LSMLIB: User's Guide

Kevin T. Chu
Department of Mechanical & Aerospace Engineering
Princeton University

Maša Prodanović
The Institute for Computational Engineering and Sciences
University of Texas at Austin

Contents

1. Introduction

The level set method is a mature theoretical and numerical method for using implicitly defined surfaces study moving interface problems [1, 2]. In recent years, there has been a surge of interest in the application of level set methods to a wide range of problems. A *very* small sampling of these applications (that the authors are involved with) includes: shape optimization [3, 4, 5, 6], flow in porous media [7, 8], and simulation of dislocation dynamics [9, 10, 11]. The growing number of applications of level set methods signals a need for high-quality software that makes it easier for non-experts to write level set method programs. The Level Set Method Library (LSMLIB) is an effort to deliver state-of-the-art level set method algorithms to the scientific and engineering communities.

LSMLIB provides a collection of level set method algorithms and numerical kernels which support serial and parallel simulation of dynamic interface problems in two and three space dimensions on structured grids. LSMLIB supports solution of the time-dependent and time-independent level set equations. For the time-dependent level set equations, LSMLIB provides functionality to solve both the vector velocity and normal velocity forms of the equations. Support is also provided for extension/extrapolation of field variables off of the zero level set, reinitialization of level set functions, and, for codimension-two problems, orthogonalization of the gradients of pairs of level set functions [1, 2].

The primary goal of the LSMLIB development effort is to deliver an accessible and versatile software package that allow scientists and engineers to rapidly develop high-performance level set method simulations without requiring them to become experts in the details of the underlying numerics. LSMLIB achieves this goal by defining interfaces that only require the user to supply problem specific information. Moreover, interfaces are provided for several common programming languages (C, C++, MATLAB, and Fortran) to suit the user's programming experience and computational needs. High-performance is achieved by using mixed-language programming techniques that make it possible to implement low-level numerical kernels in a fast language, such as Fortran or C.

LSMLIB has been designed to be modular and easy to use. There are several points of entry depending on the particular needs of the specific application. The library is divided into four packages:

- LSMLIB Toolbox Package: core algorithms and numerical kernels for the level set method calculations
- Serial LSMLIB Package: high-level C data structures and algorithmic components for level set method computations on a single processor
- Parallel LSMLIB Package: C++ classes that encapsulate high-level algorithmic components of parallel level set method computations
- LSMLIB MATLAB Toolbox: MATLAB interface for several components of the level set method algorithm

For single processor calculations, the Serial LSMLIB Package and LSMLIB Toolbox Package supply the basic components needed by typical level set method calculations (free of any dependencies on external software libraries). For large scale calculations, the Parallel LSMLIB Package (which is based on the SAMRAI software library developed at Lawrence Livermore National Laboratory) provides support for parallel (and eventually AMR) level set calculations. Finally, LSMLIB also provides a MATLAB interface for several components of level set method calculations, which may be useful for rapid prototyping and moderate-sized production calculations.

1.1. Function Documentation.

This user's manual is still under development and will continue to be improved over time. It currently gives an overview of how to use the LSMLIB library. However, it lacks details about the usage of the many functions provided by LSMLIB. Fortunately, the LSMLIB header files are extensively documented. Usage information for each function is available in the header file in which it is declared. Moreover, function documentation is available in HTML form thanks to the wizardry of the `doxygen` documentation tool. This version of the documentation is available in the doc/lsmlib-dox/html directory.

1.2. Software Dependencies.

The core library (i.e. LSMLIB Toolbox Package) and Serial LSMLIB Package do not depend on any external software libraries. The MATLAB Toolbox and Parallel packages, however, *do* have external dependencies. The MATLAB LSMLIB Toolbox depends on having a MATLAB MEX compiler available. The Parallel LSMLIB Package depends on the SAMRAI library, which in turn requires the HDF5 library (required for visualization and restart) and an MPI library (required to build parallel executables).

