The Flexible, Extensible and Efficient Toolbox of Level Set Methods

Ian M. Mitchell*

July 3, 2007

Submitted to the *Journal of Scientific Computing*.
Please do not redistribute.

Running Head: The Toolbox of Level Set Methods

Abstract

Level set methods are a popular and powerful class of numerical algorithms for dynamic implicit surfaces and solution of Hamilton-Jacobi PDEs. While the advanced level set schemes combine both efficiency and accuracy, their implementation complexity makes it difficult for the community to reproduce new results and make quantitative comparisons between methods. This paper describes the Toolbox of Level Set Methods, a collection of MATLAB routines implementing the basic level set algorithms on fixed Cartesian grids for rectangular domains in arbitrary dimension. The Toolbox's code and interface are designed to permit flexible combinations of different schemes and PDE forms, allow easy extension through the addition of new algorithms, and achieve efficient execution despite the fact that the code is entirely written as m-files. The current contents of the Toolbox and some coding patterns important to achieving its flexibility, extensibility and efficiency are briefly explained, as is the process of adding two new algorithms. Code for both the Toolbox and the new algorithms is available from the Web.

Keywords: numerical software; level set methods; Hamilton-Jacobi equations; dynamic implicit surfaces; reproducible research.

*Department of Computer Science, University of British Columbia. Postal: 2366 Main Mall, Vancouver, BC, Canada V6T 1Z4. Voice: 604-822-2317. Fax: 604-822-5485. Email: mitchell@cs.ubc.ca Web: http://www.cs.ubc.ca/~mitchell

1 Introduction

Level set methods [29] have proved a popular technique for dynamic implicit surfaces and approximation of the time-dependent Hamilton-Jacobi (HJ) partial differential equation (PDE), as evidenced by the many survey papers, textbooks and edited collections devoted to their development; for example [24, 26–28, 34, 35]. The ease with which the earliest schemes could be implemented in two or three dimensions was a key facet of their popularity, and the dimensional flexibility of many advanced schemes remains a major asset. However, algorithm simplicity has largely lost out in recent work to the competing demands of efficiency and accuracy. From a scientific computing perspective improving either or both of these is generally worth the increased complexity—users always have the option of going back to the simpler schemes—but there are two unintended and potentially adverse consequences of advanced methods. The first is that scientists and engineers who might be interested in using dynamic implicit surfaces in their application field but who are not experts at level set methods may give up when they are either unable to recreate with simple schemes the impressive published results generated by the advanced schemes, or they are overwhelmed by the details of those advanced schemes. The second is that designers of new schemes find it increasingly difficult to promulgate their results in a reproducible manner and to provide quantitative comparisons with alternative methods because of the complex algorithm and software infrastructure underlying each new advance.

The Toolbox of Level Set Methods (ToolboxLS) is designed to address these concerns. Its goal is to provide a collection of routines which implement the basic level set algorithms in MATLAB[1] on fixed Cartesian grids for rectangular domains in arbitrary dimension. In using MATLAB we seek to minimize not execution time, but the combination of coding, debugging and execution time. In our experience the visualization, debugging, data manipulation and scripting capabilities of MATLAB make construction of numerical code so much simpler, when compared to compiled languages like C++ or Fortran, that the increase in execution time is quite acceptable when designing new algorithms or exploring proof-of-concept for new applications. If the algorithms should prove successful but execution time and/or the restrictive class of Cartesian grids remains an impediment to adoption, a side benefit of ToolboxLS is that all of the source code is available so that recoding in a compiled language is straightforward.

The Toolbox has a lengthy, indexed user manual [18], and users interested in applying level set methods to applications will probably find this manual a good place to get started. In this paper we instead explore the features of the Toolbox

[1] MATLAB is a product and trademark of The Mathworks Incorporated of Natick, Massachusetts. For more details see `http://www.mathworks.com/products/matlab/`. ToolboxLS was developed by the author of this paper, and is neither endorsed by nor a product of The Mathworks.

that make it suitable for developers of new level set methods. In section 2, we briefly describe the contents of TOOLBOXLS version 1.1: the kernel routines that provide a mix and match implementation of level set methods; the coding patterns we use to achieve efficiency, flexibility and dimensional independence; and the many documented examples we have recreated from the literature. To demonstrate the extensibility of TOOLBOXLS, in section 3 we describe two newly implemented schemes. We extend a class of SSP RK integrators [38] to handle time-dependent operators, and demonstrate their efficiency on several dynamic implicit surface problems. Then we extend a new monotone motion by mean curvature spatial approximation [21, 41] to handle Cartesian grids with variable Δx, provide some additional order of accuracy analysis, and demonstrate that while the new scheme's quantitative accuracy is poor, it provides qualitatively reasonable results in less time than the standard centered difference approximation. In the spirit of the reproducible research initiative, the code for both the base Toolbox and the new additions are available as separate downloads from [16].

1.1 Limitations

The decision to restrict the Toolbox to dense solutions on Cartesian discretizations of rectangular domains simplifies many aspects of the implementation. A major benefit is that a relatively high degree of computational efficiency can be achieved despite the fact that all of the code is written in m-files. On these grids the level set functions, their derivatives, and the spatially varying problem data can be represented by dense MATLAB arrays of appropriate dimension. Applying operators to these arrays can then be vectorized in the MATLAB sense; for example, a single short m-file command like `speed .* data` becomes a loop performing a multiplication at every node in the grid. We reap three benefits from such code: (1) it is dimensionally independent, (2) any computational overhead for interpreting the command is swamped by the huge number of floating point operations that are subsequently issued, and (3) in most MATLAB installations, such an operator will invoke compiled code highly optimized for both cache and processor efficiency. In fact, such MATLAB operators will often outperform equivalent looping code in naively written and compiled C/C++ or Fortran. Consequently, the speed benefit of compiled versions of the Toolbox algorithms is likely to be fairly small for problems where the PDEs are solved globally on Cartesian grids.

Of course, this restriction is also the primary limitation of the Toolbox. Unstructured and adaptive grids are not part of TOOLBOXLS and are never likely to be included, because the spatially varying nature of their nodes' connectivity gives rise to irregular data access patterns. Despite the addition of just-in-time compilation to recent versions of MATLAB, m-files that include such irregular data access patterns are often orders of magnitude slower to execute than those

which access the same amount of data organized in a dense regular array. On the other hand, numerical schemes for these grids are often simple—spatial refinement is used to improve accuracy rather than complex schemes with high order convergence rates—so the lack of support for these grids does not significantly detract from the goals of the Toolbox.

Given that the Toolbox is constrained to Cartesian grids, a more glaring omission is the lack of support for narrowband [4] or local level set [31] algorithms. These algorithms focus their computational effort only on the nodes near the interface, so they can often achieve the same dynamic implicit surface evolution in less time than global algorithms (such as those in the Toolbox) despite the overhead of tracking this constantly evolving set of nodes. While localized algorithms generally present a clear win in compiled implementations, in a MATLAB m-file implementation it is not clear whether the benefits of working only on a subset of nodes would offset the costs of identifying that subset (additional discussion can be found in section 2.4). What is clear is that MATLAB-style vectorization in this situation would require significantly more complex implementations throughout the kernel. Because minimum execution time is not the primary objective of TOOLBOXLS, algorithmic simplicity has been chosen over an uncertain speed improvement. Should compiled code ever be added to the Toolbox, localization would become a much more appealing option (see section 4 for additional comments on including compiled code with the Toolbox).

1.2 Other Software Packages

With a version 1.0 release date of July 2004, TOOLBOXLS is to our knowledge the first publicly released code implementing the high accuracy level set algorithms, and it remains the only one that works in any dimension; however, it is no longer the only such package.

For comparison purposes, version 1.1 of TOOLBOXLS has a 140 page indexed user manual, supports ten different types of time-dependent evolution, includes over twenty documented examples, and is implemented with over 120 MATLAB m-files (each with its own help entry). TOOLBOXLS is licensed under a modified version of the ACM Software Copyright and License Agreement for free non-commercial use; we are investigating switching to a similar Creative Commons license [9].

We are aware of the following other packages:

- The Level Set Method Library (LSMLIB) [5] supports serial and parallel simulation in dimensions one to three. The code is written in a mixture of C, C++ and Fortran, with MATLAB interfaces to some components. Only two types of time-dependent evolution are presently supported (normal direction and convection), but the algorithms are localized. Fast

marching algorithms for the time-independent PDEs arising in signed distance construction and velocity extension are included, as are routines for computing surface and volume integrals. Three short manuals (overview, users guide and reference) and complete code documentation are part of the download. Version 0.9 contains many examples (although they are not documented in the manuals) and several hundred files. LSMLIB does not seem to be driven by any particular application field. The software is restricted to noncommercial use.

- The Multivac C++ Library [15] works only in two dimensions. It includes both localized algorithms and fast marching for signed distance construction. Six types of evolution are supported, of which two are forest fire models. A short hypertext user manual and complete code documentation can be found at the web site. Five examples (some with multiple versions) are included, although the user manual contains details on only one. Applications include forest fire propagation, image segmentation, and nanofilm growth. An optional display package and a GUI for image segmentation are available (written in Python with calls to Gnuplot). Version 1.10 includes more than 100 files, and is released under the GNU General Public License (GPL).
- "A Matlab toolbox implementing Level Set Methods" [39] is the most similar of these packages to ToolboxLS, since it is also implemented entirely by Matlab m-files. The application emphasis is on vision and image processing, an important field missing from the set of examples in the current version of ToolboxLS. In keeping with this emphasis, version 1.1 of Sumengen's package supports only two dimensional problems and three types of evolution (normal direction, curvature and/or convection). The restricted problem domain translates to a more compact package of roughly 50 m-files. This package does not seem to have a user manual—although the web site includes a tutorial and set of examples—nor is any licensing arrangement specified.

The package [32] implements fast marching methods, which are used for static (time-independent) HJ PDEs and are quite distinct algorithmically from the level set methods discussed here.

2 Toolbox Design

In this section we discuss the structure of ToolboxLS, with particular attention to how it is designed to be easy to use and to extend while still maintaining reasonably fast execution.

