in each case, initial conditions for the level surface evolutions by curvature were obtained from the default ETOPO1 data and then reinitialized with medium accuracy using `methods/evoLS_reinit.m` (we omit a description of this method, as it is similar to the default methods of the level set toolbox, and the framework of `methods/evoLS_curvature.m`). The resulting surface was then used as a initial data to the level set evolution function.

6.1 Evolution by constant-multiplier mean curvature

The figures used in this section were generated by a family of scripts contained in the `constCurv` folder of the `evoLS` library. The script that generated the data matrix used in the visualizations is very similar to the framework script `radiusOfCurvatureLSevo.m` in the `evoByRoC` folder (see description above); however, it is now deprecated in favour of `radiusOfCurvatureLSevo.m`. For this reason, and because the code file is well-commented, we omit a detailed description of this MATLAB script.

The visualizations themselves were originally generated by the script file `summarizeResData.m` and the method `evoLSPlotSummary`, where the former is used as a framework to call the latter. In particular, the former was used to iterate through all of the data files created during the simulations, while the latter does the bulk of the work of creating the visualizations. Depending on the system on which the visualizations are created, we note that the output resolution of the images may be rather poor; however, we invite the user to change these settings according to their own system, as it is not computationally intensive to re-generate the visualizations (barring unreasonably high resolution values).

For the purpose of clarity of representation, the visualizations were regenerated using methods located in the `README` folder in the top-level directory of the `evoLS` library (see `README/curvPlot.m` and `README/curvPlotScriptForScalMeanCurv.m`). This was done for the purpose of creating correctly-sized images to display in this document, as well as saving the images using names that can be read easily by the document compiler.

The first level surface we choose to exemplify was run for a duration of 1.5 with a b value of 0.01. This can be interpreted as a "slow" (low constant multiplier value) level set evolution of short duration. By reference

to the default zero isocontour in Figure 1, it can be seen by comparison to Figure 2 that the overall contour has changed little. While many of the rather chaotic-looking features have been removed (very small islands; erratic land features in the north-west), there still remain "holes" in the continents (*e.g.,* in Florida), and there are still many other small islands, which could not be resolved by boundary penalization methods.

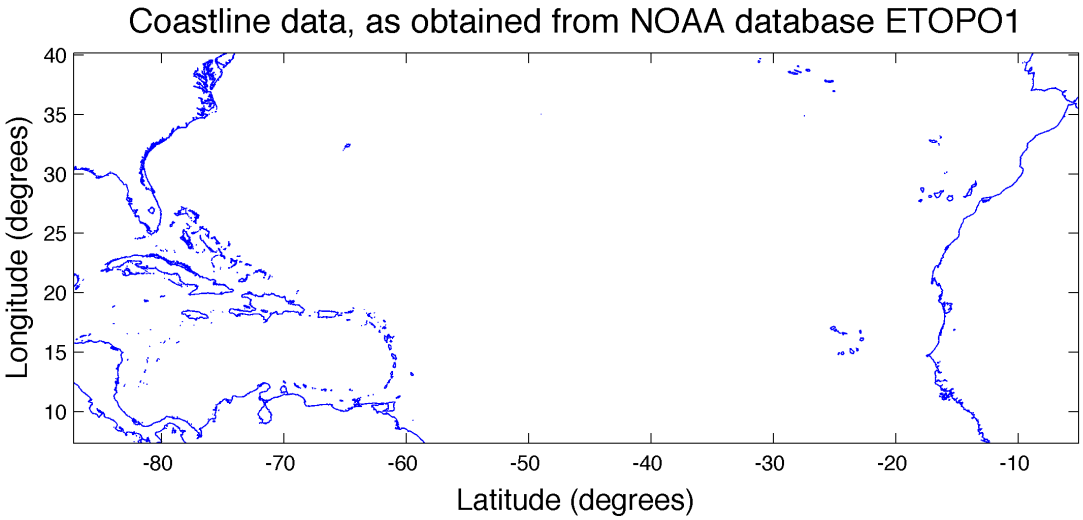

Figure 1: The zero isocontour of the bathymetry data contained in `images/etopo1Large.tif`, as obtained from the NOAA database. This visualization was created using MATLAB's `contour` function to generate a zero isocontour from the level surface given by the NOAA bathymetry data. Note that this figure displays the coastline *without* any modification by level set evolution (*i.e.,* it is a representation of the zero isocontour of the *initial* level surface). Small features can be better resolved by zooming in on the image in the user's favourite PDF viewer.

By Figure 3 it becomes clear that too high a b value results in a too *little* detail in the coastline, in addition to a significantly morphed coastline (it visibly seems no longer to resemble the real coastline of the Earth). This evolution was run for a duration of 1.5 with a b value of 1. This can be interpreted as a sensible duration, with a very high value for the constant multiplier. One prominent difference between Figure 3 and Figure 2 is the amount of noise that is present in the visualization of log-curvature. Indeed, there is significantly more noise in Figure 2, which is not present in Figure 3. We expect this behaviour, since the surface in Figure 3 has evolved much more, relative to the other surface. Thus, it will have become much smoother, and will retain less of the noise artifacts that exist as a result of the data collection.

We include as our last example of level set evolution by mean curvature with constant multiplier the surface that we believe turned out "best". The coastline in Figure 4 appears smooth, and retains significant detail present in the real coastline. There are no "holes" in the continents, and the remaining islands are significantly larger than the previous examples. The duration of this evolution was 2.3, with a constant multiplier of $b = 0.04$. The visualization of curvature that is also present in this figure shows that the curvature near the zero isocontour (the coastline) is relatively small. This is a type of "safety check" that bodes well for the level surface, as those surfaces with highly varying curvature values near the zero isocontour are typically indicative of parameters and initial conditions that result in unstable evolutions/numerical instability and/or unwanted features.

It is for the features that have been described in the past several paragraphs that we seek when implementing level set evolution by generalized radius of curvature. Moreover, we will refer back to Figure 4 as point of comparison for the surfaces discussed below, highlighting similarities, differences and any disadvantages that may result from the generalized method.

