Location and mapping of hydrofractures from arrival times at wells

Victor Pereyra and Mihai Popovici
Weidlinger Assoc. Los Altos, CA and 3D-Geo, M. View, CA

SUMMARY

We combine a fast eikonal solver with an optimization algorithm to locate and map fractures caused by injection in the secondary recovery of hydrocarbons. The data used is arrival times on wells, although the method is applicable to other problems and acquisition geometries. We describe the problem and the various processes involved and illustrate the numerical behavior with a synthetic data example.

INTRODUCTION

We consider the problem of locating and mapping fractures produced by forced injection in a reservoir. The initial assumption is that the velocity of propagation of elastic waves is known on a mesh covering the area under investigation and that geophones are placed on wells to passively listen to hydrocracking. The medium will be assumed to be isotropic.

The measured quantities are times of arrival of signals produced by the fractures. Neither the locations nor the origin times of the signals are known.

We propose to use differential times in order to eliminate the origination time from the problem. Then we will calculate travel time tables from the receivers to every point on a three dimensional mesh using a fast eikonal solver. We will produce also calculated differential times by substracting each one of these tables from a fixed reference one.

The algorithm will then consist of finding the best matching set of differential times in the resulting calculated tables. Although we could just do a brute force inspection of the whole mesh, which in 3D could have more than 1,000,000 nodes, we prefer to use an optimization technique which will provide the answer in just a few hundred nodal evaluations.

THE PROBLEM AND ITS SOLUTION

Let $P_{ijk} = [x_i, y_j, z_k], i = 1, ..., I; j = 1, ..., J; k = 1, ..., K$, be an uniform mesh in three dimensions, with mesh spacings: $\delta x, \delta y, \delta z$. Let v_{ijk} represent the velocity of propagation of pressure waves at the point P_{ijk}. Let $G_l = [g_{xl}, g_{yl}, g_{zl}], l = 0, ..., L$ be a set of geophone positions. Finally, let T_l^o be a set of arrival times recorded at the geophone positions and corresponding to a signal produced by a crack in the vicinity of the geophones.

We first create the differential times:

$$DT_l^o = T_l^o - T_0^o, \ l = 1, ..., L.$$

The next step requires calculating travel times from the geophones to each point in the mesh. This generates $L+1$ travel time tables: T_l^c, and by taking the differences with T_0^c we similarly create the calculated difference times: $DT_l^c, \ l = 1, ..., L$.

Once these tables are created, the problem is reduced to a minimization one, namely:

$$min_{x,y,z} \parallel DT^c(x, y, z) - DT^o \parallel_2 .$$

In order to solve this problem we need to extend our mesh function DT^c to continuous values by interpolation, and then we can call upon an appropriate derivative free minimization procedure. We have tested an intelligent search algorithm due to Nelder and Mead [5] as implemented by Hill [4], and a derivative-free scheme called PRAXIS, due to R. Brent [1].

EIKONAL SOLVER

We discuss now a fast algorithm for solving the eikonal equation in three dimensions, based on the Fast Marching Method (FMM). The algorithm is of order $O(N \log N)$, where N is the total number of grid points in the computational domain. it can be used in any orthogonal coordinate system, and globally constructs the solution to the eikonal equation for each point in the coordinate domain. The method is unconditionally stable, and constructs solutions consistent with the exact solution for arbitrarily large gradient jumps in velocity. In addition, the method resolves any overturning propagation wavefronts. The algorithm presented here is designed for computing first arrival traveltimes. Nonetheless, since it exploits the Fast Marching Method for solving the eikonal equation, we believe that it is the fastest of all possible consistent schemes to compute first arrivals.

ENTROPY-SATISFYING APPROXIMATIONS TO THE EIKONAL EQUATION

A propagating interface can develop corners and discontinuities as it evolves. This requires the introduction of a weak solution in order to proceed. The correct weak solution comes from enforcing an entropy condition posed by Sethian (1985) for the propagating interface, similar to one used in gas dynamics. To solve the problem numerically we consider the ideas of upwind finite difference approximations introduced by Osher and Sethian (1988) which are appropriate for the eikonal equation:

$$|\nabla u(x,y,z)| = s(x,y,z), \tag{1}$$

where $u(x,y,z)$ is the traveltime field and $s(x,y,z)$ is the slowness function in the three dimensional model.

An upwind scheme which is convenient for our implementation of the fast marching method [9], is given by

$$\begin{bmatrix} \max(D^{-x}_{ijk}u, -D^{+x}_{ijk}u, 0)^2 + \\ \max(D^{-y}_{ijk}u, -D^{+y}_{ijk}u, 0)^2 + \\ \max(D^{-z}_{ijk}u, -D^{+z}_{ijk}u, 0)^2 \end{bmatrix}^{1/2} = F_{ijk}, \tag{2}$$

where we use the same forward and backward operators D^- and D^+ and F_{ijk} is the slowness at the gridpoint ijk.