2.1 The Equation

TOOLBOXLS is written with the vision of providing routines to approximate the solution of degenerate parabolic PDEs of the form [8]

$$D_t\phi(t,x) + G(t,x,\phi,D_x\phi,D_x^2\phi) = 0 \text{ for } x \in \Omega \text{ and } t \geq t_0 \tag{1}$$

on domain $\Omega \subseteq \mathbb{R}^d$ and subject to initial and possibly boundary conditions

$$\begin{aligned} \phi(t_0,x) &= \phi_0(x) && \text{for } x \in \Omega, \\ \phi(t,x) &= \phi_{\partial\Omega}(t,x) && \text{for } x \in \partial\Omega \text{ and } t \geq t_0. \end{aligned} \tag{2}$$

We assume that the initial conditions are bounded and continuous and that G satisfies a monotonicity requirement

$$G(t,x,r,p,\mathbf{X}) \leq G(t,x,s,p,\mathbf{Y}), \text{ whenever } r \leq s \text{ and } \mathbf{Y} \leq \mathbf{X}, \tag{3}$$

where $\mathbf{X}$ and $\mathbf{Y}$ are symmetric matrices of appropriate dimension. For such G, there may not exist a classical solution to (1), and so the Toolbox routines are designed to approximate the viscosity solution [6], which is the appropriate weak solution for many problems that lead to equations of the form (1), although it is not the only possible weak solution. Included in the class of degenerate parabolic PDEs are those arising in dynamic implicit surfaces and the time-dependent HJ PDE. A key feature of the viscosity solution of (1) is that under suitable conditions ϕ remains bounded and continuous for all time. This property may not hold for other types of HJ PDE, such as the static equations arising in minimum time to reach problems (although see section 2.5 for comments regarding a transformation [25] between static and time-dependent forms).

2.2 Toolbox Components

A single scheme to handle (1) in all its generality would be impossibly complex to design and use. Instead, the Toolbox has different routines to handle different subclasses of this equation. In the rest of this section, we demonstrate the design with the example equation

$$\begin{aligned} D_t\phi(t,x) + a(t,x)\|D_x\phi(t,x)\| &= 0 && \text{for } x \in \Omega \text{ and } t \geq t_0, \\ \phi(t_0,x) &= \phi_0(x) && \text{for } x \in \Omega, \end{aligned} \tag{4}$$

which for dynamic implicit surfaces corresponds to motion in the normal direction with speed $a(t,x) : \mathbb{R} \times \Omega \to \mathbb{R}$. Ideally we would choose $\Omega = \mathbb{R}^d$ because the physical problem has no boundaries which would influence the evolution of the implicit surface.

Approximating the solution of (4) (or the more general (1) and (2)) requires the Toolbox to handle a number of features of the equations:

- Discretization of the domain Ω into a grid, including artificial boundary conditions $\phi_{\partial\Omega}$ for the necessarily finite computational domain.
- Construction of initial conditions ϕ_0.
- Approximation of spatial derivatives $D_x\phi$ (and possibly $D_x^2\phi$).
- Selection of appropriate versions of those spatial derivatives (for example, upwinding) and their combination with problem parameters in terms such as $a\|D_x\phi\|$ (or more generally G).
- Timestepping routines for $D_t\phi$.
- Visualization of the results.

To maximize flexibility, each of these tasks is a separate component of the code. Consequently, it is often possible to swap schemes for one component without having to rewrite an entire example. With reference to (4), we consider each of these features in turn.

Grid: For the rectangular Cartesian grid, the user specifies the dimension, and for each dimension the upper and lower bounds on the domain, the number of grid nodes (or equivalently the grid node spacing Δx), and the boundary conditions. The boundary conditions are handled by functions which insert appropriate ghost values into the array representing $\phi(t, x)$. The user can create their own boundary condition functions or select among functions supplied by TOOLBOXLS, including periodic, homogenous Dirichlet, homogenous Neumann, or an extrapolation method designed to maintain stability [12]. Since (4) does not have physically motivated boundary conditions, homogenous Neumann or extrapolation would normally be chosen to try to minimize the impact of the computational boundary, and the domain would be chosen large enough that the implicit surface would not venture too near those boundaries during the time interval of interest.

Grid information is stored in a structure `grid`. During initialization, TOOLBOXLS populates this structure with additional useful data, such as the `grid.xs` cell vector discussed in section 2.4. Scalar functions on this grid, such as $\phi_0(x)$, are stored in a standard d dimensional MATLAB array, where each element represents the value of ϕ_0 at the corresponding grid node.

Initial Conditions: For dynamic implicit surfaces, the Toolbox provides routines for common shapes (circles/spheres, squares/cubes, hyperplanes, cylinders) and the operations of computational solid geometry (union, intersection, complement and set difference). For general HJ PDEs, the user can often construct suitable $\phi_0(x)$ through simple array operations on the data in `grid.xs`.

Spatial Derivatives: Level set methods use upwinding when treating first order derivatives, so the routines for $D_x\phi$ all return both left and right looking approximations. In two or more dimensions, each component of the gradient is computed independently. TOOLBOXLS provides the standard first order accurate upwind approximations [29] as well as second and third order accurate

essentially non-oscillatory (ENO) [30] and fifth order accurate weighted essentially non-oscillatory (WENO) schemes [11]. The routines are interchangeable, so switching from low to high order requires changing only one function handle. ENO and WENO interpolants are computed with divided difference tables to maximize information reuse between neighboring nodes.

Motion in the Normal Direction: The vector normal to the implicit surface is given by $\hat{n}(t,x) = D_x\phi(t,x)/\|D_x\phi(t,x)\|$. For motion in this direction, the Toolbox uses a dimension by dimension Godunov upwinding scheme based on the signs of $D_x\phi(t,x)$ and $a(t,x)$ [27]. One of the upwinding spatial derivative approximation routines mentioned above is selected by the user. This routine also handles estimation of the CFL timestep restriction for explicit integrators; in this case, a function of $a(t,x)$, $\|D_x\phi(t,x)\|$ and the grid's node spacing.

Explicit Time Integration: As can be seen in the previous paragraphs, ToolboxLS adopts a method of lines approach to increase flexibility. The result of the term approximation routine (in this case, an approximation of $a\|D_x\phi\|$) is treated as the right hand side of an ODE, which is solved by an explicit strong stability preserving (SSP) Runge-Kutta (RK) integrator (formerly called total variation diminishing (TVD) Runge-Kutta). The Toolbox provides the standard first, second and third order accurate SSP RK schemes [36], which are also designed to be used interchangeably. The actual timestep size is chosen by the integrator, based on the CFL factor and the estimates of the CFL bound provided by the term approximation. The user can set parameters (such as CFL factor), choose routines to execute after each timestep (such as event detection routines to force early termination), and choose one or more of the term approximation routines described in section 2.3 (such as the motion in the normal direction term discussed above).

Visualization: One of the primary benefits of working directly in Matlab is access to all of its two and three dimensional visualization routines at all times; for example, even within the debugger. The Toolbox does not provide any new visualization features, although there are helper routines to simplify function calls and the `grid.xs` arrays often prove useful in this context.

2.3 Current Features

As mentioned in section 2.2, no single numerical scheme can achieve maximum accuracy, efficiency and ease-of-use for (1) in its full generality. Instead, the

current Toolbox implements a variety of special cases:

$$
\begin{aligned}
0 = & D_t\phi(t,x) && (5)\\
& + v(t,x)\cdot D_x\phi(t,x) && (6)\\
& + a(t,x)\|D_x\phi(t,x)\| && (7)\\
& + \mathrm{sign}(\phi(0,x))(\|D_x\phi(t,x)\|-1) && (8)\\
& + H(t,x,\phi,D_x\phi) && (9)\\
& - b(t,x)\kappa(t,x)\|D_x\phi(t,x)\| && (10)\\
& - \mathrm{trace}[\sigma(t,x)\sigma^T(t,x)D_x^2\phi(t,x)] && (11)\\
& + \lambda(t,x)\phi(t,x) && (12)\\
& + F(t,x,\phi), && (13)
\end{aligned}
$$

subject to constraints

$$
\begin{aligned}
D_t\phi(t,x) \geq 0, \qquad & D_t\phi(t,x) \leq 0, && (14)\\
\phi(t,x) \leq \psi(t,x), \qquad & \phi(t,x) \geq \psi(t,x), && (15)
\end{aligned}
$$

Note that the time derivative (5) and at least one term involving a spatial derivative (6)–(11) must appear, otherwise the equation is not a degenerate parabolic PDE.

In addition to the routines discussed in section 2.2, TOOLBOXLS also provides numerical approximations for each of the terms (5)–(15). Except where noted below, $D_x\phi(t,x)$ is approximated dimension by dimension by ENO/WENO upwind finite difference schemes with user chosen order of accuracy between one and five [11, 30], as described above.

Time derivative (5): Treated by the standard explicit SSP RK schemes with order of accuracy one to three [36], as described in section 2.2. Also, see section 3.1 for some new SSP RK schemes that are now available.

Motion by a velocity field (6) (also called advection or convection): The user provides the velocity vector field $v : \mathbb{R} \times \Omega \to \mathbb{R}^d$. Upwinding is used to choose the spatial derivative.

Motion in the normal direction (7): The user provides the speed of the interface $a : \mathbb{R} \times \Omega \to \mathbb{R}$. See section 2.2.

Reinitialization equation (8): In steady state, the solution of this equation is a signed distance function [40], a class of functions often used in dynamic implicit surfaces. For localized implementations of level set methods reinitialization is mandatory, but because the Toolbox solves the PDE(s) throughout Ω it is often possible to avoid the extra expense. However, there are some examples whose motion sufficiently distorts the initial implicit surface function so that reinitialization is necessary. While there are advantages and disadvantages to

using this equation for reinitialization, it is at present the only reinitialization procedure available in the Toolbox. This term is implemented in TOOLBOXLS with a specifically designed Godunov scheme [10], and uses the "subcell fix" from [33] to minimize movement of the zero isosurface.

General Hamilton-Jacobi term (9): The user provides the analytic Hamiltonian $H : \mathbb{R} \times \Omega \times \mathbb{R} \times \mathbb{R}^d \to \mathbb{R}$. Any dependence of H on ϕ must satisfy the monotonicity requirement (3). The average of the upwinded approximations of $D_x\phi$ is used, and Lax-Friedrichs schemes [7, 30] are available for stabilizing the approximation of $H(t, x, r, p)$ with different amounts of artificial dissipation. For scaling the dissipation, the user must provide bounds on $|\partial H/\partial p|$ given bounds on p determined by the Toolbox and the specific Lax-Friedrichs scheme. This term can be used for optimal control, differential games, and reachable set approximation.

Motion by mean curvature (10): The user provides the speed $b : \mathbb{R} \times \Omega \to \mathbb{R}^+$, while the mean curvature $\kappa(t, x)$ and gradient $D_x\phi(t, x)$ are approximated by centered second order accurate finite differences [27]. See section 3.2 for a new monotone scheme for mean curvature motion.

Potentially degenerate second order derivative term (11): The user provides the rectangular matrix $\sigma : \mathbb{R} \times \Omega \to \mathbb{R}^{d \times k}$ (where $k \leq d$) while the Hessian matrix of mixed second order spatial derivatives $D_x^2\phi(t, x)$ is approximated by centered second order accurate finite differences. The current implementation of this term suffers from the same non-monotonicity as the current mean curvature approximation. If the new mean curvature approximation described in section 3.2 proves effective, it can be extended to handle this type of term. Expectations of stochastic differential equations (whose diffusion coefficient is σ) give rise to this term in the form of Kolmogorov or Fokker-Planck equations [13, 23]. In the Toolbox documentation this term is referred to as "motion by the Trace of the Hessian," which is in retrospect a confusing and poor choice of name.