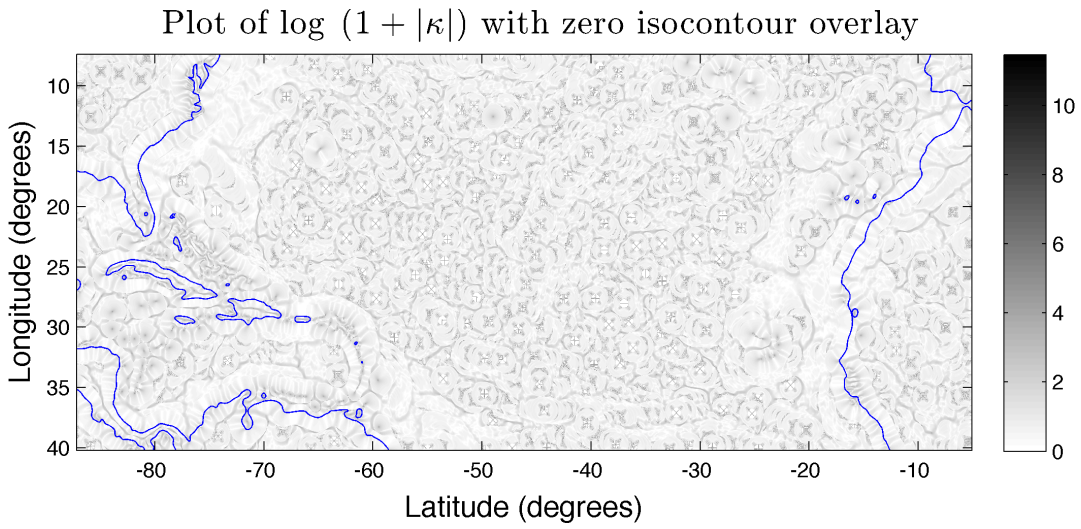

Figure 2: This visualization shows the the log-curvature of a constant-multiplier mean curvature level set evolution. The evolution was run with $b = 0.01$ for a duration of $t = 1.5$. The data for this visualization was originally obtained from the NOAA database. The latitude and longitude of the region used can be obtained from the `.mat` file containing the data matrix and its domain, `evoLS/constCurv/resdata/mapL_curvatureLS.tMax1.5.b0.01.mat`.

6.2 Evolution by scaled-multiplier mean curvature

Evolution by scaled-multiplier mean curvature was implemented by introducing a scaling function into the `termRadiusOfCurvature.m` file. [4] The scaling function is defined as in Equation 10, with a β value of 3. β was chosen to be 3 because once the curvature is less than 3, evolution slows to approximately 70% of the maximum possible rate and a curvature of 3 is approximately what we expect the grid domain to be able to resolve. Below we exemplify three different simulations, all having the same b value of 0.04, but run for different durations.

The level surface corresponding to the Figure 5 was run for a duration of $t = 1.5$. As above, this length of time can be interpreted as relatively short. With this configuration of parameters, we notice that most of the small, problematic features have disappeared, and that the curvature near the zero isocontour looks relatively smooth, and small. As per subsection 6.2, we note that these are good traits for the level set evolution to have. Unfortunately, it is still clear from the figure that there exist numerous small scale details which appear as distorted features not present in the real coastline contour (*e.g.,* the two "holes" in southern Central America/northwestern South America). Moreover, the loop created in northern South America around longitude 10, latitude 14 appears as though the top of the loop would be too small to implement Brinkman penalization.

Because South America seems to present obvious difficulties with respect to level set evolution, we zoomed in on this region and ran the simulation for a longer duration ($t = 2.3$). Figure 6 exemplifies that the "loop" is indeed small, but appears to be less problematic than in the previous figure. Using the larger image (see `evoByRoC/resdata/mapL_RoCLS.tMax2.3.b0.04.mat`) as a point of comparison, we see that there are no longer two holes present in the southern part of Central America — they appear to have merged. Otherwise, note that Figure 6 and Figure 4, having the same parameter values for simulation, appear very similar, boding well for the evolution by radius of curvature using a scaled-multiplier to the curvature term.

Unfortunately, as the artifacts discussed above may still present numerical difficulties for a boundary

[4] Note that this was retrospectively realized *not* to be the best way. The user should see `evoByRoC/README_ImplementationNote.rtf` for more details and a suggestion on reimplementation.

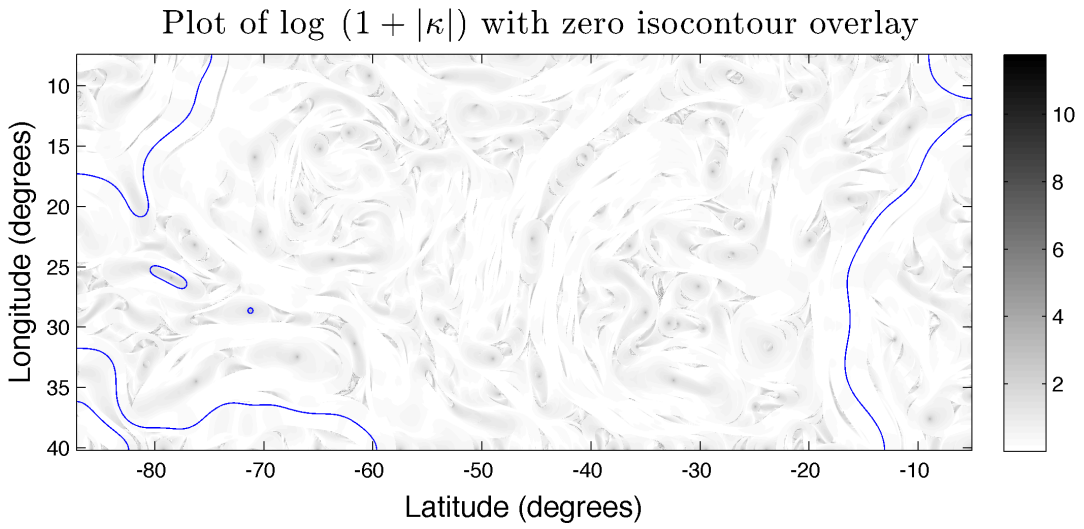

Figure 3: This visualization shows the the log-curvature of a constant-multiplier mean curvature level set evolution. The evolution was run with $b = 1$ for a duration of $t = 1$. The data for this visualization was originally obtained from the NOAA database. The latitude and longitude of the region used can be obtained from the `.mat` file containing the data matrix and its domain, `evoLS/constCurv/resdata/mapL_curvatureLS.tMax1.b1.mat`.

penalization scheme, we examine one last visualization (Figure 7) to examine the differences between the figures just discussed and a figure generated from a simulation of still-longer duration.