The central idea behind the fast marching method is to solve the eikonal equation by systematically constructing the travel times $u(x,y,z)$ in an upwind fashion. Essential to the method is the observation that the upwind difference structure of equation (2) means that information propagates "one way", that is, from smaller values of $u(x,y,z)$ to larger values. Hence, the fast marching algorithm rests on solving equation (2) by building the solution outwards from the smallest $u(x,y,z)$ value.

The algorithm is made fast by confining the "building zone" to a narrow band around the front.

OPTIMIZATION: THE NELDER-MEAD ALGORITHM

The Nelder-Mead algorithm is an implementation of a so-called polytope or simplex method. We follow the discussion in [3] to give an introduction to it.

At each stage of the algorithm, $n+1$ points $\mathbf{x_1}, ..., \mathbf{x_{n+1}}$, and their corresponding function values are retained. It is assumed that the function values are in ascending order:

$$f_{n+1} \geq f_n \geq ... \geq f_1.$$

These points can be considered the vertices of a polytope in n dimensions. At each iteration, a new polytope is generated by producing a new point that replaces the "worst" point $\mathbf{x_{n+1}}$

Let $\mathbf{c}$ denote the centroid of the first n points:

$$\mathbf{c} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x_i}.$$

At the begining of each iteration, a trial point is generated by a single reflection step:

$$\mathbf{x_r} = \mathbf{c} + \alpha(\mathbf{c} - \mathbf{x_{n+1}}), \quad with\ \alpha > 0.$$

There are three possible cases to consider:

- If $f_1 \leq f_r \leq f_n$, then $\mathbf{x_r}$ replaces $\mathbf{x_{n+1}}$.
- If $f_r < f_1$ we assume that the reflection direction is "good" and try to expand the polytope in this direction by defining:

$$\mathbf{x}_e = \mathbf{c} + \beta(\mathbf{x_r} - \mathbf{c}),$$

with $\beta > 1$. If $f_e < f_r$, then $\mathbf{x_e}$ replaces $\mathbf{x_{n+1}}$. Otherwise, $\mathbf{x_r}$ replaces $\mathbf{x_{n+1}}$.

- If $f_r > f_n$ then the polytope is assumed to be too large and a contraction step is carried out:

$$\mathbf{x_c} = \begin{cases} \mathbf{c} + \gamma(\mathbf{x_{n+1}} - \mathbf{c}), & if\ f_r \geq f_{n+1}, \\ \mathbf{c} + \gamma(\mathbf{x_r} - \mathbf{c}) & if\ f_r < f_{n+1}, \end{cases} \tag{3}$$

with $0 < \gamma < 1$. If $f_c < min(f_r, f_{n+1})$ then $\mathbf{x_c}$ replaces $\mathbf{x_{n+1}}$, otherwise a further contraction is carried out.

A number of modifications can be made to this basic procedure in order to improve its performance and increase its robustness.

PRAXIS

PRAXIS [1] is an implementation of a modified version of Powell's [8] method for minimization of $f(\mathbf{x})$ without using derivatives. The basic idea of Powell's method is as follows.

Let $\mathbf{x_0}$ be an initial approximation to the minimum, and let $\{\mathbf{u_i}\}_{i=1,...,n}$ be the columns of the identity matrix. One iteration of the basic procedure consists of the following steps:

- For $i = 1, ..., n$, calculate β_i that minimizes $f(\mathbf{x_i} + \beta_i \mathbf{u_i})$, and define $\mathbf{x_i} = \mathbf{x_{i-1}} + \beta_i \mathbf{u_i}$.
- For $i = 1, ..., n$, replace $\mathbf{u_i}$ by $\mathbf{u_{i+1}}$.
- Replace $\mathbf{u_n}$ by $\mathbf{x_n} - \mathbf{x_0}$.
- Compute β that minimizes $f(\mathbf{x_0} + \beta \mathbf{u_n})$ and replace $\mathbf{x_0}$ by $\mathbf{x_0} + \beta \mathbf{u_n}$.

These steps are repeated until a stopping criterium is satisfied.

If f is quadratic, then it can be shown by induction that the vectors $\mathbf{u_{n-k+1}}, ..., \mathbf{u_n}$ are conjugate, and after n steps we would reach the minimum, provided that no $\mathbf{u_i}$ vanishes. This will be true if at each iteration $\beta_1 \neq 0$.

A number of modifications and safeguards have been introduced by Brent into this basic procedure as explained in the reference mentioned at the begining.