Discounting terms (12) and **forcing terms (13)**: The user provides $\lambda : \mathbb{R} \times \Omega \to \mathbb{R}^+$ or $F : \mathbb{R} \times \Omega \times \mathbb{R} \to \mathbb{R}$ respectively, and must ensure that the result satisfies the monotonicity requirement (3). Because there are no derivatives, implementation of these terms is trivial.

Constraints on the change in ϕ (14) or **on ϕ itself (15)**: For dynamic implicit surfaces, the former controls whether the surface is allowed to shrink or grow, and the latter can be used to mask a region into which the surface cannot enter [34]. The user provides $\psi : \mathbb{R} \times \Omega \to \mathbb{R}$. Although not treated in [8], viscosity solution theory has been extended to handle these constraints [22], and the Toolbox's implementation simply applies them to ϕ after each timestep.

In addition to these specific terms, the toolbox allows multiple terms to be combined, arbitrary callback functions which are executed after each timestep,

and vector level set equations where $\phi : \mathbb{R} \times \Omega \to \mathbb{R}^k$ for some constant k and each component of ϕ can be subject to a separate PDE. This collection of terms covers most of the cases arising in applications, although the Toolbox is organized so that adding more types of terms is relatively straightforward (as demonstrated in section 3).

2.4 Coding Patterns

Users of ToolboxLS—particularly those interested in adding new schemes—should be aware of three design patterns used in the code to achieve efficiency, flexibility and dimensional independence. The first is the method by which parameters of the PDE and numerical schemes are passed up and down through the different layers of routines in the Toolbox. The second is the method by which functions of $x \in \Omega$ are stored and computed. The third is the method by which we achieve dimensional independence when indexing into arrays.

Passing parameters: As can be seen in sections 2.2 and 2.3, there are many parameters related to the PDE and/or the numerical schemes which must be passed through a sequence of function calls. While object oriented programming is the typical approach adopted for such tasks, in ToolboxLS we have chosen a more light weight design. Apart from the temporal integrator, all parameters are collected into a single structure `schemeData`, whose contents are visible to (almost) all user and Toolbox routines in the call stack. Typical members of `schemeData` include the `grid` structure, function handles for the spatial derivative operator, and PDE parameters like velocity fields or speeds. By packing all of this information into a single structure, the Toolbox can maintain parameter compatibility between the various term and derivative approximation routines despite their very different internal details. The `schemeData` structure has been so successful that a similar protocol for the temporal integrator's parameters is proposed in section 3.1.

Functions of $x \in \Omega$: Remembering that $\Omega \subseteq \mathbb{R}^d$, a scalar function $\rho(x)$, $\rho : \Omega \to \mathbb{R}$ is stored as a d dimensional Matlab array `rho` of doubles. Examples include $\phi_0(x)$ or $\phi(t, x)$ for fixed t. If there are n nodes in each dimension, these arrays contain n^d elements and are typically very large. The key to the Toolbox's efficiency is to perform "vectorized" operations on these arrays. For example, when taking a time step at time t according to motion in the normal direction (4), the speed function $a(t, x)$ is collected into one array `speed` and the magnitude of the gradient $\|D_x\phi(t, x)\|$ into another `magnitude`. The normal motion term approximation $a(t, x)\|D_x\phi(t, x)\|$ is stored in a third array `delta` computed by `delta = speed .* magnitude` as a single vectorized operation (no explicit loops). This syntactic structure has the side benefit of being dimensionally independent, but its primary benefit is speed of execution. For grids with many nodes, it is memory access time that dominates total execution time. Most Matlab installations have compiled versions of elementwise

(such as ".*") and basic linear algebra operations that are optimized for memory access efficiency. Consequently, the operation above will often run faster in MATLAB than its straightfoward translation into explicit C/C++ or Fortran loop(s). For this strategy to be successful, every node by node operation in the PDE solver must be performed in this manner. In the example above, `speed` and `magnitude` must be constructed without any explicit loops by the user and the derivative approximation routine respectively. Likewise the update `delta` must be applied as a single operation by the integrator to the current solution approximation `data` (which stores $\phi(t, x)$). All of the core Toolbox is coded in this efficient manner.

It turns out that these full grid operations are so much more efficient than any type of looping that if we wish to modify only a selected set $\mathcal{N}$ containing any significant number of nodes, it is faster to modify all nodes (where that modification will be zero for nodes not in $\mathcal{N}$) than it is to loop through only the nodes in $\mathcal{N}$. As a consequence, there does not appear to be any benefit to implementing narrowbanded [4] or local level set [31] algorithms in TOOLBOXLS.

A vector function $w : \Omega \to \mathbb{R}^k$ is just a collection of k scalar functions. Each scalar function is stored in a d dimensional array as above, and these k different arrays are collected together into a MATLAB cell vector[2] of size k by 1. For example, a velocity field $v(x)$ for motion by convection (6) is stored in a d element cell vector `v`, where `v{i}` is a d dimensional array representing $v_i(x)$ (the i^{th} component of the velocity field as a function of $x \in \Omega$). While such vector functions $w(x)$ could have been stored in a single $d+1$ dimensional array, the indexing for elementwise operations becomes quite cumbersome. Matrix functions of x can similarly be stored in cell arrays of appropriate size.

One particular vector function used extensively in the Toolbox is the mapping from a node to its own location. This function is stored in the cell vector `grid.xs`, an automatically constructed part of the grid data structure `grid`. Two examples of how this vector function can be efficiently used:

- For constructing other functions of x; for example, the distance to the origin in two dimensions can be efficiently calculated for all nodes in Ω by `sqrt(grid.xs{1}.^2 + grid.xs{2}.^2)`.
- For calls to MATLAB functions; for example, `surf(grid.xs{:}, data)` generates a properly scaled surface plot of $\phi(t, x)$ for $\Omega \subset \mathbb{R}^2$. Another common command is `interpn` for interpolation in arbitrary dimension.

It should be noted that `grid.xs` is constructed through a call to `ndgrid`, which is more dimensionally consistent than the incompatible `meshgrid` but which is

[2] A cell array is a MATLAB data structure where each element can store any other MATLAB object (including other cell arrays). Element `i` of cell vector `w` is accessed by the syntax `w{i}` (the "squiggle braces" instead of the regular round braces). The syntax `w{:}` generates a comma separated list of the elements of `w`, such as might be passed as parameters to a function or used as indices into a regular array.

```
index0 = cell(grid.dim, 1);
for d = 1 : grid.dim
  index0{d} = (1 : grid.N(d)) + ghostNodes;
end

indexL = index0;
indexL{diffDim} = indexL{diffDim} - 1;
indexR = index0;
indexR{diffDim} = indexR{diffDim} + 1;

Dx2 = (data(indexR{:}) - 2 * data(index0{:}) ...
       + data(indexL{:})) / grid.dx(diffDim)^2;
```

Figure 1: Sample code for computing the standard centered difference approximation of $\partial^2\phi(t,x)/\partial x_i^2$ for all $x \in \Omega \subseteq \mathbb{R}^d$ for any d. The array `data` stores $\phi(t,x)$ and the scalar `diffDim` $= i$. It is assumed that `data` has been padded with `ghostNodes` > 1 ghost nodes in each direction in dimension i. Cell arrays are used to generate appropriately dimensioned index lists.

also consequently incompatible with `interp2`, `interp3`, and some (but not all) three dimensional MATLAB visualization routines. For additional comments on this incompatibility and some workarounds, see the Toolbox documentation.

Dimensional independence when indexing: Without any additional tricks, this method of storing functions of x allows most of the Toolbox code to be written in a dimensionally independent manner. The two exceptions which require more advanced MATLAB syntax are the finite difference stencils in the spatial derivative approximations and the boundary conditions. Figure 1 illustrates how the former code is written, and a similar indexing method is used for the latter. By creating a cell vector of dimension d with indices to every node in the grid, we can then produce appropriate offsets for every element of a finite difference stencil; for example, in figure 1 the stencil consists of nodes with offsets -1, 0 and $+1$ in dimension i. The operations in the actual finite difference computation are vectorized in the same manner as described earlier by the final line of code in figure 1.

Through these coding methods, we have removed almost all hardcoded dependence on the domain's dimension d from the Toolbox's implementation. There are a few special cases related to $d = 1$, because MATLAB arrays must have at least two dimensions for historical reasons. Visualization remains dimensionally dependent: function plots in one dimension, contours and surfaces in two, isosurfaces and slices in three. We have successfully run a few applications in four dimensions and one test case in five; however, for these high dimensional problems the grid resolution is low due to memory constraints and we have not determined any particularly good methods of visualizing the results.

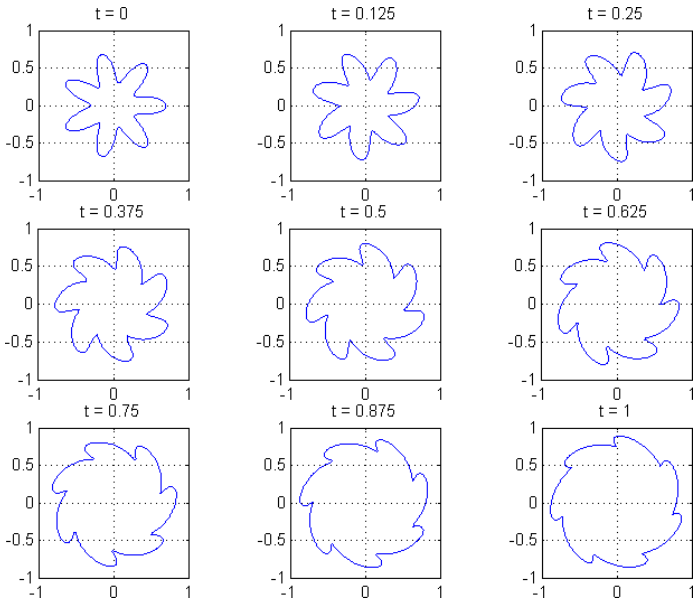


Figure 2: Dynamic implicit surface combining motion in the normal direction with a radially dependent rotational convection. Computed with second order accurate spatial and temporal approximations on a 201^2 grid.

2.5 Current Examples

Learning how to use and modify a software package written by somebody else is never a trivial task, particularly since coding conventions that seem obvious to the original programmer may be much less so to others. To help new users get started, the basic TOOLBOXLS download package includes complete code and documentation for more than twenty examples taken from the level set literature. Some additional examples are available separately from the same web site [16].

The Toolbox provides the infrastructure code for the grid, initial and boundary conditions, approximations of the spatial derivatives and terms, and time integrators. Consequently, the code for most examples is primarily concerned with initialization—in the form of data structures containing simulation parameters and selecting among the Toolbox's options—and visualizing intermediate results. In some cases, parameters are provided by short functions, such as time dependent velocity fields for (6) or general Hamiltonian terms (9). It is expected that when tackling new applications, users can cut and paste the majority of this initialization and visualization code from existing examples.

