Ordered Line Integral Methods for Solving the Eikonal Equation

Samuel F. Potter Maria Cameron

April 3, 2018

Abstract

TODO: write this.

1 Introduction

The eikonal equation. We are interested in solving the *eikonal equation*, a nonlinear PDE encountered in wave propagation and the modeling of a wide variety of problems in computational and applied science (cite). With $n \geq 2$, and given a domain $\Omega \in \mathbb{R}^n$, the eikonal equation is of the form:

$$|\nabla u(x)| = s(x), \qquad x \in \Omega, \tag{1}$$

where $s : \Omega \to \mathbb{R}_+$ is a fixed, nonnegative *slowness function*, which forms part of the problem data. Hence, we solve for $u : \Omega \to \overline{\mathbb{R}}_+$. The rest of the problem data is a subset of $D \subset \Omega$ where u has been fixed; i.e., $u|_D = g$, for some $g : D \to \overline{\mathbb{R}}_+$. As an example, if $s \equiv 1$ and $g \equiv 0$, then the solution u of eq. (1) is the distance to D at each point in Ω:

$$u(x) = d(x, D) = \inf_{y \in \Omega} ||x - y||_2 \,. \tag{2}$$

The rapid solution of this problem on complicated geometries is an example of an exceedingly practical application of the fast marching method, which sees use in areas such as computer graphics, computational geometry, computed-aided design, and the like (cite).

The fast marching method. As we have just alluded to, the fast marching method is a particularly efficient method for solving the eikonal equation and a variety of others equations besides [3]. What are other equations solved using the FMM?

To describe this method, first let $\mathcal{P} = \{p_i\}_{i \in I} \subseteq \Omega$ be a set of *nodes* where we would like to approximate the true solution u with a numerical solution $U : \mathcal{P} \to \overline{\mathbb{R}}_+$. Additionally, for each node $p \in \mathcal{P}$, define a set of neighbors, $\texttt{nb}(p) \subseteq \mathcal{P} \backslash \{p\}$. The standard fast marching method takes $\mathcal{P}$ to be a lattice in $\mathbb{R}^n$ and $\texttt{nb}(p)$ to be each node's $2n$ neighbhors. With $\mathcal{P}$ defined, we also define the *boundary nodes*, the set $\texttt{bd}$. The set $\texttt{bd}$ and D may not coincide; to reconcile this difference, the initial value of $U(p)$ for each $p \in \texttt{bd}$ must take g into account in the best way possible.

We will give a brief description of the fast marching method. There are several extra pieces of information that need to be kept track of in order to implement the algorithm. For each node p, apart from the current value of $U(p)$, the most salient piece of information is the *state* of each node p, written $p.\texttt{state} \in \{\texttt{valid}, \texttt{trial}, \texttt{far}\}$. The meaning of each of these states will become clear from the following high-level description of the algorithm:

1. For each $p \in \mathcal{P}$, initially set $p.\texttt{state} = \texttt{far}$ and $U(p) = \infty$.
2. For each $p \in \texttt{bd}$, set $p.\texttt{state} = \texttt{valid}$, initialize $U(p)$, and execute items 3c and 3d.
3. While there are `trial` or `far` nodes left:
   (a) Let p be the `trial` node with the smallest value $U(p)$.
   (b) Set $p.\texttt{state} \leftarrow \texttt{valid}$.
   (c) For each $q \in \texttt{nb}(p)$, set $p.\texttt{state} \leftarrow \texttt{trial}$ if $p.\texttt{state} = \texttt{far}$.
   (d) For each $q \in \texttt{nb}(\hat{p})$ such that $q.\texttt{state} = \texttt{trial}$, update $U(q)$.

The exact sequencing and details of some of these steps is variable. Our intent here is to give a birds-eye view of the algorithm—for concrete details, a suitable reference should be consulted (cite).

The preceding algorithm is generic in the following ways:

- There are a variety of ways to compute `bd` and subsequently approximate the initial value of U for $p \in \texttt{bd}$ using g (cite).
- The method of keeping track of the node with the smallest value is variable: most frequently, a binary heap storing pointers to the nodes is dynamically updated, leading to $O(N \log N)$ update operations, where N is the number of nodes.
- The update procedure itself can take different forms: for methods based on finite differences, there are first-order methods and "almost" higher-order methods (methods which take advantage of second- and higher-order discretizations when the nodes are available for them, and which default the lower order schemes otherwise).
- Related to the foregoing point, the arrangement of the nodes (into a grid or otherwise) varies; hence, the neighborhood of each node varies. This naturally affects the update procedure.

