A second-order fast marching eikonal solver[a]

[a]Published in SEP Report, 100, 287-292 (1999)

James Rickett and Sergey Fomel[1]

INTRODUCTION

The fast marching method (Sethian, 1996) is widely used for solving the eikonal equation in Cartesian coordinates. The method's principal advantages are: stability, computational efficiency, and algorithmic simplicity. Within geophysics, fast marching traveltime calculations (Popovici and Sethian, 1997) may be used for 3-D depth migration or velocity analysis.

Unfortunately, first-order implementations lead to inaccuracies in computed traveltimes, which may lead to poor image focusing for migration applications. In addition, first-order traveltimes are not accurate enough for reliable amplitude calculations. This has lead to the development of the fast marching method on non-Cartesian (Alkhalifah and Fomel, 1997; Sun and Fomel, 1998), and even unstructured (Fomel, 1997) grids. These non-Cartesian formulations reduce inaccuracies, while retaining the fast marching method's characteristic stability and efficiency. Unfortunately, the cost is the loss of algorithmic simplicity.

We implement a second-order fast marching eikonal solver, which reduces inaccuracies while retaining stability, efficiency *and* simplicity.

FAST MARCHING AND THE EIKONAL EQUATION

Under a high frequency approximation, propagating wavefronts may be described by the eikonal equation,

$$\left(\frac{\partial t}{\partial x}\right)^2 + \left(\frac{\partial t}{\partial y}\right)^2 + \left(\frac{\partial t}{\partial z}\right)^2 = s^2(x,y,z), \tag{1}$$

where t is the traveltime, s is the slowness, and x, y and z represent the spatial Cartesian coordinates.

The fast marching method solves equation (1) by directly mimicking the advancing wavefront. Every point on the computational grid is classified into three groups: points behind the wavefront, whose traveltimes are known and fixed; points on the wavefront, whose traveltimes have been calculated, but are not yet fixed; and points ahead of the wavefront. The algorithm then proceeds as follows:

1. Choose the point on the wavefront with the smallest traveltime.
2. Fix this traveltime.
3. Advance the wavefront, so that this point is behind it, and adjacent points are either on the wavefront or behind it.

[1]**e-mail:** james@sep.stanford.edu, sergey@sep.stanford.edu

4. Update traveltimes for adjacent points on the wavefront by solving equation (1) numerically.

5. Repeat until every point is behind the wavefront.

The update procedure (step 4.) requires the solution of the following quadratic equation for t,

$$\begin{aligned} \max(D^{-x}_{ijk}t,0)^2 + \min(D^{+x}_{ijk}t,0)^2 &\quad + \\ \max(D^{-y}_{ijk}t,0)^2 + \min(D^{+y}_{ijk}t,0)^2 &\quad + \\ \max(D^{-z}_{ijk}t,0)^2 + \min(D^{+z}_{ijk}t,0)^2 &= s_{ijk} \end{aligned} \tag{2}$$

where D^{-x}_{ijk} is a backward x difference operator at grid point, ijk, D^{+x}_{ijk} is a forward x operator, and finite-difference operators in y and z are defined similarly. The roots of the quadratic equation, $at^2 + bt + c = 0$, can be calculated explicitly as

$$t = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}. \tag{3}$$

Solving equation (2) amounts to accumulating coefficients a, b and c from its non-zero terms, and evaluating t with equation (3).

If we choose a two-point finite-difference operator, such as

$$D^{-x}_{ijk}t = \frac{t_{ijk} - t_{(i-1)jk}}{\Delta x} \tag{4}$$

$$\text{then} \quad (D^{-x}_{ijk}t)^2 = \alpha t^2_{ijk} + \beta t_{ijk} + \gamma \tag{5}$$

where $\alpha = \frac{1}{\Delta x^2}$, $\beta = -2t_{(i-1)jk}\alpha$ and $\gamma = t^2_{(i-1)jk}\alpha$. Coefficients a, b and c can now be calculated from $a = \Sigma_l \alpha_l$, $b = \Sigma_l \beta_l$, and $c = \Sigma_l \gamma_l - s^2$, where the summation index, l, refers to the six terms in equation (2) subject to the various min/max conditions.