Run for a duration of $t = 4$, Figure 7 appears as though it may be slightly better than the last two figures in some respects, and slightly worse than others. Note that the detail of the coastline is significantly lessened by comparison with either Figure 5 or Figure 6. Moreover, the problematic "loop" is still a presence in northern South America, as well as a new "feature" to the east. However, the log-curvature values of the level surface appear to be better behaved than the other two surfaces, which lends some credence to the stability of the method (nothing is "blowing up"). Additionally, while the coastline *is* much smoother than the other scaled-multiplier visualizations, we observe that this coastline retained significantly more detail than its constant multiplier counterpart (cf. Figure 3 or `constCurv/resdata`).

6.3 Discussion and *caveats*

We can conclude several pieces of information from the above figures. Most obviously, level set evolution is a complex, but effective way to smooth a given space curve. There are many parameters to control that do not appear to have an obvious relation with one another. Moreover, while it is not yet clear how reinitializing the level surface mid-simulation may affect results, it serves as an example of one more factor that could affect the outcome of the level surface. Additionally, we observe that scaled-multiplier level set evolution by radius of curvature could serve as an effective way to localize where evolution of the level surface could occur. Unfortunately, this is yet another complex addition to level set evolution process and it is not yet clear mathematically how this multiplier impacts convergence or numerical stability of the algorithm. Below we provide several suggestions on how the evolution of the level surface using scaled multipliers could be improved — with the potential to benefit output, computational efficiency, code readability/usability and possibly stability.

A few notable *caveats* of using motion by generalized mean curvature to evolve a level surface are the following:

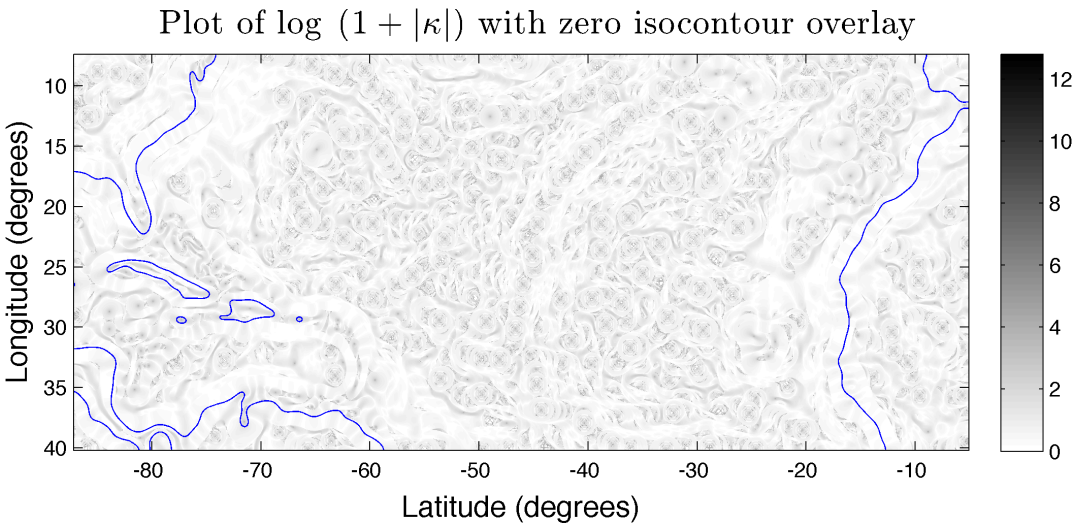

Figure 4: This visualization shows the the log-curvature of a constant-multiplier mean curvature level set evolution. The evolution was run with $b = 0.04$ for a duration of $t = 2.3$. The data for this visualization was originally obtained from the NOAA database. The latitude and longitude of the region used can be obtained from the `.mat` file containing the data matrix and its domain, `evoLS/constCurv/resdata/mapL_curvatureLS.tMax2.3.b0.04.mat`.

- We have not had time to verify the numerical stability of these algorithms, especially in regions where the curvature appears to oscillate on a scale that is close to that of the grid size. Indeed, we have noticed some numerical instability in such cases (see, for example, the small feature detail in the southwest region of Figure 5). We believe that such instabilities could be handled by a more careful treatment of the evolution and/or the level surface, as well as reinitializing the level surface part way through the level set evolution by generalized mean curvature.
- It is absolutely imperative that the scaling function for generalized level set evolution by mean curvature be in the range [0, 1]. Note that the PDE for the level set equation is ill-posed if $b > 0$ and $\psi < 0$. Moreover, note that numerical instabilities result as a consequence of erroneous calculation of the CFL time-stepping bound if $\psi > 1$. For "faster" level set evolution, either increase the value of b, or increase the duration of the level set evolution.
- The last thing to note is that it's entirely possible that the scaling function should scale according to the *two-dimensional* curvature of the zero isocontour, since it is this curvature that we are trying to smoothen. For now, the calculation is implemented according to the three-dimensional mean curvature. As this appears to be effective, anyway, we leave this reimplementation for a future update.

7 Bilinear interpolation of zero contour

This last section is devoted to the interpolation of a zero isocontour for the purpose of being able to "zoom in" on a region of the domain. Since this method was not coded in tandem with an adaptive wavelet scheme, its implementation is likely subject to change; however, the premise is as follows.

Given a space curve that is represented implicitly (*e.g.,* the zero isocontour of a level surface), this method will use bilinear interpolation to calculate a finer discrete approximation to the zero isocontour, and output points in the domain through which the isocontour passes. While other possibilities exist, one useful application