Minimum action integral of the eikonal equation. The eikonal equation eq. (1) is a Hamilton-Jacobi equation for u. If we let each fixed characteristic (ray) of the eikonal equation be parametrized by some parameter σ and denote $p = \dot{x} = dx/d\sigma$, the corresponding Hamiltonian is defined as:

$$\mathcal{H}(p, x) = \frac{||p||^2}{2} - \frac{s(x)^2}{2} = 0. \tag{3}$$

Since, by Eq. (3), H = 0, the Lagrangian is L = p^T \dot(x) = H = p^T\dot{x} where \dot{x} = \partial H/\partial p = p.

From Hamilton's equation, $\mathcal{H}(p, x) = p^\top \dot{x} - \mathcal{L}(x, \dot{x})$, we can see that the Lagrangian is given by:

Therefore, L = \|\dot{x}\|^2.

$$\mathcal{L}(x, \dot{x}) = p^\top \dot{x} = \dot{x}^\top \dot{x} = ||\dot{x}||^2. \tag{4}$$

= \|\dot{x}\|

From eq. (3), we can also see that $s(x) = ||p||$, since s was assumed to be nonnegative. Now, ~~applying the Cauchy-Schwarz inequality to eq. (4),~~ we obtain:

Substituting this to Eq. (4)

$$\mathcal{L}(x, \dot{x}) = \dot{x}^\top \dot{x} = ||p|| \cdot ||\dot{x}|| = s(x)\, ||\dot{x}||. \tag{5}$$

By definition of eq. (3), we have $\dot{x} = p = \nabla u(x)$. Since $\mathcal{L} > 0$, this lets us write:

$$\mathcal{L}(x, \dot{x}) = ||\dot{x}||^2 = \left|\nabla u(x)^\top \dot{x}\right| = \nabla u(x)^\top \dot{x} = \dot{u}. \tag{6}$$

Let x(\sigma) be a characteristic arriving at \hat{x} from x_0\in D, and x(0) =x_0, x(\hat{\sigma}) = \hat{x}.

Say that using eq(7) we get the multiple arrival solution by method of characteristics

And, integrating this along a fixed ray, say from σ_0 to σ_1:

$$u(\sigma_1) - u(\sigma_0) = \int_{\sigma_0}^{\sigma_1} \mathcal{L}(x, \dot{x}) d\sigma = \int_{\sigma_0}^{\sigma_1} s(x) \, ||\dot{x}|| \, d\sigma. \quad = \int_0^L s(x)\, dl \tag{7}$$

where dl is the length element, and L is the length of the characteristic from x(\sigma_0) to x(\sigma_1)

A characteristic of eq. (1) minimizes eq. (7) if the path is allowed to vary. Then, if $\hat{x}$ is fixed and $\alpha : [0, L] \to \mathbb{R}^n$ is an arc-length parametrized curve with $\alpha(L) = \hat{x}$, eq. (7) is equivalent to:

Mention that we are looking for the first arrival solution u satisfying Eq. (8).

$$\hat{u} = u(\hat{x}) = \min_{\alpha} \left\{ u(\alpha(0)) + \int_{\alpha} s dl \right\}, \tag{8}$$

where dl is the differential element of displacement along α. Our update procedure is based on eq. (8). While the standard finite difference method effectively discretizes the Hamiltonian, the method presented here discretizes eq. (8). In this way, we can see that it relates to the method of characteristics.