The most basic example uses motion by convection (6) and is fully annotated in the Toolbox documentation [18]. By modifying just one parameter, the user

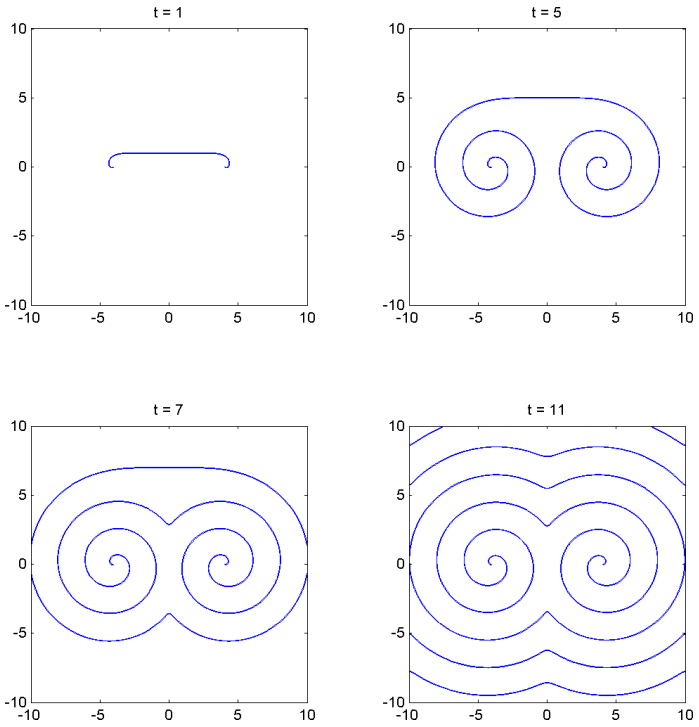
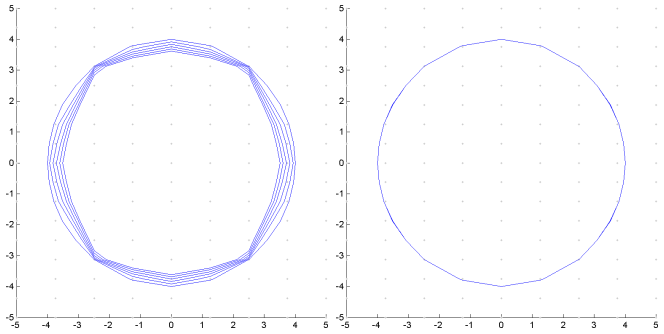


Figure 3: Counter-rotating spirals simulating a growth pattern in crystals [37]. Note that the curve is not closed, so this simulation requires the use of vector level sets.

can try a variety of schemes with different orders of accuracy. Modifying a single number allows the example to be run in dimensions one, two or three (higher dimensions will run, but there is no way to visualize the results). Time dependent motion is also supported.

Among the other examples that are included with the Toolbox download:

- Conversion of an implicit surface function for the seven pointed star (see the initial conditions in figure 2) into a signed distance function via the reinitialization equation (8).
- Motion by convection (6) subject to masking constraints (15).
- Motion by mean curvature (10) replicating [34, figures 2.6, 2.7 and 14.2] as well as replicating [27, figures 4.1 and 4.2] and extending to time-dependent motion.
- Motion in the normal direction (7) replicating [27, figure 6.1] and extending to time-dependent motion.
- The combination of convection and motion in the normal direction replicating [27, figure 6.2] with both radially independent and radially dependent velocity. The latter is shown in figure 2, animations of that motion



(a) Without subcell fix. (b) With subcell fix.

Figure 4: The initial conditions are the signed distance function for a circle, so applying the reinitialization equation should not change the function or its zero isosurface. Both figures show contours at iterations $160i$ for $i = 0, 1, \ldots, 5$. The case without the subcell fix shows that over many iterations the zero isosurface incorrectly shrinks, while the contours are indistinguishable when the subcell fix is applied. The two subplots correspond to [33, figures 7 and 8], except that the nodes of the 9×17 grid are denoted by small dots and these nodes have twice the spacing in the horizontal direction as in the vertical.

are available from [16], and the code for generating the animation is also part of the download package.

- The convex (Burgers' equation) and nonconvex general HJ PDEs from [30, figures 1(d),2(d), 3(b) and 3(d)] in both one and two dimensions.
- The continuous reach set computations for the game of two identical vehicles [19] and the acoustic capture game [3], as well as a multimode hybrid reach set for collision avoidance [17].
- Approximation of the solution of several static HJ PDEs using the transformation proposed in [25]. Time to reach the origin for both holonomic dynamics and a double integrator can be computed, although quantitative accuracy is disappointing.
- Expected values from one dimensional stochastic differential equations whose analytic solutions are known [13, 23]. Reinitialization cannot be used in these examples because the value of ϕ is meaningful throughout the domain. The approximate solution's error is much larger near the domain boundary, thus demonstrating some shortcomings of the boundary condition choices currently implemented in ToolboxLS.
- Evolution of open curves using vector level sets, including recreation of [37, figures 4 and 12]. The former is shown in figure 3. Even qualitative recreation of these results has required very high resolution simulations and regular reinitialization, so these particular examples take hours to run on a typical machine, unlike most of the others.
- Several examples from [33] demonstrating the effectiveness of the subcell

fix proposed there for maintaining the location of the interface while applying the reinitialization equation. Figure 4 shows how effectively the subcell fix maintains the location of the interface over many iterations.

Separate downloads available from [16] include code for computing the expected transmission rate and standard deviation of a stochastic hybrid system model of the Transmission Control Protocol for transmitting packets on the Internet [20] and an example of optimal control under state constraints [14].

While the examples in the Toolbox are primarily aimed at those new to level set methods, the combination of these examples and the description of internal coding patterns in section 2.4 should be sufficient for level set researchers to easily build and test new numerical schemes while leveraging the infrastructure provided by the Toolbox.

3 Adding New Features

In order to demonstrate the ease with which new schemes are added to ToolboxLS, we implement an entire class of SSP RK schemes and a monotone motion by mean curvature approximation. Not only are the implementations relatively straightforward, but once they are completed we can easily run a variety of examples to compare the new schemes against existing ones. This ability to rapidly compare against and build on existing work is one of the primary benefits that the Toolbox provides to designers of level set methods.

All of the code for the schemes and examples in this section is available as a separate download from [16]; the download also contains some additional examples which could not be included in this paper for reasons of length. Tests were performed on a 1.7 GHz Pentium M laptop (from 2004) with 1 GB memory running Windows XP version 2002 with Service Pack 2.

3.1 New Temporal Integration Schemes

The explicit SSP RK schemes in [36] were originally directed at approximating the solution of conservation laws, but have since been extensively applied to HJ PDEs by the level set community. Let p denote the order of the scheme and s the number of stages. Although schemes with up to $p = 5$ are given in [36], the restriction to $p = s$ permits only schemes with $p \leq 3$ to be implemented without providing a special backward temporal operator; furthermore, the largest CFL coefficient that maintains SSP status for the $p \leq 3$ schemes is one.

A new class of SSP schemes was proposed for conservation laws in [38] with $s > p$. Relaxing the connection between s and p permits construction of a $p = 4$

scheme without the need for the backward temporal operator, as well as more efficient $p \leq 3$ schemes: $s > p$, but a larger timestep size due to a larger CFL coefficient more than offsets the cost of the additional stages.

Following on a tradition in level set methods, we implement these schemes in TOOLBOXLS and apply them to dynamic implicit surfaces. The $s = p = 1, 2, 3$ schemes from [36] that are part of the basic Toolbox download are each implemented in separate routines, but in the interests of flexiblity we implement a major subset of the entire class of SSP schemes here. The SSP schemes are designed to solve a system of ODEs of the form

$$\frac{d}{dt}\Phi(t) = \mathcal{L}(t, \Phi(t))$$

which are generated by applying the method of lines to a time-dependent PDE. The vector $\Phi(t)$ is a discretization of $\phi(t, x)$ storing the values of $\phi(t, x)$ at the grid nodes $x \in \Omega$. Adopting the α-β parameterization from [36], but extending to time-dependent $\mathcal{L}$ requires definition of the substep sample times for $i = 0, \ldots, s$ as

$$t^{(i)} = t_n + \Delta t \sum_{k=0}^{i-1} c_{ik}, \qquad \text{where} \qquad c_{ik} = \beta_{ik} + \sum_{j=k+1}^{i-1} \alpha_{ij} c_{jk},$$

(the c_{ik} can be computed recursively [38]). Given these times $t^{(i)}$, we write the SSP schemes for the ODE as

$$\begin{aligned} \Phi^{(0)} &= \Phi(t_n), \\ \Phi^{(i)} &= \sum_{k=0}^{i-1} \left[\alpha_{ik} \Phi^{(k)} + \Delta t \beta_{ik} \mathcal{L}(t^{(k)}, \Phi^{(k)}) \right], \qquad i = 1, 2, \ldots, s, \\ \Phi(t_{n+1}) &= \Phi^{(s)}. \end{aligned} \tag{16}$$

where $\alpha_{ik} \geq 0$ and $\sum_{k=0}^{i-1} \alpha_{ik} = 1$. Because the Toolbox term approximation routines presently generate only the forward temporal operator $\mathcal{L}$, we stick to the subclass of SSP schemes where $\beta_{ik} \geq 0$. Given the relatively low priority attached to memory conservation by MATLAB and TOOLBOXLS, we have not yet worked on specifically low storage SSP schemes either.

The routine `odeCFLab` implements CFL constrained timestepping with the general scheme (16). The routine's interface is copied from the routines `odeCFLn` for `n` $= 1, 2, 3$ included with the Toolbox, and the basic integration options (such as CFL factor and callbacks) are handled in the same manner through an `options` structure controlled by `odeCFLset`. This method of controlling integrator options is designed to be similar to MATLAB's `odeset` mechanism and has been effective for parameters that are common to all integrators; however, it does not have the flexibility of the `schemeData` structure pattern adopted by TOOLBOXLS for the term approximation routines and requires modification of

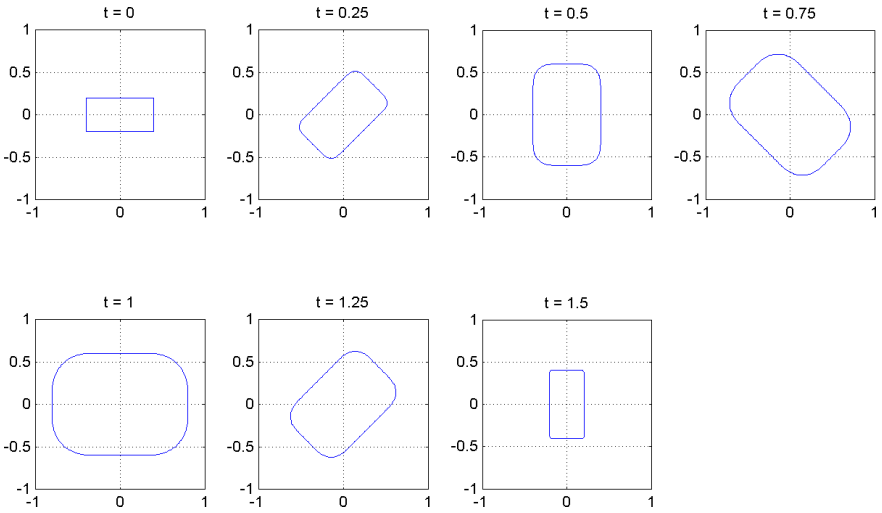


Figure 5: The rectangle spins by convection with a velocity field that is counter-clockwise rotational motion around the origin. The rectangle also moves outward in its normal direction at unit speed for $t \in [0, 1]$ and inward in its normal direction at twice that speed for $t \in [1, 1.5]$. This particular run used the (2,2) scheme from [36] on a 101^2 grid.