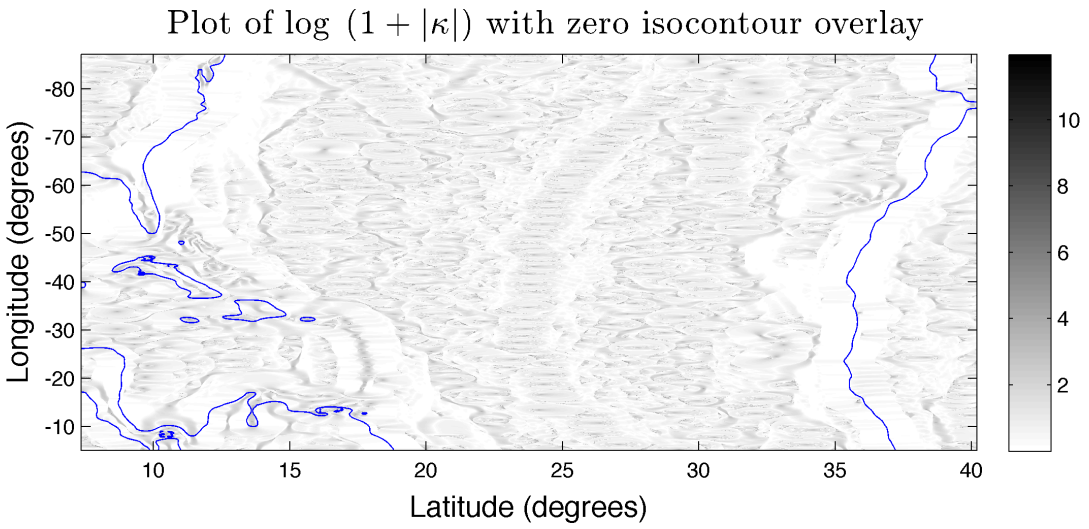

Figure 5: The log-curvature of a mean curvature level set evolution where the curvature term of the level set equation has multiplier proportional to the curvature of the level set. The scaling function used was $\exp(-\beta/\kappa^2)$, with $\beta = 3$, and scaling began at half the total duration (the first half was constant multiplier). The evolution was run on the "large" map (`images/etopo1Large.tif` with $b = 0.04$ for a duration of $t = 1.5$. The data used as the initial conditions for the level set evolution were obtained from the NOAA database ETOPO1 [1]. The latitude and longitude of the region used can be obtained from the `.mat` file containing the data matrix and its domain, `evoByRoC/resdata/mapL_RoCLS.tMax1.5.b0.04.mat`.

of this method would be using these points to generate a new level surface of higher resolution by computing a signed distance function. This level surface could then be used as input to a level set method (see above).

7.1 Kronecker products

Before outlining the method for computing a fine discretization approximation to a zero isocontour using bilinear interpolation, it is first necessary to introduce a few tools that we will use later on to code the numerical implementation.

Given matrices

$$\mathbf{A} = (a_{ij}) = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \qquad \mathbf{B} = (b_{k\ell}) = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix},$$

the Kronecker product of $\mathbf{A}$ and $\mathbf{B}$ is given by

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & a_{12}\mathbf{B} \\ a_{21}\mathbf{B} & a_{22}\mathbf{B} \end{bmatrix}.$$

Next, we define the vec operator as the operator that creates a column vector from the matrix $\mathbf{A}$ by

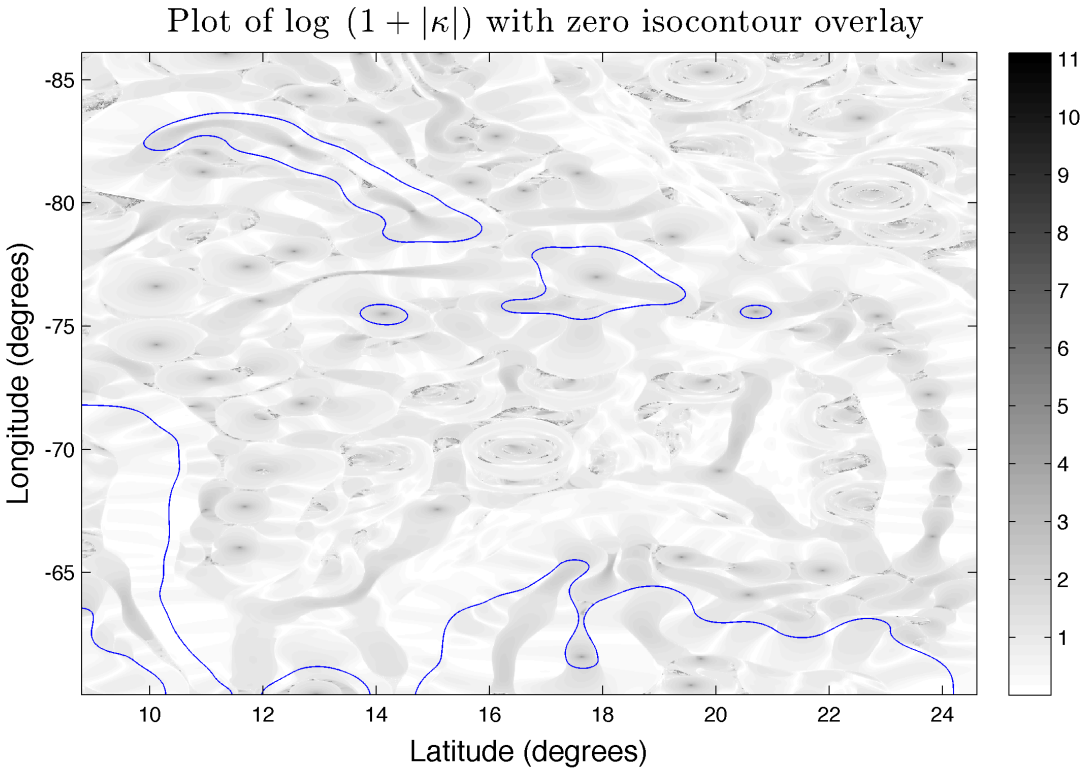

Figure 6: The log-curvature of a mean curvature level set evolution where the curvature term of the level set equation has multiplier proportional to the curvature of the level set. The scaling function used was $\exp(-\beta/\kappa^2)$, with $\beta = 3$, and scaling began at half the total duration (the first half was constant multiplier). The evolution was run on the "small" map (`images/etopo1Small.tif` with $b = 0.04$ for a duration of $t = 2.3$. The data used as the initial conditions for the level set evolution were obtained from the NOAA database ETOPO1 [1]. The latitude and longitude of the region used can be obtained from the `.mat` file containing the data matrix and its domain, `evoByRoC/resdata/mapS_RoCLS.tMax2.3.b0.04.mat`. Note that evolution of the large map with the same parameter settings can be found in the `evoByRoC/resdata/` folder, named with the usual convention.

stacking the column vectors of $\mathbf{A} = [\mathbf{a}_1\mathbf{a}_2\cdots\mathbf{a}_n]$ below one another, so that

$$\operatorname{vec}(\mathbf{A}) = \begin{bmatrix} \mathbf{a}_1 \\ \mathbf{a}_2 \\ \vdots \\ \mathbf{a}_n \end{bmatrix}, \qquad \text{where} \qquad \mathbf{a}_i = \begin{bmatrix} a_{i1} \\ a_{i2} \\ \vdots \\ a_{im} \end{bmatrix}, \quad i = 1, 2, \ldots, n, \text{ and } n, m \in \mathbb{N}.$$

7.2 Derivation of bilinear interpolation

Fix a rectangular domain $D \subset \mathbb{R}^2$ whose corners are given by the points, $\{(x_1, y_1), (x_1, y_2), (x_2, y_1), (x_2, y_2)\}$, where $x_1 < x_2$ and $y_1 < y_2$. Let $f : D \to \mathbb{R}$ be a continuous function. Then we can approximate the value of f at the point $(p, q) \in \operatorname{int} D$ as follows.