2 Notation

In this section, we briefly present some notation and definitions that will be used throughout. Let $n \geq 2$ be an integer and assume $d \in \{0, \ldots, n-1\}$. Let $\{p_0, \ldots, p_d\} \subseteq \mathbb{R}^n$ be a linearly independent set of vectors (implying $d < n$). We will be interested in convex combinations of these vectors. Define the set Δ^d by:

$$\Delta^d = \left\{ (\lambda_1, \ldots, \lambda_d) : \lambda_i \geq 0 \text{ for } i = 1, \ldots, d \text{ and } \sum_{i=1}^{d} \lambda_i \leq 1 \right\}.$$

For each $\lambda = (\lambda_1, \ldots, \lambda_d) \in \Delta^d$, we will write $\lambda_0 = 1 - \lambda_1 - \cdots - \lambda_d$. Then, $\sum_{i=0}^{d} \lambda_i p_i$ lies in the convex hull of $\{p_0, \ldots, p_d\}$. We will typically write this vector p_λ. If we define $\delta p_i = p_i - p_0$, we can also write $p_\lambda = p_0 + \sum_{i=1}^{d} \lambda_i \delta p_i$. This is a homogeneous transformation applied to λ; to this end, we define the matrix:

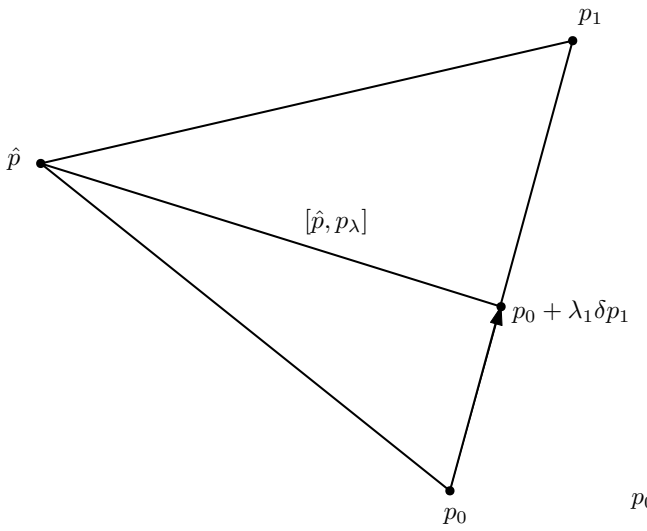
$$\delta P = \begin{pmatrix} \delta p_1 & \cdots & \delta p_d \end{pmatrix} \in \mathbb{R}^{n \times d},$$

so that we can write $p_\lambda = \delta P \lambda + p_0$. We will have reason to make use of this "δ" notation elsewhere—in general, if x is some indexed object, then $\delta x_i = x_i - x_0$.

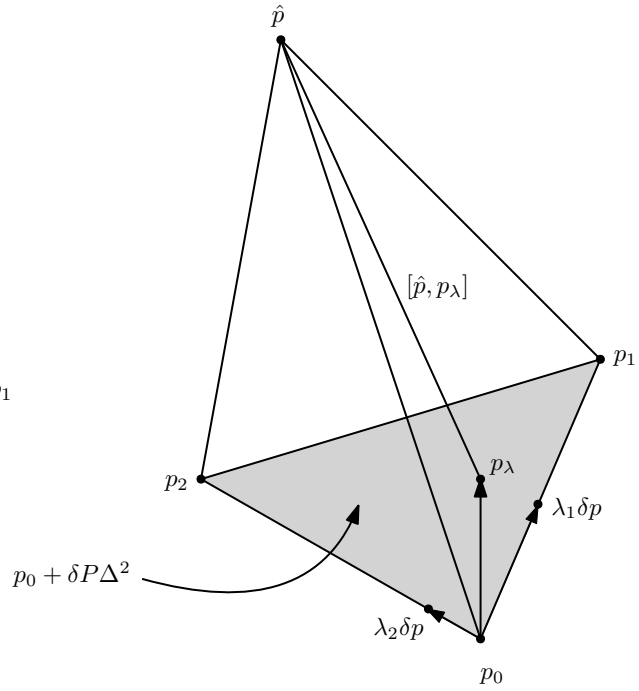
3 Quadrature Rules

Make here a reference to step 3(d) of the outline of the algorithm in Section \ref{sec:intro}.