`odeCFLset` whenever an integrator with new parameters is introduced.[3] As an experiment, the α-β parameters for the SSP scheme are provided to `odeCFLab` through a structure `integratorData` added to the end of the parameter list.

Although `odeCFLab` can be directly called by the user to test new schemes, for the schemes in [36, 38] it is easier to call it indirectly through `odeCFLsp`, which chooses the appropriate α-β values from a collection of tables and then calls `odeCFLab` to do the work. The user must appropriately set the CFL factor to take advantage of the larger timesteps permitted by the $s > p$ schemes.

To test the accuracy and efficiency of these schemes, five simple examples were run. The first three involved a three-quarter rotation about the origin under convective flow of three different initial conditions: (1) a circle centered at the origin, (2) a rectangle centered at the origin, (3) a pair of slotted disks offset from the origin (a vertically symmetric version of Zalesak's disk [42]). The effect of this flow should be rigid body rotation of the initial conditions. The last two examples use the circle and rectangle as initial conditions, but in addition to the rotational convection the front moves outward at unit speed for the first two-thirds of the simulation, and then inward at twice that rate for the final third of the simulation. This combined motion with the rectangular initial conditions is

[3] A problem that MATLAB's `odeset` shares: consider all of the properties related to Jacobians, mass matrices and backward/numerical differentiation formulas that `odeset` contains but which are entirely ignored by the most commonly used routines `ode23` and `ode45`.

shown in figure 5. The end result of the combined motion should be the same as that produced by the purely rotational flow—a fact which would not be true for Zalesak's disk, which is why that initial condition was not used for this flow field.

The goal of these simulations was to study the effect of the order of accuracy of the time integrator on the error in the final solution. Unfortunately, error due to the spatial approximation will inevitably effect the outcome. In order to minimize its effect, we use a spatial approximation with higher order of accuracy than any of the temporal integrators that we are testing, chose very simple initial conditions, and simulate flow fields under which the spatial complexity of the front will not increase.

The spatial approximation scheme should achieve its full degree of accuracy for the circular initial conditions because there are no kinks (derivative discontinuities) in the implicit surface function near the interface; however, the lack of corners in the interface make this example too simple to be representative of typical level set problems. The interface in Zalesak's disk has both convex and concave corners, but we cannot run the second flow field for this initial condition. Consequently, we report the results for the rectangular initial conditions here.

The exact implicit surface function for the final condition is known: in every case it is just the initial implicit surface function rotated three-quarters of a turn. To test convergence and order of accuracy, the error at a node is defined as the difference between the computational solution and the analytic solution at the final time. In the case of the rectangular initial conditions, only the errors at the nodes within $\pm 1/4\ \Delta x$ of the final rectangle are included; in other words, a band of width one of nodes lying closest to the rectangle. For the other two initial conditions, only the errors at the nodes within $\pm 3/2\ \Delta x$ of the interface are included.

Code for the Toolbox's convection example was modified to construct the appropriate initial conditions and term approximations, call `odeCFLsp`, and compute the error. A wrapper routine was then written to execute each of the (s, p) integrators on a sequence of grids for each example.

The spatial derivative approximation in all cases was the fifth order accurate WENO approximation from [11]. No reinitialization was performed, and linearly extrapolated boundary conditions were used at the edge of the computational domain. All runs were performed with a CFL coefficient of 75% of the theoretical maximum for stability. Additional details can be found by examining the code in the download package. We specify the integration schemes by the pair (s, p). The schemes tested were the $(2, 2)$ and $(3, 3)$ schemes from [36] and the $(3, 2)$, $(4, 2)$, $(4, 3)$, $(5, 3)$, and $(5, 4)$ schemes from [38]. As expected, the $(1, 1)$ scheme (Forward Euler) was unstable with WENO5, so it was not tested.

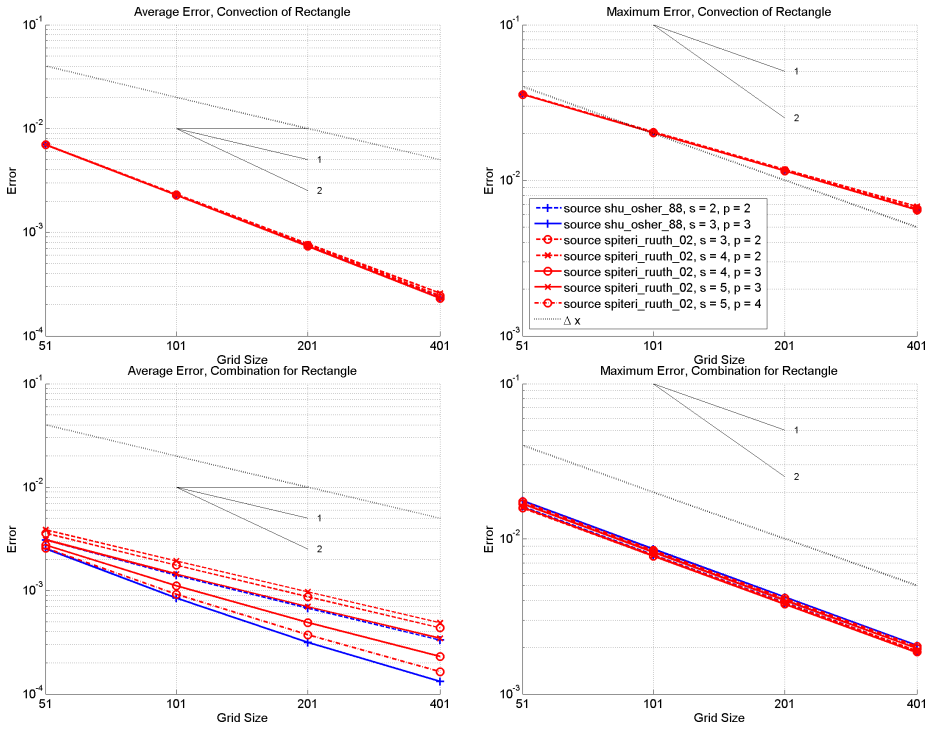


Figure 6: Convergence plots for the dynamic implicit surface representation of a rectangle integrated by a variety of explicit SSP RK schemes. A fifth order accurate WENO scheme was used in all cases for the spatial derivatives. The line style legend in the upper right plot applies to all plots. *Top row:* Purely convective motion. *Bottom row:* Convection plus motion in the normal direction ("combination") as shown in figure 5. *Left column:* Average error. *Right column:* Maximum error. Error is measured only at nodes closest to the analytic solution's interface, and the error is the magnitude of the difference between the computational and analytic solution at these nodes. For comparison purposes, the dotted line in each plot is the grid spacing Δx.

A few representative results are shown in figure 6. As can be seen in the first row, the errors for all of the schemes were essentially identical for the purely convective case and the rectangular shape. As expected, the convergence rate is far below that predicted by the formal order of accuracy for the schemes because the level set function is not sufficiently smooth. The second row is slightly more interesting, as the third and fourth order accurate schemes do show a slight advantage over the second order accurate schemes, although all of the schemes still fail to achieve their formal order of accuracy. The maximum error for this combination of motion is essentially the same for all schemes. It should be noted that this combination motion for the rectangle is actually easier than the purely

Scheme Source	Steps s	Order p	CFL Bound	Convection Time (sec)	Combination Time (sec)
Shu & Osher [36]	2	2	1.0	530	957
	3	3	1.0	853	1662
Spiteri & Ruuth [38]	3	2	2.0	390	1097
	4	2	3.0	385	840
	4	3	2.0	602	1110
	5	3	$\sim$2.65	542	1257
	5	4	$\sim$1.51	847	2221

Table 1: Execution time for the rectangular initial conditions and either pure convection or convection plus motion in the normal direction ("combination"). Timings are for a 201^2 grid run in MATLAB version 7.2 (R2006a).

rotational motion for level set methods to accurately approximate, because the shrinking phase of the combination motion causes the corners of the rectangle to sharpen, and it is in the corners that the maximum error develops.

Interested readers can generate the results for the other three examples from the code in the download package. For the circular initial conditions and purely convective motion, all schemes achieved nearly fifth order of accuracy in both average and maximum error. This super-resolution can perhaps be explained by the fact that the initial conditions are a fixed-point of the motion. For the circular initial conditions and the combination motion, all schemes were first order accurate in both average and maximum error, although there was a difference of a constant factor between the schemes—the higher order schemes had lower error, with the schemes falling in the same relative order as that seen in the lower left subplot of figure 6. For the Zalesak's disk initial conditions, the results were essentially the same as for the convective motion of the rectangle, although the maximum error was larger in magnitude and the convergence rate was noisier.

The test for the spinning rectangle was repeated with a third order accurate ENO spatial derivative approximation [30]. All errors were about 50% worse, but the relative errors among the various temporal schemes remained very similar.

While the accuracy of the new schemes from [38] was similar to or slightly worse than the standard schemes from [36], the new schemes did show improved performance. Table 1 gives some representative timings for the various schemes on the rectangular examples. Similar results were seen for the other examples.

These results seem to confirm the general wisdom that for dynamic implicit surfaces the accuracy of the spatial derivative approximation more strongly determines the overall accuracy than that of the time integrator. Furthermore, temporal schemes with orders of accuracy above two are for these examples

largely a waste of effort. The most efficient scheme was the new $(4,2)$, although the standard modified Euler $(2,2)$ scheme was nearly as effective and in some cases had slightly lower error.

The decision to break `odeCFLab`'s parameter compatibility with MATLAB's ODE integrators is an experiment, and could be reversed when this routine is brought into the base Toolbox. When the integrators were originally created, it was hoped that compatibility with MATLAB's ODE routines would make the transition to PDE solvers easier for users and would allow interoperation. Experience in the field seems to indicate instead that the near but not complete compatibility is potentially confusing, does not allow interoperation, and is definitely limiting when adding new schemes. If the experiment with the new parameter list for `odeCFLab` proves successful, the existing Toolbox routines could be modified to accept it (while still remaining backward compatible).