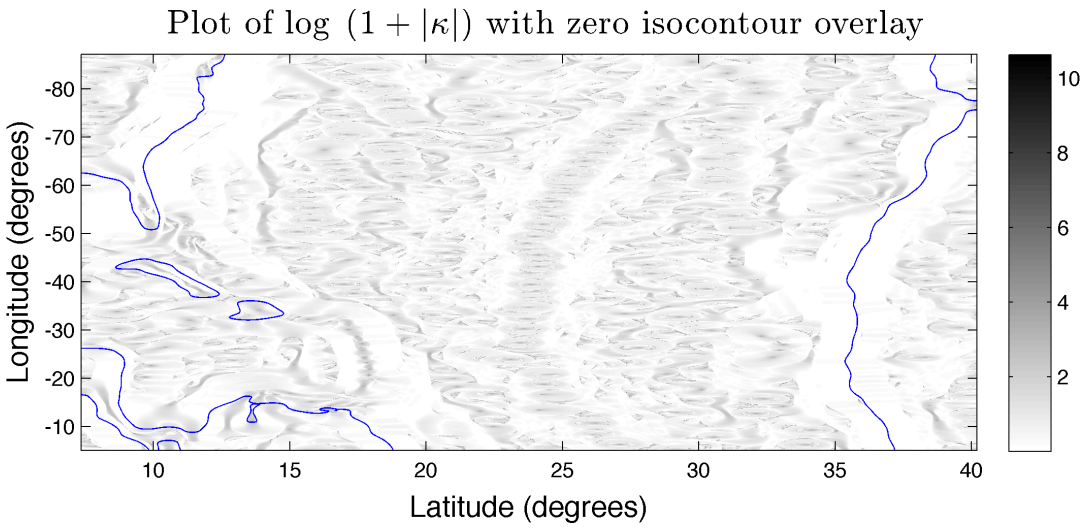

Figure 7: The log-curvature of a mean curvature level set evolution where the curvature term of the level set equation has multiplier proportional to the curvature of the level set. The scaling function used was $\exp(-\beta/\kappa^2)$, with $\beta = 3$, and scaling began at half the total duration (the first half was constant multiplier). The evolution was run on the "large" map (`images/etopo1Large.tif` with $b = 0.04$ for a duration of $t = 4$. The data used as the initial conditions for the level set evolution were obtained from the NOAA database ETOPO1 [1]. The latitude and longitude of the region used can be obtained from the `.mat` file containing the data matrix and its domain, `evoByRoC/resdata/mapL_RoCLS.tMax4.b0.04.mat`.

Approximate $f(p, y_1)$ and $f(p, y_2)$ using the linear interpolations

$$\tilde{f}(p, y_1) = f(x_1, y_1)\frac{x_2 - x}{x_2 - x_1} + f(x_2, y_1)\frac{x - x_1}{x_2 - x_1}$$
$$\tilde{f}(p, y_2) = f(x_1, y_2)\frac{x_2 - x}{x_2 - x_1} + f(x_2, y_2)\frac{x - x_1}{x_2 - x_1}$$

so that

$$f(p, q) \approx \tilde{f}(p, q) = \tilde{f}(p, y_1)\frac{y_2 - y}{y_2 - y_1} + \tilde{f}(p, y_2)\frac{y - y_1}{y_2 - y_1} \tag{11}$$

7.3 Using bilinear interpolation to approximate a zero isocontour

Now, assume that f is a level surface, as per the objective described at the beginning of this section. Then the issue with the above derivation is that the points of f where $f \neq 0$ are nearly irrelevant to us. Namely, we conceive of f as a level surface, because we are generally only interested in the zero isocontour of f. In particular, we wish to approximate a finer discretization of those points $(x, y) \in \mathbb{R}^2$ such that $f(x, y) = 0$. Consequently, assume that $f^{-1}(\{0\})$ is infinite (so that the isocontour is non-trivial). We now relax our criteria and approximate the isocontour (a space curve) by instead solving for all points $(x, y) \in \mathbb{R}^2$ such that $\tilde{f}(x, y) = 0$. We start by expanding the previous formula in Equation 11 to obtain

$$0 = \tilde{f}(p,q) = \frac{1}{(x_2-x_1)(y_2-y_1)}\left(f(x_1,y_1)(x_2-x)(y_2-y) + f(x_1,y_2)(x_2-x)(y-y_1) + f(x_2,y_1)(x-x_1)(y_2-y) + f(x_2,y_2)(x-x_1)(y-y_1)\right). \tag{12}$$

Thus, since each term in the above equation can be expanded as in Equation 13,

$$(x_2-x)(y-y_1) = -x_2y_1 + y_1x + x_2y - xy \tag{13a}$$
$$(x_2-x)(y_2-y) = x_2y_2 - y_2x - x_2y + xy \tag{13b}$$
$$(x-x_1)(y-y_1) = x_1y_1 - y_1x - x_1y + xy \tag{13c}$$
$$(x-x_1)(y_2-y) = -x_1y_2 + y_2x + x_1y - xy, \tag{13d}$$

it is possible to re-write Equation 12 in the form

$$\tilde{f}(x,y) = a + bx + cy + dxy = 0,$$

where a, b, c, d are linear combinations of $f(x_i, y_j)$, $i, j = 1, 2$ and where the latter inequality follows by assumption. Rearranging this equation in terms of x, then of y:

$$y = -\frac{a+bx}{c+dx}, \text{ for } c + dx \neq 0 \quad \text{and} \quad x = -\frac{a+cy}{b+dy}, \text{ for } b + dy \neq 0.$$

Consequently, by forming the matrices

$$\mathbf{X} = \begin{bmatrix} x_2 & -x_1 \\ -1 & 1 \end{bmatrix} \quad \text{and} \quad \mathbf{Y} = \begin{bmatrix} y_1 & -y_2 \\ -1 & 1 \end{bmatrix},$$

it is easily verified that

$$(\mathbf{X} \otimes \mathbf{Y})\,(\text{vec}\mathbf{F}) = \begin{bmatrix} a & c & b & d \end{bmatrix}^T,$$

where

$$\mathbf{F} = (f(x_i, y_j))_{i,j=1}^2 = \begin{bmatrix} f(x_1,y_1) & f(x_2,y_1) \\ f(x_1,y_2) & f(x_2,y_2) \end{bmatrix}.$$

Thus, assuming that $c + dx \neq 0$, we can write the equation of the approximation to the zero isocontour of f through D as

$$y = -\frac{a+bx}{c+dx},$$

while if the isocontour happens to pass vertically through D, we can instead write

$$x = -\frac{a+cy}{b+dy},$$

assuming that $b + dy \neq 0$.