1.3. **Building and Installing LSMLIB.** The build procedure for LSMLIB is straightforward on UNIX-based platforms because it follows the "`configure; make; make install`" procedure that is standard for UNIX software. The configure script is automatically generated using the `autoconf` tool to carry out tests to ensure that the system is compatible with LSMLIB and set the build options (*e.g.,* debug vs. optimized mode). A sample of the systems on which LSMLIB has been successfully deployed are: Linux (GCC 4.0.2), Linux (Intel compilers 8.0 & 9.0), Mac OS X (GCC 3.3, IBM XL Fortran compiler), Mac OS X (GCC 3.3, g77), and BlueGene/L (IBM XL compilers).

The hardware and software requirements for LSMLIB are flexible. Building the Parallel LSMLIB Package is optional and may be disabled using the appropriate configure option. The same is true for the LSMLIB MATLAB Toolbox.

For more detailed instructions on building and installing LSMLIB, see the INSTALL.txt file that comes with the LSMLIB distribution.

2. Overview of Level Set Methods

The level set method was originally developed by Osher and Sethian in the 1980s [12] and has since been applied to many problems involving dynamic surfaces and curves. The fundamental idea underlying level set methods is that a surface[1] can be represented implicitly as the zero level set of an embedding function, ϕ. That is, a surface is taken to be the set of points in space where ϕ is zero: $\{\mathbf{x} : \phi(\mathbf{x}) = 0\}$. This viewpoint differs from other theoretical and computational methods which rely on explicit representations of the surface, such as marker/string [2] and front-tracking methods [13, 14].

2.1. Time-dependent Level Set Equations.

For an implicitly represented surface, the dynamics of the surface are determined by the evolution of the embedding function. It is straightforward to derive an evolution equation for the embedding function by considering a particle on the surface. Because the value of ϕ associated with the particle is always zero, the evolution equation for ϕ in a Lagrangian frame moving with the particle is just

$$\frac{d\phi}{dt} = 0. \tag{1}$$

Transforming this equation to an Eulerian frame requires only that we replace the time derivative with a material derivative (or use the chain-rule) to obtain

$$\frac{\partial\phi}{\partial t} + \mathbf{V} \cdot \nabla\phi = 0 \tag{2}$$

where $\mathbf{V}$ is the velocity of the surface, which may be a function of both position and time. This equation gives the vector velocity form of the time-dependent level set equation. To determine the surface at a specified time, one need only solve (2) and locate the points $\mathbf{x}$ where $\phi(\mathbf{x}) = 0$.

For many problems, it is convenient to use a form of the time-dependent level set equation which only involves the normal velocity V_n:

$$\frac{\partial\phi}{\partial t} + V_n\,|\nabla\phi| = 0. \tag{3}$$

The normal velocity form of the level set equation follows directly from (2) by writing $\mathbf{V}$ as $\mathbf{V} = V_n\hat{\mathbf{n}} + V_s\hat{\mathbf{s}} + V_t\hat{\mathbf{t}}$, where $\hat{\mathbf{n}}$ is a unit vector normal to the surface, and $\hat{\mathbf{s}}$ and $\hat{\mathbf{t}}$ are an orthonormal basis for the tangent plane. Because $\nabla\phi$ is orthogonal to the surface, $\nabla\phi \cdot \hat{\mathbf{s}}$ and $\nabla\phi \cdot \hat{\mathbf{t}}$ are both zero, leaving only the normal velocity to contribute to the evolution of the embedding function.

Even when a vector velocity for the surface is available, it can be advantageous to cast the problem in normal velocity form because motion of the interface in the tangential direction does not change the shape of the surface. Furthermore, the use of the special extension velocities techniques described in a later section only apply to problems where the normal velocity is specified.

2.1.1. *Signed Distance Function.*

In the above discussion, there are no constraints placed on the embedding function except that the zero level set coincide with the surface of interest. However, it is often convenient to additionally require that

$$|\nabla\phi| = 1, \tag{4}$$

so that the embedding function is a signed distance function. The main purpose of this restriction on ϕ is to ensure that, near the zero level set, numerical computations of gradients are accurate. Other choices of the embedding function, such as Heaviside functions, have been shown to result in decreased numerical accuracy [15].