In this section, we discuss how to compute updates. For each update, we identify the updated point with the origin. We assume our method is operating on a uniform grid with lattice constant h. The vectors $\{p_0, \ldots, p_d\}$ of the previous section are the vectors of the simplex corresponding to the current update. We assume these vectors are scaled so that, if $\hat{p}$ denotes the update grid point, then the points which lie on the (approximate) front of the solution are $\hat{p} + hp_i \in \mathbb{R}^n$ for $i = 0, \ldots, d$. This is useful because on a uniform grid, it will be convenient for us to work with vectors p_i whose components satisfy $(p_i)_j \in \{0, -1, 1\}$ for $j = 1, \ldots, n$. In particular, many of the quantities that we will encounter become integer-valued and simpler to compute digitally (e.g., the dot product between two such vectors can be computed rapidly using bitwise operations).



(a) An example with $d = 1$ (a triangle update).



(b) An example with $d = 2$ (a tetrahedron update).

Figure 1: Characteristics emanate from the set $p_0 + \delta P \Delta^d = \{p_0 + \lambda_1 \delta p_1 + \cdots + \lambda_d \delta p_d\}$, which approximates the front of the numerical solution.

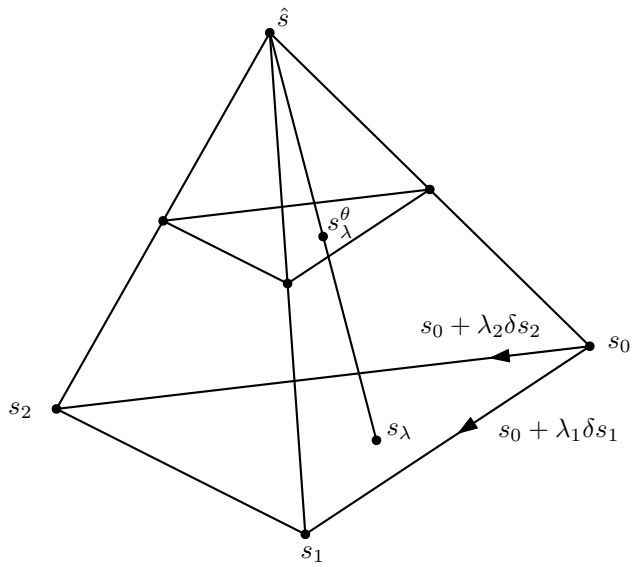


Figure 2: A depiction of the different quantities related to $s^{(\theta)}$ and $s_\lambda^{(\theta)}$ (the case of $d = 2$, a tetrahedron update).

I would remove parentheses around \theta in the superscripts.

Each simplex update approximately minimizes a line integral corresponding to a trial evolution of a characteristic curve of the numerical solution. We identify the current grid point with the origin and denote the trial value of the numerical solution at this point by $\hat{u}$. In general, we use a hat to denote quantities for the update point; e.g., below, $\hat{s} = s(\hat{p})$. We also define $u_\lambda = u_0 + \delta u^\top \lambda$. Then, the goal is to approximate:

to be updated

$$\hat{u} = \min_{\lambda \in \Delta^n} \left\{ u_\lambda + h \int_{[p_\lambda, 0]} s(\gamma(t)) dt \right\}. \tag{9}$$

using a quadrature rule, where γ is an arc length parametrization of $[p_\lambda, 0] = \mathrm{conv}(\{0, p_\lambda\})$. We consider two approximations to eq. (9): the difference between the two pertains to the way we incorporate the speed function s. Let $\theta \in [0, 1]$ and define $l_\lambda = ||p_\lambda||_2$. Then, we define:

, i,e, s^{\theta} is the approximation of s at the theta-point of the segment connecting the center of mass of the base of the simplex with the origin

$$F_0(\lambda; \theta) := u_\lambda + h s^{(\theta)} l_\lambda = u_\lambda + h \left[(1-\theta)\hat{s} + \frac{\theta}{d+1} \sum_{i=0} s_i \right] l_\lambda,$$

by linear interpolation.

$$F_1(\lambda; \theta) := u_\lambda + h s_\lambda^{(\theta)} l_\lambda = u_\lambda + h \left[(1-\theta)\hat{s} + \theta s_\lambda \right] l_\lambda.$$

i.e., s^{\theta}_{\lambda} is s evaluated at the theta-point of the segment [0,p_{\lambda}].