7.4 Implementation in MATLAB

We now describe a generalization of the previous method, implemented in MATLAB to approximate the zero isocontour of an implicitly represented level surface on a finite regular grid domain.

```
function [Xd Yd] = bilinZC(x, y, F, N, varargin)
% BILINZC - Uses bilinear interpolation to calculate approximately the zero↩
    isocontour of a given function.
%
% The function is represented by the values of the matrix F,
% corresponding to the values of the function on the points
% of the rectangular domain, which is implicitly represented by the
% vectors x and y.
%
% F(1,1) corresponds to (x(1), y(1)); F(1,end) with (x(end), y(1));
% F(end, 1) with (x(1), y(end)); F(end, end) with (x(end), y(end))
%
% N represents the maximum number of points to interpolate in
% each domain "box", where a box is given by four mutually
% neighbouring elements of the matrix F. The line segments
% resulting from the interpolation will not, in general, be of
% uniform length; in particular, the points are evenly spaced
% in the x direction, so that the arc lengths may differ
% greatly between pairs of points.
%
% Functionality exists for a fifth argument. This argument is a
% Boolean (represented as 1 or 0), corresponding to whether or
% not to display debugging information. Default assumption no
% debugging info.
%
% As future update: Instead of spacing points evenly along the
% x-axis, space points according to the arc length of the zero
% contour, so that the length of each line segment defined by two
% points calculated on the isocontour is equal.
```

The method `bilinZC` takes as input: a function `F`, implicitly represented as a matrix; vectors `x` and `y`, whose Cartesian product is the domain of F; a number `N` representing the maximum number of points at which to interpolate in each grid box; and one optional argument, a Boolean value, which, if set to 1 will display debugging information for the method. The method outputs two vectors, `Xd` and `Yd`, whose entries correspond to the x and y values, respectively, of the points in the domain through which the zero isocontour passes.

The method operates by computing the zero isocontour for each "grid box" in the input matrix `F` that "crosses" the $z = 0$ plane. That is, for an input matrix $\mathbf{F}$, define the (k, ℓ)-th grid box as

$$\mathbf{B}_{k\ell} := \begin{bmatrix} F_{k,\ell} & F_{k,\ell+1} \\ F_{k+1,\ell} & F_{k+1,\ell+1} \end{bmatrix}.$$

A grid box crosses the $z = 0$ plane if, for example, two corners are positive, and two are negative; however, it should be noted that this is not the only scenario in which a grid box crosses the $z = 0$ plane. In any such grid box, the zero isocontour is represented by as a maximum of `N` discrete points that are spaced evenly in the x direction (if $c + dx = 0$ for any such points, x, then the x values corresponding to the isocontour are recomputed using points spaced evenly in the y direction).

```
% Ensure that x and y are row vectors
  if size(x, 1)>1 & size(x,2)==1
    x = x.';
  end
  if size(y,1)>1 & size(y,2)==1
    y = y.';
  end

  % Initialize return variables
  Xd = [];
  Yd = [];

  % static information
  sy = length(y);
  sx = length(x);
```

For this method to function correctly, it is necessary that `x` and `y` are row vectors. If column vectors are passed to the method, then their transpose is stored to these variables and the method continues. For the sake of good coding practices, we initialize the vectors `Xd` and `Yd`, which will be returned by the method. These vectors store the x and y locations of the approximation to the zero isocontour.

```
  % find boxes framed by elements where a zero contour will exist
  % (i.e., we actually need the function to cross zero in order
  % for there to be a zero contour. Find these locations).
  %
  % Define a matrix whose elements are linearly independent (these
  % elements 1, 10, 100, 1000 were chosen more or less
  % arbitrarily. What's important is that each element is a
  % different magnitude than the others).
  A = [1 100; 10 1000]; % elements are L.I. by magnitude

  % example of how the convolution works for the box
  % {1,3,7,9} \in F
  % for F = [1 3 5; 7 9 13]: C_11 = 1*1 + 7*10 + 3*100 + 9*1000
  C = abs(conv2(sign(F), A, 'valid'));
  % list of element types we don't need to examine
  nonZero = [1111, 1110, 1101, 1011, 111];
  % find members of zero boxes
  LIA = ~ismember(C, nonZero);
```

The purpose of the above code chunk may not be immediately clear. In summary, we only the method to attempt to calculate the zero isocontour through a grid box when the function actually crosses zero through that grid box. This section of the code uses the sign values of `F` to determine in which boxes `F` passes through zero. Firstly, we define a matrix `A`, whose elements are each different orders of magnitude than the others. This matrix is convolved with the signs of the function `F`. By the choice of the values of `A`, it is easy to verify that those entries of `C` which do not cross zero are those contained in `nonZero`. To determine which elements do have a value belonging to this set and which do not, we define a binary matrix `LIA` of the same size as `C`, whose elements are equal to 1 if their value is not contained in the set `nonZero`, and 0 otherwise.

```
% store row and column vectors as  variables to allow
% references to subsets
colvec = 1:sx-1;
rowvec = 1:sy-1;

% iterate over elements of F
% ((selected columns of F containing nontrivial zero contour
% points))
for k = colvec(any(LIA))
  dx = x(k+1)-x(k);

  % set up desired domain points for current column
  xd = linspace(x(k), x(k+1), N+1);
  % handle the case when we're in the last column (to allow
  % points along far right boundary.
  if ~(k==sx-1)
    xd = xd((2:end)-1);
  end
    ⋮
```

In the above code chunk, we begin by creating reference variables for the rows and columns of the matrix `F`, to be used when defining grid boxes in each loop iteration, as well as the iterates of the loops themselves. The first loop is initialized over all columns of `F` in which the corresponding columns of `LIA` have a nonzero element. This means that the loop only iterates over columns of `F` in which there is at least one grid box in which to interpolate.