Creation of the routines `odeCFLab` and `odeCFLsp` required a few hours of coding, plus another few hours to create the examples and debug.

3.2 A Monotone Mean Curvature Term

While new temporal integrators are useful, the form of (1) gives much more flexibility to the spatial operator(s), and hence it is new schemes for these terms that are more often proposed. To demonstrate addition of a new spatial scheme to TOOLBOXLS, we implement the monotone scheme for motion by mean curvature proposed in [21] and further analyzed in [41].

The standard approach to the parabolic term (10) is to use the formula

$$\begin{aligned}\Delta_1\phi = \|D_x\phi\|\kappa(\phi) &= \|D_x\phi\| \operatorname{div}\left(\frac{D_x\phi}{\|D_x\phi\|}\right) \\ &= \sum_{i=1}^{d}\frac{\partial^2\phi}{\partial x_i^2} - \frac{1}{\|D_x\phi\|^2}\sum_{i,j=1}^{d}\frac{\partial^2\phi}{\partial x_i\partial x_j}\frac{\partial\phi}{\partial x_i}\frac{\partial\phi}{\partial x_j}\end{aligned} \tag{17}$$

and approximate the derivatives with centered differences [29]. This method is implemented in the base Toolbox by routines `curvatureSecond` (which computes a second order accurate centered difference approximation of the curvature and gradient) and `termCurvature` (which handles the operations in (10) and estimates the CFL timestep bound).

In [21] it is argued that this scheme is not monotone, and hence we cannot use the theory in [2] to prove convergence. The proposed alternative for approximating (17) at a node $\hat{x} \in \Omega$ begins by gathering a stencil of nodes $\mathcal{S}_{\hat{x}}$ such that each $x_k \in \mathcal{S}_{\hat{x}}$ is roughly the same distance from $\hat{x}$: $\|x_k - \hat{x}\| \approx d_x$. This stencil will be roughly circular, must be symmetric with respect to all coordinates, and

will thus have an even number of member nodes. The median value of ϕ on this stencil is then computed

$$\phi_*(\hat{x}) = \text{median}\{\phi(x_k) \mid x_k \in \mathcal{S}_{\hat{x}}\}, \tag{18}$$

and the approximation of (17) is

$$\Delta_1\phi(\hat{x}) = \frac{2(\phi_*(\hat{x}) - \phi(\hat{x}))}{d_x^2} + \mathcal{O}(d_x^2 + d_\theta), \tag{19}$$

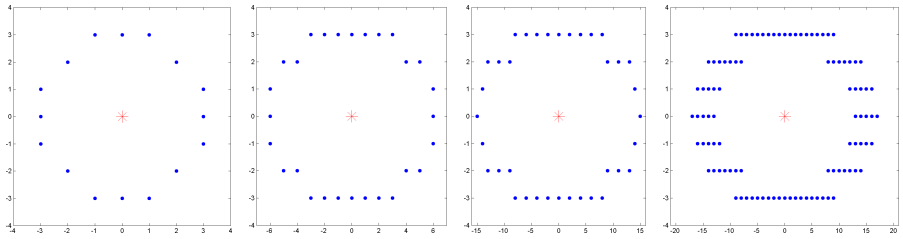
(note that (19) fixes several typos in [21, equation (6)]). The small parameter d_θ measures the angular distance between the members of $\mathcal{S}_{\hat{x}}$.

For some intuition as to why (19) approximates (17), we consider planar Ω (so $d = 2$). In this case $\Delta_1\phi$ is just a second derivative of ϕ in the direction tangent to the local isosurface of ϕ. Notice that $\phi_*(\hat{x})$ from (18) will be the average of the two middle values of $\{\phi(x_k) \mid x_k \in \mathcal{S}_{\hat{x}}\}$. Let the nodes at which these values occur be x_k' and x_k''. In regions where ϕ is sufficiently smooth, the three points x_k', $\hat{x}$ and x_k'' will lie on a line ℓ, and ℓ will be roughly tangent to the isosurface of ϕ at $\hat{x}$. Straightforward algebra shows that (19) is a standard centered difference approximation of the second derivative along ℓ. We can then interpret the two components of the error term in (19) as the error in the standard centered difference approximation of the second derivative in a given direction ($\mathcal{O}(d_x^2)$), and the error between the direction of ℓ and the true tangent direction of the isosurface ($\mathcal{O}(d_\theta)$).

This method is implemented in `termCurvatureByMedian` and `oneLaplacian`. The former has the same parameter list and is internally almost identical to `termCurvature`, since it handles various methods by which the user can supply $b(t, x)$, estimation of the CFL timestep restriction, and (if necessary) bookkeeping related to vector level sets. It calls `oneLaplacian` to compute (18) and (19). Evaluation of these two formulas is straightforward, although it can be memory intensive depending on the size of the stencil set $\mathcal{S}_{\hat{x}}$. Because the Toolbox uses a uniform mesh, the stencil set $\mathcal{S}_{\hat{x}}$ is the same for all nodes $\hat{x} \in \Omega$, so we drop the subscript from $\mathcal{S}$ in the remainder of this presentation. The vast majority of code in `oneLaplacian` is devoted to construction of $\mathcal{S}$ and its memoization (persistent storage of $\mathcal{S}$ between function calls so that it need not be reconstructed at each timestep).

The user can control the stencil pattern through a stencil width parameter w (a positive integer). We extend the algorithm in [21] to handle grids with different node spacing in each dimension. Let $\Delta x^{(i)}$ be the spacing in dimension i, and consider constructing the stencil for node $\hat{x}$; this stencil can be moved to any other node by appropriate offsets. Then we set

$$\overline{\Delta x} = \max_{1 \le i \le d} \Delta x^{(i)} \qquad \text{and} \qquad d_x = w\overline{\Delta x}$$



(a) $\mathcal{S}$ $(1:1)$ (b) $\mathcal{S}$ $(1:2)$ (c) $\mathcal{S}$ $(1:5)$ (d) $\mathcal{S}'$ $(1:5)$

Figure 7: Stencil sets $\mathcal{S}$ for stencil width $w = 3$ on grids with different node spacings in the different coordinate directions are shown in the three leftmost subplots. The stencils are built for the node $\hat{x}$, denoted by a star in the center of the stencil. The ratio below each plot is (horizontal Δx) : (vertical Δx). For comparison purposes, the intermediate stencil set $\mathcal{S}'$ is shown in the rightmost subplot for the grid ratio $1:5$. Note that many of the nodes in $\mathcal{S}'$ lie in similar or identical directions from the center node.

and create an initial stencil set

$$\mathcal{S}' = \left\{ x_k \in \Omega \mid d_x - \tfrac{1}{2}\overline{\Delta x} \leq \|x_k - \hat{x}\| < d_x + \tfrac{1}{2}\overline{\Delta x} \right\}.$$

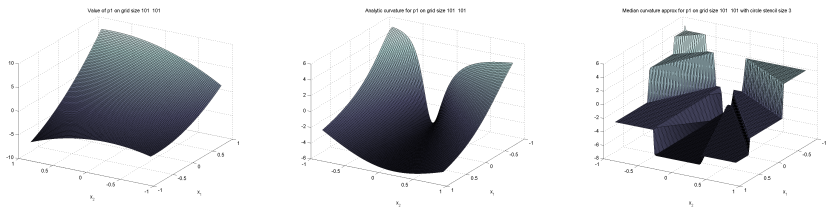
By including nodes in a band of width $\overline{\Delta x}$ around the desired stencil radius d_x, we ensure that $\mathcal{S}'$ includes at least nodes along every coordinate axis. However, when the $\Delta x^{(i)}$ are not equal for all i, the nodes in $\mathcal{S}'$ may show considerable bias toward some directions; for example, there may be multiple nodes in exactly the same direction (along the coordinate axis) if $2\Delta x^{(j)} < \overline{\Delta x}$ for some dimension j. To reduce this bias, we add node $x_l \in \mathcal{S}'$ to the final stencil set $\mathcal{S}$ only if

$$\frac{x_l - \hat{x}}{\|x_l - \hat{x}\|} \cdot \frac{x_k - \hat{x}}{\|x_k - \hat{x}\|} \geq \cos(\mu \tilde{d}_\theta)$$
$$\text{for all } \{x_k \in \mathcal{S}' \mid |\|x_k - \hat{x}\| - d_x| < |\|x_l - \hat{x}\| - d_x|\}.$$

In words, a node is added to the final stencil set $\mathcal{S}$ only if it is in a direction which makes an angle of at least $\mu\tilde{d}_\theta$ with all nodes in $\mathcal{S}'$ which are closer to the desired stencil width d_x, where $\tilde{d}_\theta$ is an estimate of the angular accuracy parameter d_θ and μ is some fractional constant (we use $\mu = 1/4$). The goal is to choose a stencil set $\mathcal{S}$ whose angular sampling in every direction is roughly constant. The resulting stencil sets for several two dimensional grids with varying ratios of $\Delta x^{(1)}$ to $\Delta x^{(2)}$ are shown in figure 7 for $w = 3$.

Any procedure for constructing $\mathcal{S}$ inevitably chooses nodes that are not precisely distance d_x from $\hat{x}$. As proposed in [41], we can adjust for this inconsistency using linear interpolation. Instead of (18), we use the definition

$$\phi_*(\hat{x}) = \text{median}\{\tilde{\phi}(x_k) \mid x_k \in \mathcal{S}_{\hat{x}}\}, \tag{20}$$



(a) $\phi_1(x)$ (b) Analytic $\Delta_1\phi_1(x)$ (c) Median-based approximation of $\Delta_1\phi_1(x)$

Figure 8: The test polynomial $\phi_1(x)$, the analytic $\Delta_1\phi_1(x)$, and the median-based approximation of $\Delta_1\phi_1(x)$ for a stencil with $w = 3$ on a 101^2 grid. Note that for better visualization the view of $\phi_1(x)$ is rotated 180° in azimuth compared to the other two plots. Because $\mathcal{S}$ only includes eight distinct directions for this w, the median-based approximation displays significant quantization.

where we interpolate a value $\tilde{\phi}(x_k)$ at a point exactly d_x along the line between $\hat{x}$ and x_k

$$\tilde{\phi}(x_k) = \phi(\hat{x}) + \frac{d_x}{\|x_k - \hat{x}\|}(\phi(x_k) - \phi(\hat{x})).$$

Such interpolation also makes it possible to use a square stencil for $\mathcal{S}$, instead of the approximately circular one discussed above. Although originally proposed in [21] without interpolation, such square stencils are only consistent approximations of Δ_1 when interpolation is used.