We will primarily concern ourselves with F_0 and F_1 for $\theta = 0$ and $\theta = 1/2$. For $\theta = 0$, we have $F_0 = F_1$, so we define $F^{(\mathrm{rhr})} = F_0 = F_1$. On the other hand, $F_0(\lambda; 1/2) \neq F_1(\lambda; 1/2)$ unless $s \equiv 1$, so we write $F_0^{(\mathrm{mp})}(\lambda) = F_0(\lambda; 1/2)$ and $F_1^{(\mathrm{mp})}(\lambda) = F_1(\lambda; 1/2)$, where "mp" stands for "midpoint".

where rhr stands for the right-hand rule.

To compute $\hat{u}$, we will need access to the gradient and Hessian of F_0 and F_1. These quantities are easy to compute, but we have found a particular form for them to be appropriate. We compute these quantities here.

Lemma 1. *The gradient and Hessian of $F_0(\lambda;\theta)$ satisfy:*

$$\nabla_\lambda F_0(\lambda;\theta) = \delta u + \frac{s^{(\theta)}h}{l_\lambda}\delta P^\top p_\lambda, \tag{10}$$

$$\nabla^2_\lambda F_0(\lambda;\theta) = \frac{s^{(\theta)}h}{l_\lambda}\delta P^\top \mathcal{P}^\perp_{p_\lambda}\delta P, \tag{11}$$

where $\mathcal{P}_{p_\lambda}$ denotes the orthogonal projector onto $\mathrm{span}(p_\lambda)$ *and $\mathcal{P}^\perp_{p_\lambda}$ is the orthogonal projector onto* $\mathrm{span}(p_\lambda)^\perp$.

Proof. For the gradient, we have:

$$\nabla_\lambda F_0(\lambda;\theta) = \delta u + \frac{s^{(\theta)}h}{2l_\lambda}\nabla_\lambda p_\lambda^\top p_\lambda = \delta u + \frac{s^{(\theta)}h}{l_\lambda}\delta P^\top p_\lambda,$$

since $\nabla_\lambda p_\lambda^\top p_\lambda = 2\delta P^\top p_\lambda$. For the Hessian:

$$\begin{aligned}\nabla^2_\lambda F_0(\lambda;\theta) &= \nabla_\lambda\left(\frac{s^{(\theta)}h}{l_\lambda}p_\lambda^\top\delta P\right) = s^{(\theta)}h\left(\nabla_\lambda\frac{1}{l_\lambda}p_\lambda^\top\delta P + \frac{1}{l_\lambda}\nabla_\lambda p_\lambda^\top\delta P\right)\\ &= \frac{s^{(\theta)}h}{l_\lambda}\left(\delta P^\top\delta P - \frac{\delta P^\top p_\lambda p_\lambda^\top\delta P}{p_\lambda^\top p_\lambda}\right) = \frac{s^{(\theta)}h}{l_\lambda}\delta P^\top\left(I - \frac{p_\lambda p_\lambda^\top}{p_\lambda^\top p_\lambda}\right)\delta P,\end{aligned}$$

from which the result follows. □

Lemma 2. *The gradient and Hessian of $F_1(\lambda;\theta)$ satisfy:*

$$\nabla_\lambda F_1(\lambda;\theta) = \delta u + \frac{h}{l_\lambda}\left(\theta p_\lambda\delta s^\top + s^{(\theta)}_\lambda\delta P\right)^\top p_\lambda, \tag{12}$$

$$\nabla^2_\lambda F_1(\lambda;\theta) = \frac{h}{l_\lambda}\left(\theta\left(\delta P^\top p_\lambda\delta s^\top + \delta s p_\lambda^\top\delta P\right) + s^{(\theta)}_\lambda\delta P^\top\mathcal{P}^\perp_{p_\lambda}\delta P\right). \tag{13}$$

Proof. Since $F_1(\lambda;\theta) = u_\lambda + hs^{(\theta)}_\lambda l_\lambda$, for the gradient we have:

$$\nabla_\lambda F_1(\lambda;\theta) = \delta u + h\left(\theta l_\lambda\delta s + \frac{s^{(\theta)}_\lambda}{2l_\lambda}\nabla_\lambda p_\lambda^\top p_\lambda\right) = \delta u + \frac{h}{l_\lambda}\left(\theta p_\lambda^\top p_\lambda\delta s + s^{(\theta)}\delta P^\top p_\lambda\right),$$

and for the Hessian:

$$\nabla^2_\lambda F_1(\lambda;\theta) = \frac{h}{2l_\lambda}\left(\theta\left(\nabla_\lambda p_\lambda^\top p_\lambda\delta s^\top + \delta s(\nabla_\lambda p_\lambda^\top p_\lambda)^\top\right) + s^{(\theta)}_\lambda\left(\frac{1}{2p_\lambda^\top p_\lambda}\nabla_\lambda p_\lambda^\top p_\lambda(\nabla_\lambda p_\lambda^\top p_\lambda)^\top - \nabla^2_\lambda p_\lambda^\top p_\lambda\right)\right).$$