This two-point stencil, however, is only accurate to first-order. If instead we choose a suitable three-point finite-difference stencil, we may expect the method to have second-order accuracy. For example, the second-order upwind stencil,

$$D^{-x}_{ijk}t = \frac{3t_{ijk} - 4t_{(i-1)jk} + t_{(i-2)jk}}{2\Delta x} \tag{6}$$

$$\text{gives} \quad (D^{-x}_{ijk}t)^2 = \alpha' t^2_{ijk} + \beta' t_{ijk} + \gamma' \tag{7}$$

$$\begin{aligned} \text{where this time} \quad \alpha' &= \frac{9}{4\Delta x^2}, \\ \beta' &= \frac{-3(4t_{(i-1)jk} - t_{(i-2)jk})}{2\Delta x^2} = -2\alpha' t'_{(i-1)jk}, \\ \gamma' &= \frac{(4t_{(i-1)jk} - t_{(i-2)jk})^2}{4\Delta x^2} = \alpha' {t'}^2_{(i-1)jk}, \\ \text{and} \quad t'_{(i-1)jk} &= \frac{1}{3}(4t_{(i-1)jk} - t_{(i-2)jk}). \end{aligned}$$

Coefficients, a, b and c can be accumulated from α', β' and γ' as before, and if the traveltime, $t_{(i-2)jk}$ is not available, first-order values may be substituted.

ACCURACY

Figure 1 shows traveltime contour maps computed with the first and second-order fast marching methods on a sparse (20×20) grid. The large errors for waves propagating at 45° to the grid are visibly reduced by the second-order formulation.

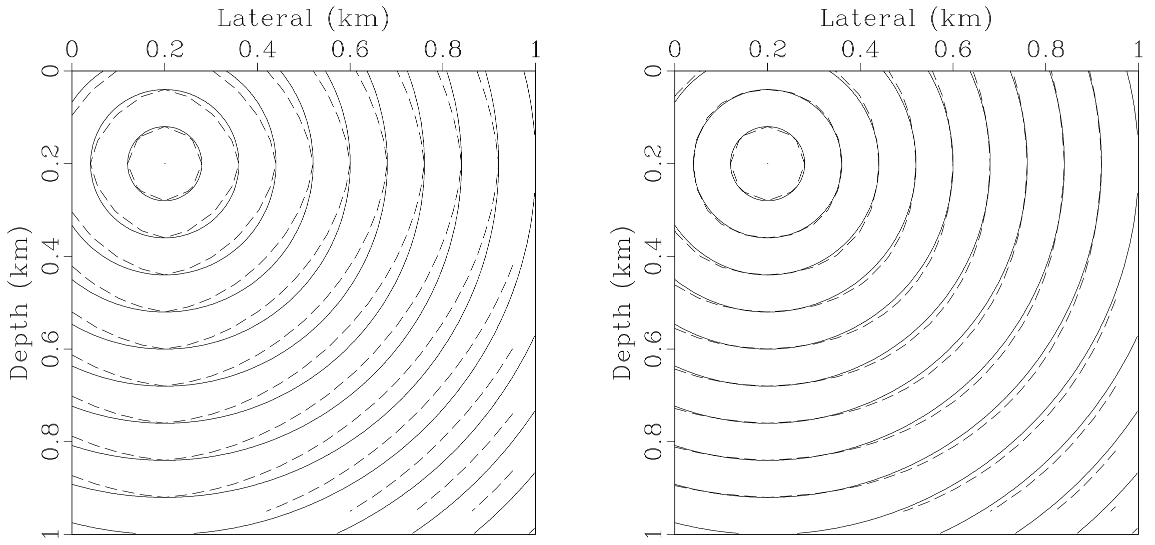

Figure 1: Traveltime contours in a constant velocity medium. The solid line shows the exact result. The dashed line shows the first-order (left panel) and second-order (right panel) fast marching results, calculated on a 20×20 grid.

Figure 2 shows the average error as a function of grid spacing for the first and second-order solvers. Not only is the second-order formulation more accurate at large grid spacing, but its accuracy increases more rapidly as grid spacing decreases. Theory predicts the $\log - \log$ plots of average error against grid spacing to be a linear function with gradient of one for first-order methods, and two for second order methods. In practice, the fast marching results come very close to these criteria up to the limits of machine precision. Figure 2 demonstrates the superiority of the second-order fast marching formulation.