To study the accuracy of the median-based approximation, we use a polynomial proposed in [21]

$$\phi_1(x) = 6x_1 + \frac{5}{4}x_2 + \frac{9}{2}\frac{x_1^2}{2} + \frac{14}{5}x_1x_2 + \frac{26}{5}\frac{x_2^2}{2}, \tag{21}$$

whose analytic curvature can be derived. Results similar to those described below are also observed for the other test polynomial proposed in [21]. Representative plots of the value of $\phi_1(x)$, the analytic $\Delta_1\phi_1(x)$ and the median-based approximation for $w = 3$ are shown in figure 8. Notice the stair-step nature of the median-based approximation—because only a small number of directions is available in the stencil, there are only a small number of possible values for the approximation. We do not include a plot of the standard centered difference approximation of $\Delta_1\phi_1(x)$ because it is visually indistinguishable from the analytic solution.

While the error in the median-based scheme (19) is $\mathcal{O}(d_x^2 + d_\theta)$, for a grid with fixed connectivity we do not have direct control over d_θ in the algorithm described above because we are constrained to use neighboring nodes to construct $\phi_*(x_k)$. We decrease d_θ indirectly by increasing the stencil width w. For a grid with fixed Δx, increasing w leads to an increase in $d_x = w\Delta x$. It is not

Ratio Δx	Stencil Radius	Grid Size	Circular Stencil w/o interp Mean	Circular Stencil w/o interp Max	Circular Stencil w/ interp Mean	Circular Stencil w/ interp Max	Square Stencil w/ interp Mean	Square Stencil w/ interp Max
1 : 1	1	101 × 101	1.995	6.523	1.439	5.459	1.439	5.459
	2	284 × 284	1.425	3.704	1.191	3.554	0.896	2.960
	3	521 × 521	0.878	2.652	0.796	2.654	0.708	2.302
	4	801 × 801	0.560	1.647	0.433	1.197	0.604	2.013
	5	1119 × 1119	0.565	1.696	0.493	1.591	0.547	1.854
1 : 2	1	101 × 51	1.720	6.436	1.094	5.353	1.094	5.353
	2	284 × 142	0.929	3.669	0.670	2.580	0.737	2.782
	3	521 × 261	0.614	2.179	0.445	2.047	0.579	2.293
	4	801 × 401	0.476	1.642	0.347	1.488	0.512	2.007
	5	1119 × 560	0.401	1.216	0.295	1.162	0.477	1.849
1 : 5	1	126 × 26	1.517	6.271	0.938	5.125	0.888	5.125
	2	355 × 72	0.825	2.739	0.607	2.536	0.601	2.896
	3	651 × 131	0.593	2.137	0.429	1.830	0.503	2.271
	4	1001 × 201	0.482	1.751	0.351	1.464	0.458	1.996
	5	1399 × 281	0.378	1.173	0.271	1.040	0.437	1.843

Table 2: Error in the median approximation of $\Delta_1\phi_1(x)$ as d_θ and d_x are decreased at roughly the optimal theoretical rate. The polynomial $\phi_1(x)$ is given in (21). The error in this median-based scheme is enormous and does not consistently achieve its theoretical order of accuracy.

discussed in [21, 41] and it is not immediately obvious that we can achieve a consistent scheme $\mathcal{O}(d_x^2 + d_\theta) \to 0$ through some combination of $\Delta x \to 0$ and $w \to +\infty$. To show that consistency is possible, note that the number of nodes in the stencil $|\mathcal{S}|$ has asymptotic behaviour

$$|\mathcal{S}| = \mathcal{O}\left(\frac{\text{circumference of stencil}}{\text{distance between } x_k}\right) = \mathcal{O}\left(\frac{2\pi d_x}{\Delta x}\right) = \mathcal{O}(\Delta x^{\gamma-1}),$$

where we have used the ansatz $d_x = \Delta x^\gamma$ for some unknown constant γ. Since $d_\theta = \mathcal{O}(|\mathcal{S}|^{-1})$, we see that the error in (19) is $\mathcal{O}(\Delta x^{2\gamma} + \Delta x^{1-\gamma})$. Balancing the exponents leads to the choice $\gamma = 1/3$, asymptotic error $\mathcal{O}(\Delta x^{2/3})$ and consistency. Of course, our control over d_x is indirect through stencil width w, so to study the convergence of the scheme experimentally using the algorithm above we choose a sequence of w values and adjust Δx according to

$$w\Delta x = d_x = \Delta x^\gamma = \Delta x^{1/3} \implies \Delta x = w^{-3/2}$$

Using this relationship between w and Δx we approximate $\Delta_1\phi_1(x)$ on a variety of grids and record the error in the approximation in table 2. When compared to a typical error of $\sim 10^{-10}$ in the standard centered difference approximations of $\Delta_1\phi_1$ for these grids, the error in the median-based approach is terrible. Furthermore, although the error is decreasing as the grid is refined, the theoretical order of accuracy $(\Delta x)^{2/3}$ is not always achieved; in fact, the error for $w = 5$ on the $1:1$ grid is actually larger than for $w = 4$. It should be noted that using

interpolation (20) provides a significant benefit at all stencil sizes when compared to computing the median without interpolation (18). The square stencil is cheaper to construct (no need to search for a set of nodes approximating a circle) but is more expensive to evaluate (there are more nodes) and is generally not as accurate as the circular stencil with interpolation.

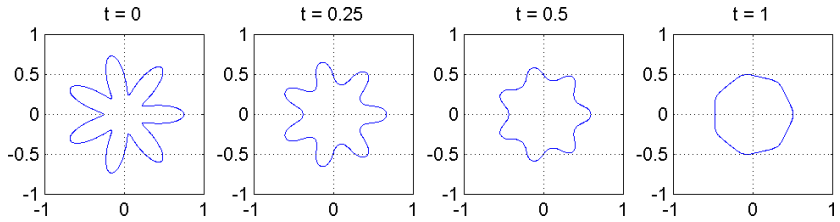
Given that the quantitative error in the median-based approximation is so large, why use it? The monotonicity of the median-based approximation is a nice feature from a theoretical perspective, but we know of no practical problems where the failure of the standard centered difference approximation to be monotone results in a significant degradation of the approximate solution when compared with the results from the median-based scheme.

Instead, the reason to use the new scheme is its speed. Motion by mean curvature (10) is often used as a regularizing term in applications like image segmentation. In these applications it is the qualitative flattening of the front generated by Δ_1 that is more important than a specific quantitative solution. To examine the qualitative effectiveness of the median-based approximation, in figure 9 we apply a variety of approximations of (10) to a star-shaped initial interface [27, figure 4.2]. Despite the poor quantitative accuracy of the median-based approximation, in motion by mean curvature the results are qualitatively very similar to the much higher accuracy standard centered difference approximation. Furthermore, the CFL timestep restriction is $\mathcal{O}(d_x^2) = \mathcal{O}(w^2 \Delta x^2)$, so for large w much longer timesteps can be taken and execution is considerably faster. In fact, the implementation in `oneLaplacian` becomes quite memory intensive for large w and hence rather slow; it is likely that a compiled implementation of the median-based scheme would have an even larger speed advantage over the standard centered difference scheme.

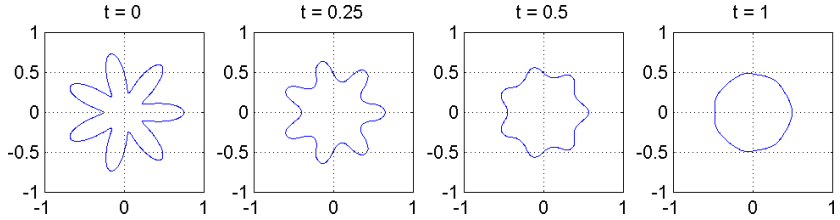
In addition to code for the tests presented here, the download package includes code recreating many of the examples from [21]. Creation of `termCurvatureByMedian` and `oneLaplacian` took several days, most of which was devoted to designing the algorithm to generate and memoize stencils on grids with variable Δx. The collection of examples and tests took about two days to fill in, using existing examples as a starting point. While `termCurvatureByMedian` and `oneLaplacian` are written to be dimensionally independent, they have not yet been tested in dimensions greater than two.

4 Conclusion and Future Work

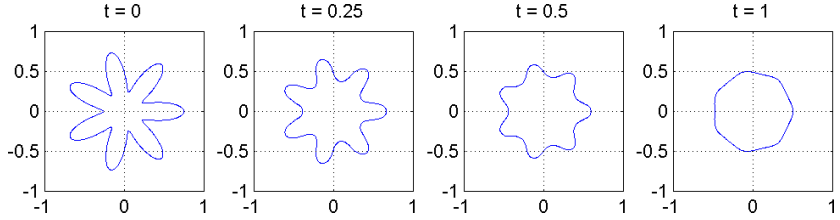
We intend for ToolboxLS to be useful to the research community in at least three ways. The first is to make easy-to-use and reasonably efficient implementations of high accuracy methods available to application scientists and engineers



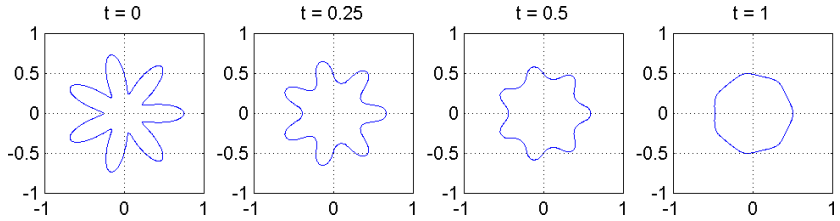
(a) Standard centered difference approximation (132 seconds)



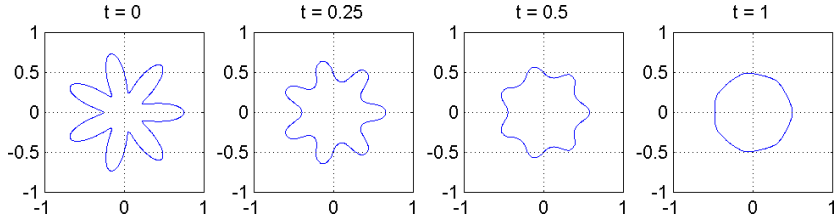
(b) Circular stencil $w = 1$ with interpolation (619 seconds)



(c) Circular stencil $w = 3$ with interpolation (78 seconds)



(d) Circular stencil $w = 5$ with interpolation (35 seconds)



(e) Square stencil $w = 3$ with interpolation (87 seconds)

Figure 9: Motion by mean curvature of a star-shaped front using various approximations of Δ_1. The grid is 201^2 nodes and the standard (2,2) time integration scheme is used. The median-based approximation may have poor quanitative accuracy, but its qualitative results are very similar to those of the standard centered finite difference, and for larger stencils it is significantly faster.

unfamiliar with the details of level set methods. The second is to furnish a flexible environment in which designers of level set methods can experiment with new schemes. The third is to provide a common software infrastructure that will aid in testing, comparison and distribution of those schemes in the spirit of reproducible research. This article has emphasized the latter two goals by presenting only a brief overview of the current Toolbox contents and devoting significant space to explaining some common internal design patterns used in the code that promote both efficiency and flexibility. The implementation of two new schemes for the Toolbox was also described, including an experimental modification of the time integrator's parameter list based on user feedback.