Simplfying this gives us the result. □

Corollary 1. *write relationship between F_1 gradient and Hessian and F_0 gradient and Hessian as observation*

TODO: *as an aside, we could use Woodbury or Sherman-Morrison to write $\nabla^2 F_1^{-1}$ in terms of $\nabla^2 F_0^{-1}$. If we came up with a simple way to compute (or precompute and cache) $\nabla^2 F_0^{-1}$, then we might have access to a cheap Newton's method for both F_0 and F_1. If all we have to do is precompute a QR decomposition for each δP, and the resulting inverse is correct, the cost of inversion would be $O(nd + d^2) = O(n^2)$ (since $d \leq n$), where n is the dimension of the ambient space and d is the number of vectors p_i.*

Which simplex? Lemmas 1 and 2 can be formulated slightly differently. If we take Δ^d to be the set of λ lying in the nonnegative orthant of $\mathbb{R}^{d+1}$ such that $\sum_{i=0}^{d} \lambda_i = 1$, then, letting $P \in \mathbb{R}^{d+1 \times d+1}$ so that the jth column of P is p_i, we need to define $F_0(\lambda; \theta) = u^\top \lambda + s^{(\theta)} h \sqrt{\lambda^\top P^\top P \lambda}$. Then:

$$\nabla F_0(\lambda; \theta) = u + s^{(\theta)} h \frac{P^\top p_\lambda}{l_\lambda}, \qquad \text{and} \qquad \nabla^2 F_0(\lambda; \theta) = \frac{s^{(\theta)} h}{l_\lambda} P^\top \left(I - \frac{p_\lambda p_\lambda^\top}{p_\lambda^\top p_\lambda} \right) P.$$

An immediate issue here is that, following this definition, $\nabla^2 F_0$ becomes singular. Before, the size of δP allowed us to mitigate the degeneracy of the orthogonal projection matrix. This motivates our choice of convention for the convex coefficients λ, along with the fact that the smaller dimension can potentially lead to savings in time and space (however modest).

4 Ordered Line Integral Methods

To justify the present work, we start by presenting the update procedure (step 6 of the fast marching method described in section 1) for the standard fast marching method, which is based on finite differences. We then present the update for a prototypical version of an *ordered line integral method* (OLIM) and show that it is equivalent to the standard fast marching method's update. The rest of the work will then concern itself with improvements and refinements of our first OLIM.

Let $h > 0$, and assume our grid forms a subset of $h\mathbb{Z}^{n+1}$. Let $\hat{p} \in h\mathbb{Z}^{n+1}$ be the node with the minimal heap value (as in step 3 in the informal description of the algorithm); and, without loss of generality, identify $\hat{p}$ with the origin. During the update step for the fast marching method, the valid neighbors with the minimum value along each axis are extracted. For example, for each j such that $0 \leq j \leq n$, one of $\pm h e_j$ is selected, depending on the values of U and whether node is `valid`. There may be fewer than n nodes selected this way. Without loss of generality, we assume that n nodes have been selected, and that they are given by $\{h e_i\}_{i=0}^{n}$. Once applied, the update rule consists of solving the following equation for $\hat{u}$:

In is better to stick with R^n, not switch to R^{n+1}

$$\sum_{i=0}^{n} (\hat{u} - u_i)^2 = \hat{s}^2 h^2. \tag{14}$$

The value of $\hat{u}$ is given by:

$$\hat{u} = \frac{1}{n} \sum_{i=0}^{n} u_i + \frac{1}{n} \sqrt{\left(\sum_{i=0}^{n} u_j \right)^2 - n \left([\sum_{i=0}^{n} u_i^2] - \hat{s}^2 h^2 \right)}. \tag{15}$$

If the discriminant of eq. (15) is negative, a lower-dimensional update is performed a subset of the selected nodes.