2.2. Time-independent Level Set Equation.

For the class of single-signed normal velocity fields (*i.e.,* $V_n > 0$ or $V_n < 0$ everywhere), there is an important alternative way to represent the dynamics of the surface. Rather than choosing the embedding function in such a way that motion of the surface is given by the dynamics of the zero level set, we can choose the embedding function so that its value at a position $\mathbf{x}$ in space is equal to the time when the surface passes through $\mathbf{x}$ from a given surface configuration at the initial time. We shall refer to this embedding function as the arrival function, $T(\mathbf{x})$, in light of its interpretation as the time it takes for the surface to reach each point in space from its initial configuration. Given an arrival

[1]For clarity, we shall focus solely on 2D-surfaces in 3D. However, much of the discussion may be generalized to codimension-one objects in any number of dimensions – in particular, to curves in 2D.

function, the surface at a later time (or earlier time if $V_n < 0$), t, is defined as the set of points where the embedding function takes the value t.

The equation for the arrival function is easily derived by recognizing that $|\nabla T|$ is the rate of change of the arrival time with respect to the change in the distance to the surface. In other words, $|\nabla T|$ is equal to the reciprocal of the normal velocity:

$$|\nabla T| = 1/V_n. \tag{5}$$

If V_n depends only on position and is independent of time, this equation becomes the well-known Eikonal equation [2]. Note that the fact that the normal velocity is single-signed is necessary in order to be able to define the arrival function. If the normal velocity changes sign, then there would be some point in space that the surface passes through more than once. As a result, the arrival function would be an ill-defined function.

2.3. **Extension Velocities.** In the preceding discussion, we have focused on the evolution of the embedding function at the surface. However, the level set equations themselves are partial differential equations that are defined throughout the entire spatial domain. In writing the level set equations, we have implicitly assumed that the velocity field is a defined quantity off of the surface. In order to use level set methods, we must ensure that we have a means for defining the velocity field throughout the spatial domain. This extended velocity field is known as the *extension velocity* [2].

For problems involving the vector velocity form of the time-dependent level set equations, the velocity field is usually defined throughout the entire domain as part of the problem. For example, in fluid flow problems, the fluid velocity is a natural choice for the extension velocity.

For problems where the normal velocity is specified, a good way to define the extension velocity is to extrapolate the velocity off the surface so that it is constant on rays normal to the surface [1, 2]. This goal can be achieved by solving either the time-dependent two-way extrapolation equation [1]

$$\frac{\partial V_n}{\partial t} + \mathrm{sgn}(\phi)\hat{\mathbf{n}} \cdot \nabla V_n = 0 \tag{6}$$

or its steady-state equivalent [2]

$$\nabla V_n \cdot \nabla \phi = 0. \tag{7}$$

The advantage of this choice of extension velocity is that it preserves signed distance functions [2].

2.4. **Field Extension.** For some problems, it may be necessary to a extend field variable, γ, other than ϕ or the V_n off of the zero level set. In these situations, equations (6) and (7) with V_n replaced by γ provide two convenient methods for computing the extension field for γ:

$$\begin{aligned} \frac{\partial \gamma}{\partial t} + \mathrm{sgn}(\phi)\hat{\mathbf{n}} \cdot \nabla \gamma &= 0 \qquad (8) \\ \nabla \gamma \cdot \nabla \phi &= 0. \qquad (9) \end{aligned}$$

2.5. **Reinitialization.** At the beginning of a time-dependent level set method calculation, there may be a need to generate a signed distance function, ϕ, from an arbitrary embedding function. In addition, as a level set method calculation progresses, numerical errors will often cause ϕ to drift away from being a signed distance function (even if the extension velocity is defined as described in the previous section). To deal with both of these issues, we use a process known as *reinitialization*, which essentially computes a signed distance function from a given embedding function.

As with the basic level set equations and extension velocity equations, there are time-dependent and time-independent formulations of the reinitialization procedure. The goal of both formulations is to solve the signed distance function equation (4). The time-dependent approach solves (4) as the steady solution to the equation

$$\frac{\partial \phi}{\partial t} + \mathrm{sgn}(\phi)\left(|\nabla \phi| - 1\right) = 0. \tag{10}$$

The time-independent approach solves (4) directly by treating it as a time-independent level set equation of the form (5) with a constant normal velocity of 1.