For the level set methods that it implements, TOOLBOXLS manages reasonable efficiency by MATLAB-style vectorization, which involves sequential access to large data arrays. Syntactic tricks involving MATLAB's cell array data type permit almost all level set operations to be written in a dimensionally independent manner. Furthermore, all of TOOLBOXLS is written in m-files, so there is no overhead to installing, executing or examining the code.

Additional level set features that we eventually hope to add to TOOLBOXLS include:

- More general and higher order accurate boundary conditions.
- Additional spatial derivative approximation schemes (such as third order accurate WENO).
- ENO/WENO function value interpolation (not just gradients) throughout Ω (not just at nodes).
- Evolution of codimension two curves in $\mathbb{R}^3$ using vector level sets.
- The "Roe with entropy fix" numerical Hamiltonian [30], which may introduce less artificial dissipation that Lax-Friedrichs.

A significant shortcoming of the present Toolbox is the lack of a Fast Marching style algorithm [34] for solving the Eikonal equation $\|D_x\phi\| = 1$ and similar static HJ PDEs. These equations play a key role in many dynamic implicit surface applications requiring reinitialization and/or velocity extension [1]. Unfortunately, these algorithms also have a random data access pattern for which MATLAB m-file implementations are extremely inefficient. It is possible to create compiled implementations that interface through MEX to MATLAB. We have performed some experiments with such implementation schemes, but their potential inclusion into TOOLBOXLS raises issues with distribution (users would have to compile), debugging (breakpoints cannot easily be set inside MEX routines), flexibility (MEX files are much more difficult to modify than m-files), and readability (users would need to know another language). We are open to comments regarding methods by which these issues might be minimized.

In fact, users of TOOLBOXLS should consider it to be a work in progress, and suggestions related to interfaces, schemes, implementations and/or applications are welcome.

Acknowledgements: A project the size of TOOLBOXLS is not constructed in isolation. The author would like to thank Ronald Fedkiw and Stanley Osher for extensive discussions about the details of numerical schemes for solving the Hamilton-Jacobi PDE. Additional discussions with David Adalsteinsson, Ken Alton, Jean-Pierre Aubin, Alexandre Bayen, Doug Enright, L. C. Evans, Chiu-Yen Kao, Alexander Kurzhanski, Doron Levy, John Lygeros, Jianliang Qian, Steven Ruuth, Patrick Saint-Pierre, James Sethian, Pravin Varaiya, Alexander Vladimirsky, Hong-Kai Zhao and many others have contributed to my understanding of the field. The support of Claire Tomlin and Shankar Sastry made possible two earlier implementations, from which the current design benefited immensely. Development of the new schemes in section 3 was motivated by discussions with Adam Oberman and Raymond Spiteri (who pointed out the instability of Forward Euler and WENO5). The author would also like to thank Adam Oberman and Ken Alton for their comments on a draft of this paper. Development and release of the code and documentation for TOOLBOXLS is supported by a grant from the National Science and Engineering Research Council of Canada.

References

[1] D. Adalsteinsson and J. A. Sethian, "The fast construction of extension velocities in level set methods," *Journal of Computational Physics*, vol. 148, pp. 2–22, 1999.

[2] G. Barles and P. E. Souganidis, "Convergence of approximation schemes for fully nonlinear second order equations," *Asymptotic Analysis*, vol. 4, no. 3, pp. 271–283, 1991.

[3] P. Cardaliaguet, M. Quincampoix, and P. Saint-Pierre, "Set-valued numerical analysis for optimal control and differential games," in *Stochastic and Differential Games: Theory and Numerical Methods*, ser. Annals of International Society of Dynamic Games, M. Bardi, T. E. S. Raghavan, and T. Parthasarathy, Eds. Birkhäuser, 1999, vol. 4, pp. 177–247.

[4] D. Chopp, "Computing minimal surfaces via level set curvature flow," *Journal of Computational Physics*, vol. 106, pp. 77–91, 1993.

[5] K. T. Chu and M. Prodanovic, "Level set method library (LSMLIB)." [Online]. Available: http://www.princeton.edu/~ktchu/software/lsmlib/

[6] M. G. Crandall and P.-L. Lions, "Viscosity solutions of Hamilton-Jacobi equations," *Transactions of the American Mathematical Society*, vol. 277, no. 1, pp. 1–42, 1983.

[7] ——, "Two approximations of solutions of Hamilton-Jacobi equations," *Mathematics of Computation*, vol. 43, no. 167, pp. 1–19, 1984.

[8] M. G. Crandall, H. Ishii, and P.-L. Lions, "User's guide to viscosity solutions of second order partial differential equations," *Bulletin of the American Mathematical Society*, vol. 27, no. 1, pp. 1–67, 1992.

[9] [Online]. Available: http://creativecommons.org

[10] R. Fedkiw, T. Aslam, B. Merriman, and S. Osher, "A non-oscillatory Eulerian approach to interfaces in multimaterial flows (the ghost fluid method)," *Journal of Computational Physics*, vol. 152, pp. 457–492, 1999.

[11] G.-S. Jiang and D. Peng, "Weighted ENO schemes for Hamilton-Jacobi equations," *SIAM J. Sci. Comput.*, vol. 21, pp. 2126–2143, 2000.

[12] C.-Y. Kao, S. Osher, and J. Qian, "Lax-Friedrichs sweeping schemes for static Hamilton-Jacobi equations," *Journal of Computational Physics*, vol. 196, pp. 367–391, 2004.

[13] P. E. Kloeden and E. Platen, *Numerical Solution of Stochastic Differential Equations*, 3rd ed., ser. Applications of Mathematics. Springer, 1999.

[14] A. B. Kurzhanski, I. M. Mitchell, and P. Varaiya, "Control synthesis for state constrained systems and obstacle problems," in *Proceedings of the IFAC Workshop on Nonlinear Control (NOLCOS)*, Vienna, Austria, 2004.

[15] V. Mallet, "Multivac C++ library." [Online]. Available: http://vivienmallet.net/fronts/index.php

[16] [Online]. Available: http://www.cs.ubc.ca/~mitchell/ToolboxLS

[17] I. Mitchell and C. Tomlin, "Level set methods for computation in hybrid systems," in *Hybrid Systems: Computation and Control*, ser. Lecture Notes in Computer Science, B. Krogh and N. Lynch, Eds. Springer Verlag, 2000, no. 1790, pp. 310–323.

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

[19] I. M. Mitchell, A. M. Bayen, and C. J. Tomlin, "A time-dependent Hamilton-Jacobi formulation of reachable sets for continuous dynamic games," *IEEE Transactions on Automatic Control*, vol. 50, no. 7, pp. 947–957, 2005.

[20] I. M. Mitchell and J. A. Templeton, "A toolbox of Hamilton-Jacobi solvers for analysis of nondeterministic continuous and hybrid systems," in *Hybrid Systems: Computation and Control*, ser. Lecture Notes in Computer Science, M. Morari and L. Thiele, Eds. Springer Verlag, 2005, no. 3414, pp. 480–494.

[21] A. Oberman, "A convergent upwind difference scheme for motion of level sets by mean curvature," *Numerische Mathematik*, vol. 99, no. 2, pp. 365–379, 2004.

[22] ——, "Convergent difference schemes for degenerate elliptic and parabolic equations: Hamilton-Jacobi equations and free boundary problems," *SIAM Journal on Numerical Analysis*, vol. 44, no. 2, pp. 879–895, 2006.

[23] B. Øksendal, *Stochastic Differential Equations: an Introduction with Applications*, 6th ed. Springer, 2003.

[24] *Journal of Scientific Computing*, vol. 19, no. 1–3, 2003.

[25] S. Osher, "A level set formulation for the solution of the Dirichlet problem for Hamilton-Jacobi equations," *SIAM Journal of Mathematical Analysis*, vol. 24, no. 5, pp. 1145–1152, 1993.

[26] S. Osher and R. Fedkiw, "Level set methods: An overview and some recent results," *Journal of Computational Physics*, vol. 169, pp. 463–502, 2001.

[27] ——, *Level Set Methods and Dynamic Implicit Surfaces.* Springer, 2002.

[28] S. Osher and N. Paragios, Eds., *Geometric Level Set Methods in Imaging, Vision and Graphics.* Springer, 2003.

[29] S. Osher and J. A. Sethian, "Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations," *Journal of Computational Physics*, vol. 79, no. 1, pp. 12–49, 1988.

[30] S. Osher and C.-W. Shu, "High-order essentially nonoscillatory schemes for Hamilton-Jacobi equations," *SIAM Journal on Numerical Analysis*, vol. 28, no. 4, pp. 907–922, 1991.

[31] D. Peng, B. Merriman, S. Osher, H. Zhao, and M. Kang, "A PDE based fast local level set method," *Journal of Computational Physics*, vol. 165, pp. 410–438, 1999.

[32] G. Peyré, "Toolbox fast marching." [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=6110

[33] G. Russo and P. Smereka, "A remark on computing distance functions," *Journal of Computational Physics*, vol. 163, pp. 51–67, 2000.

[34] J. A. Sethian, *Level Set Methods and Fast Marching Methods.* New York: Cambridge University Press, 1999.

[35] ——, "Evolution, implementation, and application of level set and fast marching methods for advancing fronts," *Journal of Computational Physics*, vol. 169, pp. 503–555, 2001.

[36] C.-W. Shu and S. Osher, "Efficient implementation of essentially non-oscillatory shock-capturing schemes," *Journal of Computational Physics*, vol. 77, pp. 439–471, 1988.

[37] P. Smereka, "Spiral crystal growth," *Physica D*, vol. 138, pp. 282–301, 2000.

[38] R. J. Spiteri and S. J. Ruuth, "A new class of optimal high-order strong-stability-preserving time discretization methods," *SIAM Journal on Numerical Analysis*, vol. 40, no. 2, pp. 469–491, 2002.

[39] B. Sumengen, "A Matlab toolbox implementing level set methods." [Online]. Available: http://barissumengen.com/level_set_methods/index.html

[40] M. Sussman, P. Smereka, and S. Osher, "An improved level set method for incompressible two-phase flow," *Journal of Computational Physics*, vol. 114, pp. 145–159, 1994.

[41] R. Takei, "Modern theory of numerical methods for motion by mean curvature," Master's thesis, Department of Mathematics, Simon Fraser University, August 2007.

[42] S. T. Zalesak, "Fully multidimensional flux-corrected transport algorithms for fluids," *Journal of Computational Physics*, vol. 31, no. 3, pp. 335–362, June 1979.