It is worth noting, at this point, that special treatment is required at the source location, since the singularity in wavefront curvature will cause numerical errors to propagate into the traveltime solution. We surround the source with a constant velocity box, within which we calculate traveltimes by ray-tracing. Errors are inversely proportional to the radius of this box. Therefore, if the radius of the box decrease with grid spacing, errors will increase linearly, reducing the accuracy of the method to first-order. For full second-order accuracy, the box size should be independent of grid spacing.

COMPUTATIONAL COST

The leading term in the computational cost of the fast marching algorithm comes from the first step: choosing the point on the wavefront with the smallest traveltime. Consequently, the cost should not depend strongly on the order of the finite-difference stencil, but rather

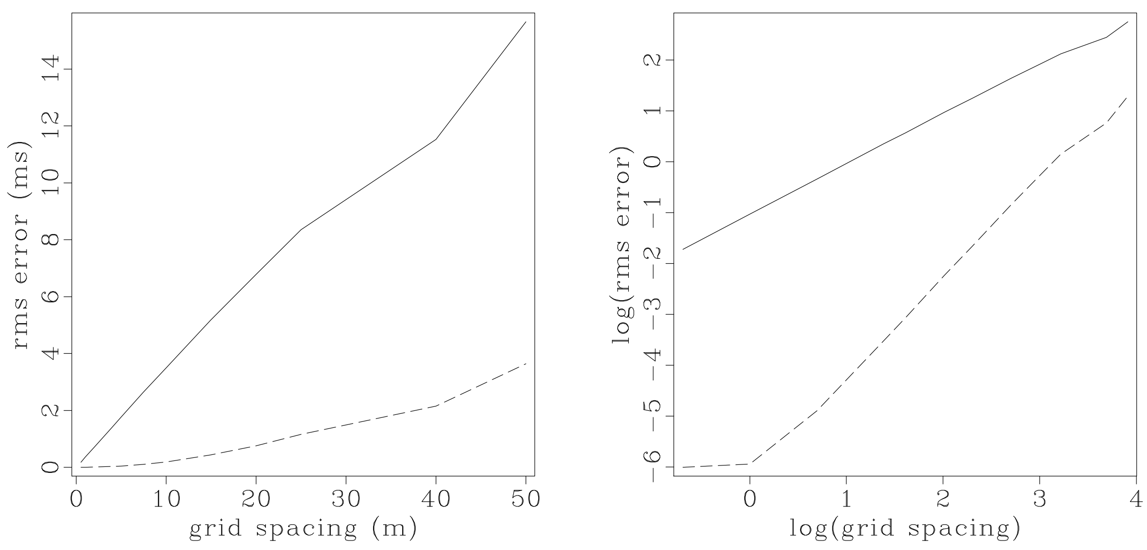

Figure 2: Average error against grid spacing for a constant velocity model. The solid line corresponds to the first-order eikonal solver, and the dashed line corresponds to the second-order solver. The left panel has linear axes, whereas the right panel is a log − log plot.

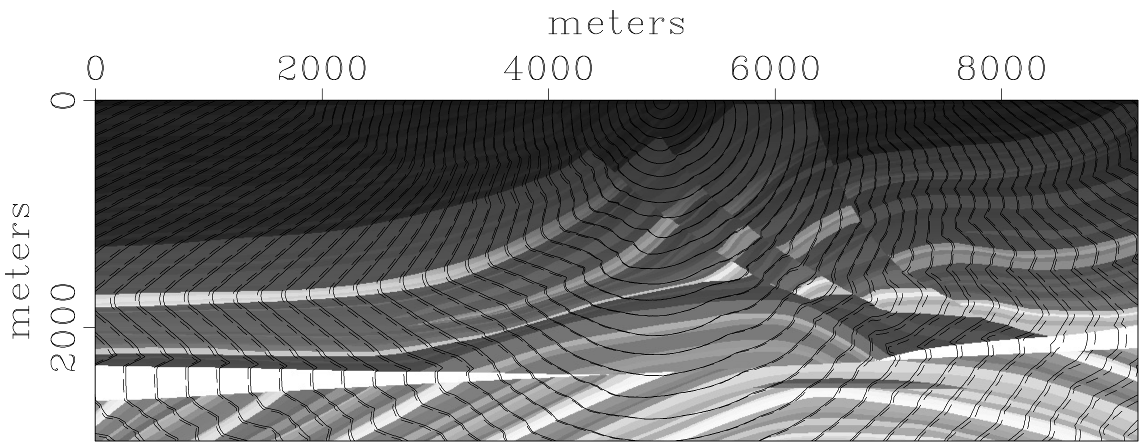

Figure 3: Traveltime contours calculated through the Marmousi velocity model sampled at 4 m. Solid line shows first-order results, and dashed line shows second-order results.

the sort algorithm used. Heap sorting has a cost of $O(\log N)$, and so in principle, with this algorithm, the fast marching method has a cost of $O(N \log N)$.