Each approach has its advantages and disadvantages. Equation (10) is convenient because it can be solved using the same machinery developed for solving the time-dependent level set equations and does not require the zero level set to be explicitly computed. Unfortunately, although the mathematical solution of

(10) guarantees that the zero level set will remain stationary, errors in the numerical solution can cause the location of the surface drift during the reinitialization process [1, 2]. The time-independent approach does not suffer from the drift problem and can be solved using an elegant algorithm known as the fast marching method (described in the Level Set Method Algorithms section). However, it requires explicit computation of the zero level set and is difficult to parallelize.

2.6. **Orthogonalization.** For codimension-two problems, it is often desireable to maintain orthogonality between the gradients of the two level set functions whose zero level sets intersect to define the 3D curve. This goal is achieved by alternatvely solving the orthogonalization equations for the pair of level set functions ϕ and ψ:

$$\frac{\partial \phi}{\partial t} + \text{sgn}(\psi) \frac{\nabla \phi \cdot \nabla \psi}{|\nabla \psi|} = 0 \tag{11}$$

$$\frac{\partial \psi}{\partial t} + \text{sgn}(\phi) \frac{\nabla \psi \cdot \nabla \phi}{|\nabla \phi|} = 0. \tag{12}$$

As with the time-dependent reinitialization equation, these equations can be solved using the same techniques used to solve the time-dependent level set equations. It should be noted that orthogonalization procedures are still an area of research, so the orthogonalization functionality provided by LSMLIB may lack the robustness of other parts of the library.

2.7. **Further Reading.** For a deeper discussion of level set methods, we refer the reader to *Level Sets Methods and Dynamic Implicit Surfaces* by S. Osher and R. Fedkiw and *Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science* by J. A. Sethian. Both books provide an excellent discussion of the theory underlying level set methods and give many interesting example applications.

3. Serial LSMLIB Package

For level set method computations on a single processor, LSMLIB provides a C-interface that contains high-level data structures and algorithmic components. For time-dependent level set method calculations, the Serial LSMLIB Package provides support for

- managing the computational grid,
- initializing level set functions,
- imposing boundary conditions, and
- using narrow-band algorithms.

The Serial LSMLIB Package also provides support for time-independent level set method calculations using fast marching methods.

3.1. **Usage.** The Serial LSMLIB Package has been designed to be minimize the time required for an application developer to implement a level set method simulation. The simplest way to develop a level set method simulation based on the serial level set method package is to first write code for computing the velocity field that is used to evolve the level set function and then to use the following procedure in the main program:

(1) Set up grid and data arrays using functions from lsm_grid.h and lsm_data_arrays.h.
(2) Initialize the level set functions using the functions in lsm_initialization2d.h or lsm_initialization3d.h or by writing custom initialization code for your particular application.
(3) Evolve the level set functions in time by
    (a) imposing boundary conditions for all field variables; several common boundary conditions for level set functions are provided in lsm_boundary_conditions2d.h and lsm_boundary_conditions3d.h.
    (b) computing the velocity field for the current time
    (c) computing a stable time step to take; functions for computing stable time steps for advection and normal velocity driven motion (of hyperbolic character) are available in lsm_utilities1d.h, lsm_utilities2d.h, and lsm_utilities3d.h.
    (d) advancing the level set functions by a single time step using the TVD Runge-Kutta time integration functions provided in the LSMLIB Toolbox Package
    (e) periodically reinitializing the level set functions
(4) Postprocess and visualize results.

The Serial LSMLIB Package also provides support for fast marching method calculations. To carry out a fast marching method calculation, step 3 in the above algorithm is replaced by a call to the appropriate computeDistanceFunction(), computeExtensionFields(), or solveEikonalEquation() function in lsm_fast_marching_method.h.

Example serial level set method calculations implemented using these procedures are provided in the examples/serial directory. The reinitialization_example also demonstrates the use of lsm_options which may be helpful for managing the options/parameters passed into a simulation via an input file.

4. Parallel LSMLIB Package

For parallel level set method calculations, the Parallel LSMLIB Package provides a C++ interface that contains a collection classes which encapsulate the high-level algorithmic components of a parallel level set method computation. To reduce software complexity for the user, the C++ interface has been designed so that the user only needs to interact with four or five classes to write a level set method program. The remaining classes in the Parallel LSMLIB Package are available for advanced users who may have special needs.