TESTING

In order to test the algorithms we generate an INTEGRA model [7] by the name of *cracks*. It consists of a layer over a half space with a velocity given by a gradient in z plus a 2D lateral correction in the form of a 31×31 tensor product B-spline. Four vertical wells are located at $[4, 5], [5, 4], [6, 5]$, and $[5, 6]$, surrounding a crack located at $x = 5, y = 5, z = 5$. Eleven receivers are placed in each well at 1.0 intervals, starting at depth 0.5.

Rays are traced from the crack to each receiver and the travel and differential times with respect to the first receiver are calculated. This is the synthetic data. For this problem, this step (which will not be needed for real data) took only 27" on a SUN 10 workstation.

We also run the eikonal solver to generate travel time tables for shots placed at the receivers (reciprocity principle) to a box containing the crack; from them we generate the corresponding arrival time differences. The mesh has origin at $[3.5, 3.5, 0.0]$, and $[61, 51, 133]$ mesh points in the $[x, y, z]$ directions, covering the box $[3.5, 6.5] \times [3.5, 6.0] \times [0.0, 11]$ with grid spacings $0.05, 0.05, 0.0833333$.

This is the most time consuming step, taking 72' on a SUN 10 workstation. Of course, we have to put the task in perspective. We have generated 44 travel time tables on a mesh with 413,763 points, so the calculation has proceeded at a rate of 4,214 travel times per second. Also this step is easily parallelizable on a distributed network of computers.

In order to avoid flat spots in the goal functional we use linear interpolation between the eight mesh points nearer to a desired target point (x, y, z). This is the function provided to the Nelder-Mead and PRAXIS algorithms for minimization. Below we give the results obtained when running the two algorithms with data from one to four wells,

in order to get a feel for the coverage necessary for good accuracy. Surprisingly, the most accurate results are obtained with just one well for the PRAXIS algorithm, although all the cases give locations with errors below 150 ft.

#Wells	#iter N-M	#iter Prax.	x	y	z	error
1	172*	102	4.924	5.000	5.027	0.069
2	254	59	4.912	4.964	5.022	0.097
3	435	78	4.865	4.939	5.005	0.148
4	232	70	4.895	5.046	5.009	0.115

Despite the disparity in the number of iterations, both algorithms take about the same time, and come up with the same solution (most of the time), so it is hard to choose among them. Nelder-Mead is a bit slower in some of the cases. It is also not clear that increasing the number of sensors buy us much in terms of accuracy. Of course, this is a test with essentially zero noise. For real data redundancy may be important.

Fortunately, for one well, Nelder-Mead gave a completely different result: [4.650, 5.600, 5.029], which makes it unreliable. Still, it can be used (most of the time) to check the results of PRAXIS by doubling the cost of the calculation. By the way, the longest computing time for the minimization was for four wells, and it amounted to five minutes on a slow SUN 10 workstation. For one well, it takes only one minute.

In the case that we record during a time period where the crack is breaking, it would be possible to apply this algorithm with continuation. That is, once we locate the first signal in space, we can then use that value to start the next calculation. That will reduce the computing time radically and it will provide a mechanism to map the complete crack event, including the length and orientation of the crack.

References

[1] Brent, R.P., Algorithms for finding zeros and extrema of functions without calculating derivatives, *Dissertation,* Stanford University, 1971.

[2] Engquist, B., and Osher, S., Stable and entropy satisfying approximations for transonic flow calculations, *Math. Comp.*, 1980, 34:45-75.

[3] Gill, P.E., Murray, W. and Wright, M.H., *"Practical Optimization".* Academic Press, San Francisco, 1981.

[4] Hill, *Appl. Statis.*, 1978, 27, 380-382.

[5] Nelder, J.A., and Mead, R., A simplex method for function minimization, *Computer Journal,* 1965, 7, 308-313.

[6] Osher, S., and Sethian, J.A., 1988, Fronts propagating with curvature dependent speed: algorithms based on Hamilton-Jacobi formulation: Journal of Computational Physics, 79: 12–49.

[7] Pereyra, V., Two point ray tracing in general 3D media, *Geophysical Prospecting,* 1992, 40, 267-287.

[8] Powell, M.J.D., An efficient method for finding the minimum of a function of several variables without calculating derivatives, *Comp. J.*, 1964, 7:155-162.

[9] Rouy, E. and Tourin, A., 1992, A Viscosity Solutions Approach to Shape-From-Shading: SIAM J. Num. Anal, **29**, 867–884.

[10] Sethian, J.A., 1985, Curvature and the evolution of fronts: Commun. in Math. Physics, 101: 487–499.

[11] VanTrier, J. and Symes, W., Upwind finite-difference calculation of travel times, *Geophysics,* 1991, 56:812-821.