On each iteration of the loop the x bounds of the interpolation remain constant for all grid boxes in that column. Thus, we define the x domain of the interpolation, `xd`, as a set of evenly spaced `N` points. If the loop is in the end-most column of the domain, then `N+1` points are used so that the values on the far edge of the domain can be determined.

```
  % Generate Kronecker matrices
  % ------------------------------------------------------------ -
  % Columns organized as Q_tl, Q_bl, Q_tr, Q_br (i.e., first
  % column is multiplied to top-left value of function, etc.)
  % rows organized as a, c, b, d, where f(x,y)=a+bx+cy+dxy
  %
  % no longer want y(2:end) --- we want to make our choices
  % according to which boxes cross zero.  select from rowvec
  % only the elements in column k of LIA that correspond to
  % boxes where graph(F) crosses zero in non-trivial way. (the
  % plus one is used to reference the "bottom" y points)
  bases = kron([x(k+1), -x(k); -1 1],...
   [-y(rowvec(LIA(:,k))+1).',...
    y(rowvec(LIA(:,k))).'; 1 -1]);
```

This code chunk implements Kronecker products in the same way as described in subsection 7.3. These "bases" will be used to determine the coefficients a, b, c and d for the interpolation of the space curve. A principle difference between this part of the code and the theory described in subsection 7.3 is that, for the current column, all coefficient combinations are computed at once. That is, instead of taking the Kronecker product between two 2×2 matrices, the matrix containing the y values actually contains all y values corresponding to grid boxes

through which the zero isocontour passes. Thus, the x matrix is of the form

$$\mathbf{X} := \begin{bmatrix} x_{k+1} & -x_k \\ -1 & 1 \end{bmatrix},$$

while the matrix containing the y values is of the form

$$\mathbf{Y} := \begin{bmatrix} -y_{\ell+1} & y_\ell \\ -y_{\ell+m_1+1} & y_{\ell+m_1} \\ \vdots & \vdots \\ -y_{\ell+m_1+\ldots+m_n+1} & y_{\ell+m_1+\ldots+m_n} \\ 1 & -1 \end{bmatrix},$$

for $\ell, m_1, \ldots, m_n$ positive integers, n nonnegative, such that $\ell + m_1 + \ldots + m_n + 1 \leq \text{row}(\mathbf{F})$, where $\text{row}(\mathbf{F})$ is the number of rows of the matrix `F`.

```
% number of elements in y matrix; = half the number of
% rows of bases matrix; = number of boxes in current column
nBox = sum(LIA(:, k));

q = 1;
% selected rows of F
for j = rowvec(LIA(:,k))
  M = F(j:j+1, k:k+1);

  dy = y(j+1)-y(j);

  % will have to cleverly reference indices of Kronecker matrix:
  % q is the products of x and y {-x2y1, x2y2, x1y1, -x1y2}
  % nBox+1 is x {x2 -x2 -x1 x1}
  % nBox+1+q is y {y1 -y2 -y1 y2}
  % 2*(nBox+1) = end row is the row of ones.  {-1 1 1 -1}
  basis = bases([q, nBox+1, nBox+1+q, 2*(nBox+1)], :);
  acbd = basis*M(:);
  yd = -(acbd(1) + acbd(3)*xd)./(acbd(2) + acbd(4)*xd);
  % yd returns as +/- Inf for singular denominator
```

The number of elements in $\mathbf{Y}$ is then calculated by summing the column of `LIA` corresponding to current iteration. This number should be equal to precisely half the number of rows of the `bases` matrix. `q` is a counting variable initialized to one. It is used in the iteration through each grid box in the current column.

Similar to above, the `for` loop is defined to iterate over only those rows where `LIA` has value one — that is, over only those grid boxes through which the zero isocontour passes. On each iteration, `j`, a minor matrix `M` is defined, representing the current grid box. A 4×4 matrix `basis` is then defined by correctly referencing `bases`. The matrix `basis` is of the form as seen subsection 7.3. The coefficients a, c, b and d are calculated (and stored in that order) by multiplying `basis` and vec(`M`) (in that order). We then use these coefficients to calculate the y values that correspond to each of the x values in `xd`. These y values are stored in `yd`.

If the denominator of the expression $(a+bx)/(c+dx)$ is singular for any values of `xd`, then the corresponding entries of `yd` will be equal to $\pm\infty$. These values will be removed and corrected for later.

```
      % Debugging info
      if nargin>4 & varargin{1}==1
  % then debug==TRUE
  fprintf('Checking size of Xd...\n');
  fprintf(['Before: %', num2str(floor(log10(size(Xd,2)))), ...
     'd; '], size(Xd,2));
      end
```

If the debugging variable is set to true, then relevant information is printed to standard output.

```
      if j == sy-1
  Xd = [Xd xd(yd >= y(j) & yd <= y(j+1))];
  Yd = [Yd yd(yd >= y(j) & yd <= y(j+1))];
      else
  Xd = [Xd xd(yd >= y(j) & yd < y(j+1))];
  Yd = [Yd yd(yd >= y(j) & yd < y(j+1))];
      end
```

Only those points that lie on the zero isocontour *and* reside within the bounds of the grid box will be added to the vectors `Xd` and `Yd`. Said differently: all points that represent segments of the zero isocontour and reside outside of the boundary of the grid box are not added to the list of values to be returned (note that this takes care of any infinite values obtained earlier).

```
      % Handle case where c+dx==0 --- can't divide by zero!
      % ---------------------------------------------------
      % If there are singularities, then we do the same process
      % as above, but for the inverse function:
      %                x = -(a+cx)/(b+dx)

      % (define domain in inverse space)
      yyd = linspace(y(j), y(j+1), N);
      % (check that denominator of inverse is not singular)
      if acbd(3)~=0 & any(acbd(4)*yyd~=0)
  % (obtain image points in inverse space
  %  (i.e. domain points in regular space))
  xxd = -(acbd(1)+acbd(2)*yyd)./(acbd(3)+acbd(4)*yyd);

  % analogous to the last time this was done
  if k == sx-1
    Xd = [Xd xxd(xxd >= x(k) & xxd <= x(k+1))];
    Yd = [Yd yyd(xxd >= x(k) & xxd <= x(k+1))];
  else
    Xd = [Xd xxd(xxd >= x(k) & xxd < x(k+1))];
    Yd = [Yd yyd(xxd >= x(k) & xxd < x(k+1))];
  end
      end
```