LSMLIB's the parallel capabilities are provided by the Structured Adaptive Mesh Refinement Application Infrastructure (SAMRAI) developed and maintained by the Center for Applied Scientific Computing at Lawrence Livermore National Laboratory. SAMRAI handles many of the important issues arise parallel computing is used to solve partial differential equations (*e.g.,* domain decomposition and load balancing). LSMLIB's robustness and scalability for parallel computation is in large part due to the high quality of the SAMRAI library. By leveraging the SAMRAI library, the Parallel LSMLIB Package is also able other useful functionality, such as input and restart capabilities.

Because the Parallel LSMLIB Package is based on SAMRAI, LSMLIB provides parallel support for distributed memory systems based on the message passing interface (MPI) standard. No special support is available for shared memory architectures beyond any features that are provided by the MPI library that a parallel LSMLIB program is linked against.

4.1. **Usage.** The Parallel LSMLIB Package has been designed to be useable with minimal user effort. The simplest way to develop a level set method simulation based on the parallel level set method package is to make use of the LevelSetMethodAlgorithm class and provide problem specific initialization, boundary conditions, and velocity field calculations by implementing concrete subclasses of the LevelSetMethodPatchStrategy and LevelSetMethodVelocityFieldStrategy abstract base classes. Once these concrete subclasses have been implemented, a level set method calculation can be carried out as follows:

1. Create a LevelSetMethodAlgorithm object using the constructor that takes pointers to LevelSetMethodVelocityFieldStrategy and LevelSetMethodPatchStrategy objects as arguments. Pass pointers to instances of the concrete subclasses for these arguments.
2. Begin the level set method calculation by invoking the LevelSetMethodAlgorithm::initializeLevelSetMethodCalculation() method. This will initialize the data on the PatchHierarchy.
3. Integrate the level set functions in time by invoking the LevelSetMethodAlgorithm::advanceLevelSetFunctions() method. Stable time step sizes can be computed for each time step using the LevelSetMethodAlgorithm::computeStableDt() method.
4. Access level set function data and other integration data through several accessor method such as getPhiPatchDataHandle(), getStartTime(), etc. These methods are documented in the LevelSetMethodAlgorithm class header file.

Example parallel level set method simulations implemented using this procedure are provided in the examples/parallel directory. For details on how to use the LevelSetMethodAlgorithm class, see any of the main() programs. For details on how to access data in a SAMRAI PatchHierarchy, pay special attention to the TestLSM_2d_VelocityFieldModule and TestLSM_2d_PatchModule classes. In particular, the following methods demonstrate the typical way to access data:

- TestLSM_2d_VelocityFieldModule::computeVelocityFieldOnLevel()
- TestLSM_2d_PatchModule::initializeLevelSetFunctionsOnPatch()

These methods also illustrate how to interface C/C++ and Fortran code. It should be emphasized, that there it is NOT necessary that the numerical kernels be implemented in Fortran. To carry out the same computation in C/C++, it is straightforward to access the data in the arrays and compute positions by using the bounding box and geometry information. However, it is ESSENTIAL that the data arrays be stored and accessed in Fortran order (i.e. column-major order).

For more details about the LevelSetMethodAlgorithm (particularly, advanced ways to use it), see the LevelSetMethodAlgorithm class header file.

4.2. **Input Parameters.** The LevelSetMethodAlgorithm accepts many input parameters that may be used to control the level set method calculation. These are passed into the level set method application program through an input file with the following format:

```
InputDatabase1 {
  input_parameter1 = 1
  ...
}

InputDatabase2 {
  SubDatabase1 {
    ...
  }
  SubDatabase2 {
    ...
  }
}
```

Note the nested structure of the database. White spaces do not matter. The input parameters for the LevelSetMethodAlgorithm are described in detail in the class header file. As an example, see the Sample Input File or the input files contained in the examples/parallel/ directory.

4.3. **NOTES.** The naming conventions used in the parallel level set method package are based on common object-oriented design patterns. For more details, please refer to *Design Patterns: Elements of Reusable Object-Oriented Software* by E. Gamma, R. Helm, R. Johnson, and J. Vlissides.