The left panel of Figure 4 shows a plot of CPU time against N for the same models as Figure 2. The time shown is elapsed (wall clock) time on a 300 MHz Pentium II. For the largest model computed here, the second-order code takes 11% longer to run than the first-order code, and this percentage decreases as N increases.

Because $\log N$ grows slowly compared to N, the plot of CPU time against N is dominated by the linear term. The right panel in Figure 4 addresses this issue by showing CPU time divided by N versus N. On this graph, the $\log N$ behaviour is clearly visible.

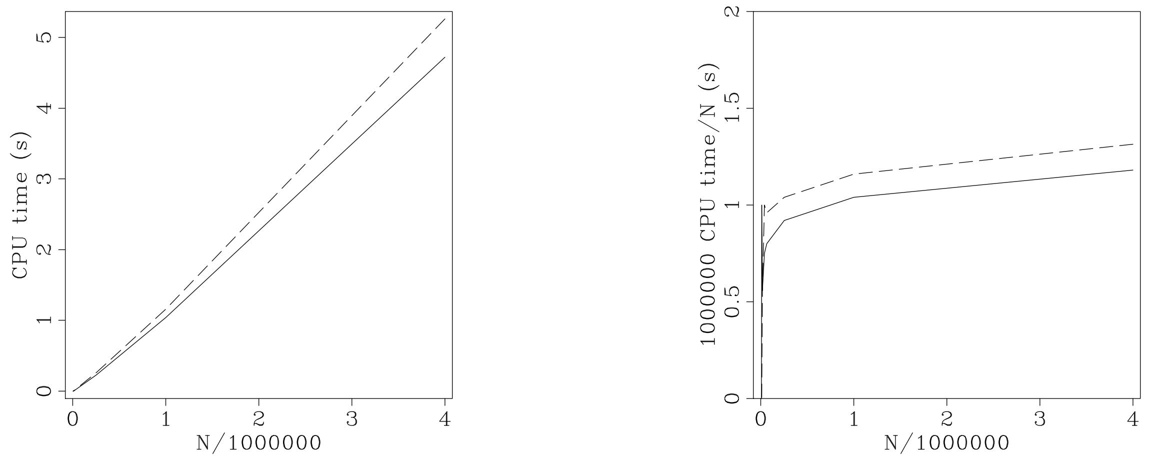

Figure 4: Elapsed CPU time vs. the number of grid points, N, for first-order (solid line) and second order (dashed line) eikonal solvers. Left panel shows CPU time vs N. Right panel shows CPU time/N vs N.

CONCLUSIONS

We have shown that a second-order implementation of the fast marching eikonal solver produces traveltimes with a much higher accuracy than the first-order implementation. What is more, the additional accuracy is acheived at only a marginal increase in cost.

This second-order implementation should become the standard method for computing first-arrival traveltimes within SEP.

REFERENCES

Alkhalifah, T., and S. Fomel, 1997, Implementing the fast marching eikonal solver: Spherical versus cartesian coordinates, *in* SEP-95: Stanford Exploration Project, 149–171.

Fomel, S., 1997, A variational formulation of the fast marching eikonal solver, *in* SEP-95: Stanford Exploration Project, 127–147.

Popovici, A. M., and J. Sethian, 1997, Three-dimensional traveltime computation using the fast marching method: 67th Ann. Internat. Mtg, Soc. of Expl. Geophys., 1778–1781.

Sethian, J. A., 1996, Level set methods: Evolving interfaces in geometry, fluid mechanics, computer vision, and materials science: Cambridge University Press.

Sun, Y., and S. Fomel, 1998, Fast-marching eikonal solver in the tetragonal coordinates, *in* SEP-97: Stanford Exploration Project, 241–250.