This code chunk functions similarly to above; however, it works inversely, in that it defines points along the y-axis and then computes the corresponding x values for the location of the space curve. This is done to resolve any vertical lines that the zero isocontour may have. Note that this implementation may consequently results in greater than `N` points per grid box being returned (but no more than `2N`).

```
    % increase counter by one
    % (i.e., move focus from box q to box q+1)
    q = q+1;


    % Debugging info
    if nargin>4 & varargin{1}==1
% print update on length of Xd (and thus Yd)
fprintf(['After: %', num2str(floor(log10(size(Xd,2)))), ...
   'd...\n'], size(Xd, 2));
    end
  end
end
```

Lastly, the counter on `q` is increased, so that the loop can move to the next box in the column. If `debug` is true, then the size of the variables being returned is printed to standard output. Once the loop over the current column has completed, the top-level loop moves to the next column, and a new loop is instantiated over all rows matching the criteria outlined previously.

7.5 Example

An example implementation of this method can be found in the `ZCzoom` folder of the `evoLS` library. We include the code and its output figure below.

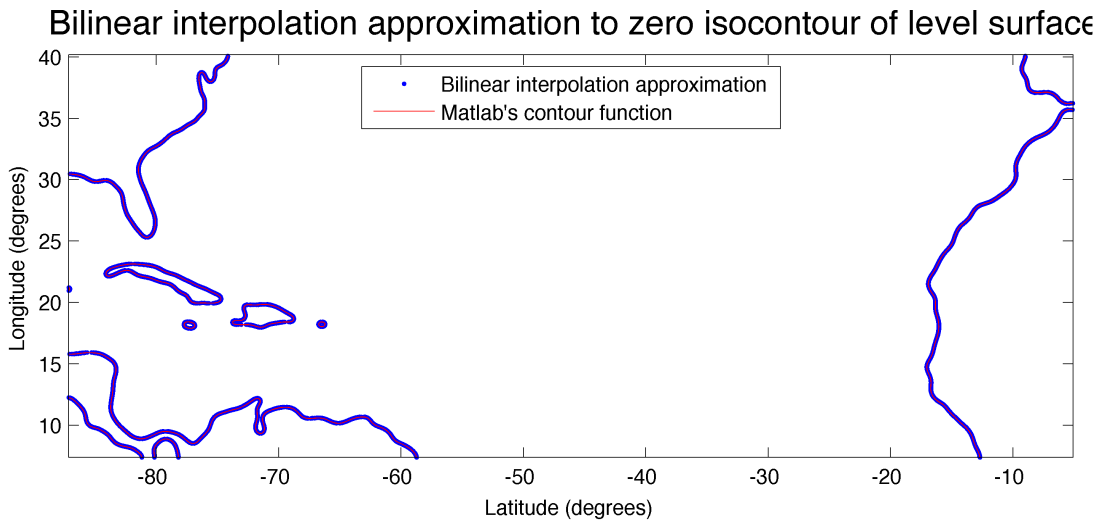

Figure 8: A plot of the bilinear interpolation approximation to the zero isocontour of a level surface (blue points). The data used to generate this image came from `constCurv/resdata/mapL_curvatureLS.tMax2.3.b0.04.mat`. The red line was computed by MATLAB's `contour` function.

```
%% This is an example script of how to use bilinZC to obtain
% the zero contour using bilinear interpolation of a given
% surface function, which is represented as a regular matrix.
```

```
% load file from constCurv/resdata as example.
% contains objects: d_curv, g, description
load('../../parentPath.mat');
load([parentPath, ...
      'evoLS/constCurv/resdata/mapL_curvatureLS.tMax2.3.b0.04.mat']);
% import bilinZC
addpath(genpath([parentPath, 'evoLS/methods']));

% set domain. (Lat and Lon were input incorrectly, so domain
% is weird by default and we have to 'fix' it.)
x = g.vs{2};
y = g.vs{1}(end:-1:1, :);

% set maximum number of points to create in each grid cell box.
% note that resulting grid points will not, in general, be
% evenly spaced. We could change this, but it's more work that
% may not be necessary.
N = 5;

% execute bilinear interpolation of zero contour.
[Xd, Yd] = bilinZC(x, y, d_curv, N);

% Note that MarkerSize can be adjusted to 5 to match the
% linewidth of the contour (5 =/= 5px)
figure(1);
% MATLAB's interpolation
[~, hc] = contour(x, y, d_curv, [0 0], 'r');
% fix aspect ratio
pbaspect([g.shape(2), g.shape(1) 1]);
hold on;
% plot bilinear interpolation
plot(Xd, Yd, '.b', 'MarkerSize', 7);
% change layering of points and lines
uistack(hc, 'top')
hold off;
% UX
legend('Bilinear interpolation approximation', 'Matlab''s contour function'↩
    , 'Location', 'North');
title('Bilinear interpolation approximation to zero isocontour of level ↩
   surface', 'FontSize', 16);
xlabel('Latitude (degrees)');
ylabel('Longitude (degrees)');

makeFigure = 0;
if makeFigure
    set(gcf, 'PaperUnits', 'inches', 'PaperPosition', [0 0 8.5 8.5*g.shape↩
       (1)/g.shape(2)]);
    print(gcf, '-r400', '-dpng', './../README/figures/bilin-interp-approx.↩
       png');
    close all;
end
```

References

[1] Amante, C. & Eakins, B. ETOPO1 1 Arc-Minute Global Relief Model: Procedures, Data Sources and Analysis. Tech. rep., NOAA Technical Memorandum NESDIS NGDC-24, 19 pp, March 2009.

[2] Mitchell, I. A toolbox of level set methods (version 1.1). Tech. Rep. TR-2007-11, Department of Computer Science, University of British Colombia, Vancouver, BC, Canada, Available: http://www.cs.ubc.ca/~mitchell/ToolboxLS/toolboxLS.pdf, June 2007.

[3] Mitchell, I. M. *A Toolbox of Level Set Methods (Version 1.1)*. University of British Colombia, June 2007.

[4] Osher, S. & Fedkiw, R. *Level Set Methods and Dynamic Implicit Surfaces*. Springer-Verlag, 2003.

[5] Reckinger, S. M. *Adaptive Wavelet-Based Ocean Circulation Modeling*. PhD thesis, University of Colorado, 2011.