5. LSMLIB MATLAB Toolbox

For rapid prototyping and moderate sized level set method calculations, LSMLIB provides a MATLAB interface that supplies several MATLAB functions that support level set method calculations. Specifically, the LSMLIB MATLAB Toolbox currently provides routines for:

- time evolution of level set functions;
- calculation of high-order ENO/WENO spatial derivatives;
- TVD Runge-Kutta time integration;
- reinitialization of level set functions;
- computation of the distance function and extension fields using fast marching methods; and
- solution of Eikonal equations using fast marching methods.

The MATLAB interface allows a user to take advantage of MATLAB's powerful data analysis and visualization capabilities to obtain results quickly and easily. High-performance of these functions is achieved by using MATLAB MEX-files to interface to LSMLIB's numerical kernels.

5.1. **Usage.**

The LSMLIB MATLAB Toolbox is designed to support rapid prototyping and development of moderate resolution level set method calculations. It is easy to carry out level set method calculations in MATLAB using the LSMLIB MATLAB Toolbox.

For calculations involving the time evolution of the level set function, the following is a typical structure for the program:

(1) Initialize the level set function.
(2) Integrate the level set functions in time by:
    (a) Filling ghostcells for the level set function and other field variables in the calculation.
    (b) Compute the velocity field used to advance the level set function. The high-order spatial derivatives provided by the LSMLIB MATLAB Toolbox can be helpful for this stage of the computation.
    (c) Advance the level set function in time using one of the time advance functions.
(3) Plot results using MATLAB's built in graphics functionality.

To use fast marching methods to compute distance functions or extension fields:

(1) Initialize the level set function (and auxilliary fields, if present).
(2) Compute the distance function (and extension fields, if desired) using the fast marching method functions provided by the LSMLIB MATLAB Toolbox.

To use fast marching methods to solve Eikonal euqations: fields:

(1) Compute the normal velocity field.
(2) Solve the Eikonal equation using the fast marching method functions provided by the LSMLIB MATLAB Toolbox.

Example level set method simulations implemented using the LSMLIB MATLAB Toolbox are provided in the examples/matlab directory.

6. LSMLIB Toolbox

The core functionality of LSMLIB is contained in the LSMLIB Toolbox Package. It provides routines for:

- calculation of high-order ENO/WENO spatial derivatives;
- TVD Runge-Kutta time integration;
- computing geometry quantities;
- integrating over regions defined by implicit functions;
- computation of the distance function and extension fields using the fast marching method;
- solution of the Eikonal equation using the fast marching method;
- computing the right-hand side of specific level set method partial differential equations; and
- general utility functions (e.g. computing a stable time step size using a CFL based criterion and computing the max norm of the difference of two field variables).

These routines form the basic numerical kernels and algorithmic components required for typical level set method calculations. Usage of these routines is straight-forward for the application developer who is familiar with the general level set method algorithm. However, the typical user will find it more convenient to use the higher level functionality provided by the Serial LSMLIB Package, Parallel LSMLIB Package, or LSMLIB MATLAB Toolbox.

While all of these routines (with the exception of the fast marching method routines) are implemented in Fortran 77 for performance reasons, C-interfaces are provided. The function signatures are provided in the following header and Fortran files.

6.0.1. *C Header Files.*

- lsm_boundary_conditions1d.h, lsm_boundary_conditions2d.h, lsm_boundary_conditions3d.h,
- lsm_calculus_toolbox.h
- lsm_fast_marching_method.h
- lsm_field_extension1d.h, lsm_field_extension2d.h, lsm_field_extension3d.h
- lsm_geometry1d.h, lsm_geometry2d.h, lsm_geometry3d.h
- lsm_level_set_evolution1d.h, lsm_level_set_evolution2d.h, lsm_level_set_evolution3d.h
- lsm_reinitialization1d.h, lsm_reinitialization2d.h, lsm_reinitialization3d.h
- lsm_spatial_derivatives1d.h, lsm_spatial_derivatives2d.h, lsm_spatial_derivatives3d.h
- lsm_tvd_runge_kutta1d.h, lsm_tvd_runge_kutta2d.h, lsm_tvd_runge_kutta3d.h
- lsm_utilities1d.h, lsm_utilities2d.h, lsm_utilities3d.h

6.0.2. *Fortran 77 source files.*

- lsm_boundary_conditions1d.f, lsm_boundary_conditions2d.f, lsm_boundary_conditions3d.f,
- lsm_calculus_toolbox.f
- lsm_field_extension1d.f, lsm_field_extension2d.f, lsm_field_extension3d.f
- lsm_geometry1d.f, lsm_geometry2d.f, lsm_geometry3d.f
- lsm_level_set_evolution1d.f, lsm_level_set_evolution2d.f, lsm_level_set_evolution3d.f
- lsm_reinitialization1d.f, lsm_reinitialization2d.f, lsm_reinitialization3d.f
- lsm_spatial_derivatives1d.f, lsm_spatial_derivatives2d.f, lsm_spatial_derivatives3d.f
- lsm_tvd_runge_kutta1d.f, lsm_tvd_runge_kutta2d.f, lsm_tvd_runge_kutta3d.f
- lsm_utilities1d.f, lsm_utilities2d.f, lsm_utilities3d.f

6.1. **NOTES.**

- Depending on the software development platform, it may be necessary to redefine the C-macros that link the C/C++ function name to the fortran object code name.
- ALL toolbox routines assume that field data are stored in Fortran order (i.e. column-major order).
- The fast marching method package is designed to be somewhat extensible to higher dimensions and higher-order numerical schemes. However, this feature is experimental and the package is currently limited to dimensions <= 8.

References

[1] S. Osher and R. Fedkiw. *Level Sets Methods and Dynamic Implicit Surfaces.* Springer-Verlag, New York, NY, 2003.

[2] J. A. Sethian. *Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science.* Cambridge University Press, 1999.

[3] S. J. Osher and F. Santosa. Level set methods for optimization problems involving geometry and constraints I. Frequencies of a two-density inhomogeneous drum. *J. Comp. Phys.*, 171:272–288, 2001.

[4] O. Alexandrov and F. Santosa. A topology-preserving level set method for shape optimization. *J. Comp. Phys.*, 204:121–130, 2005.

[5] G. Allaire, F. Jouve, and A.-M. Toader. Structural optimization using sensitivity analysis and a level-set method. *J. Comp. Phys.*, 194:363–393, 2004.

[6] Y. Jung, K. T. Chu, and S. Torquato. A variational level set approach for surface area minimization of triply periodic surfaces. *J. Comp. Phys.*, 2006. In press.

[7] M. Prodanović and S. L. Bryant. Investigating pore scale configurations of two immiscible fluids via the level set method. In *Computational Methods in Water Resources XVI Conference*, Copenhagen, Denmark, 2006.

[8] M. Prodanović and S. L. Bryant. A level set method for determining critical curvatures for drainage and imbibition. *J. Colloid Interface Sci.*, 2006. In Press.

[9] Y. Xiang, L.-T. Cheng, D. J. Srolovitz, and W. E. A level set method for dislocation dynamics. *Acta Mater.*, 51:5499–5518, 2003.

[10] Y. Xiang, D. J. Srolovitz, L.-T. Cheng, and W. E. Level set simulations of dislocation-particle bypass mechanisms. *Acta Mater.*, 52:1745–1760, 2004.

[11] S. S. Quek, Y. Xiang, Y. W. Zhang, D. J. Srolovitz, and C. Lu. Level set simulation of dislocation dynamics in thin films. *Acta Mater.*, 54:2371–2381, 2006.

[12] S. Osher and J. A. Sethian. Fronts propagating with curvature dependent speed: Algorithms based on hamilton-jacobi formulations. *J. Comp. Phys.*, 79:12–49, 1988.

[13] S. O. Unverdi and G. Tryggvason. A front-tracking method for viscous, incompressible, multi-fluid flows. *J. Comp. Phys.*, 100:25–37, 1992.

[14] G. Tryggvason, B. Bunner, A. Esmaeeli, D. Juric, N. Al-Rawahi, W. Tauber, J. Han, S. Nas, and Y.-J. Jan. A front-tracking method for the computations of multiphase flow. *J. Comp. Phys.*, 169:708–759, 2001.

[15] W. Mulder, S. Osher, and J. A. Sethian. Computing interface motion in compressible gas dynamics. *J. Comp. Phys.*, 100:209–228, 1992.