We will consider the OLIMs in $\mathbb{R}^{n+1}$ with the same stencil as the (finite difference-based) fast marching method—in particular, where the stencil for each node consists of its $2(n+1)$ nearest neighbors—and using the right-hand rule, $F^{(\mathrm{rhr})}$. From eq. (9), we solve:

$$\hat{u} = \min_{\lambda \in \Delta^n} F^{(\mathrm{rhr})}(\lambda) = \min_{\lambda \in \Delta^n} \{ u_\lambda + \hat{s} h l_\lambda \}. \tag{16}$$

where we write $s^\theta = s^0 = \hat{s}$ since $\hat{s}$ corresponds with s evaluated at $\hat{p}$. For the fast marching marching method update, the method of selecting which nodes incorporate into eq. (14) is determined

by the discretization of the differential operator. The ordered line integral method is akin to the method of characteristics, and the considerations become geometric in nature. **TODO**: say more about this.

Theorem 1. *As before, without loss of generality, let the update nodes be* $\{he_i\}_{i=0}^n$. *Then, the solution of eq.* (14) *corresponds with the solution of eq.* (16)*,* ***TODO****: under certain conditions.*

Proof. First, we solve eq. (16) by the method of Lagrange multipliers. Ignoring the requirement that $\lambda_i \geq 0$ for i such that $0 \leq i \leq n$, we solve eq. (16) only subject to the requirement that $g(\lambda) = \mathbf{1}^\top \lambda - 1 = 0$. Defining the Lagrangian $L(\lambda, \alpha) = F^{(\text{rhr})}(\lambda) - \alpha g(\lambda)$, the resulting system of equations is $\alpha \mathbf{1} - u = \hat{s}h\lambda / l_\lambda$ and $\mathbf{1}^\top \lambda = 1$. Squaring the first of these gives us $||\alpha \mathbf{1} - u||^2 = \hat{s}^2 h^2$, where the dependence on λ drops out. Furthermore, letting (λ^*, α^*) be optimum, if we multiply $\nabla_\lambda L(\lambda^*, \alpha^*) = 0$ through by λ^*, we obtain:

$$0 = \nabla_\lambda L(\lambda^*, \alpha^*)^\top \lambda^* = u^\top \lambda^* - \alpha^* \mathbf{1}^\top \lambda^* + \hat{s}h \frac{l_{\lambda^*}^2}{l_{\lambda^*}} = u_\lambda^* - \alpha^* + \hat{s}h l_{\lambda^*}.$$

Hence, $\alpha^* = u_{\lambda^*} + \hat{s}h l_{\lambda^*} = \hat{u}$, and $||\hat{u}\mathbf{1} - u||^2 = \hat{s}^2 h^2$, recovering eq. (14). □

5 Comparing F_0 and F_1

Theorem 2. *For fixed problem data and s Lipschitz continuous, let* λ_0^* *be the minimizer of* F_0 *and let* λ_1^* *be the minimizer of* F_1*. Then, the error incurred by minimizing* F_0 *and then switching to* F_1 *for evaluation is* $O(h^2)$*; i..e:*

$$|F_1(\lambda_1^*; \theta) - F_1(\lambda_0^*; \theta)| = O(h^3). \tag{17}$$

Proof. todo □

6 Constrained Newton's Method for Minimizing F_0 and F_1

Except for F_0 in special cases, F_0 and F_1 are nonlinear and nonquadratic functions. The function F_0 is convex, but the same can't be said for F_1. We explored a variety of methods for minimizing these functions, and the sequential quadratic programming (SQP) framework ended up being one of the most reliable. The minimization problem to solve is:

$$\begin{aligned} &\text{minimize} && F_i(\lambda; \theta) \\ &\text{subject to} && \lambda \in \Delta^d \end{aligned}$$

for $i = 0, 1$. Let k be the iteration number and let λ_k be the kth iterate. If we expand F_i:

$$F_i(\lambda; \theta) = F_i(\lambda_k; \theta) + \nabla_\lambda F_i(\lambda_k; \theta)^\top (\lambda - \lambda_k) + \frac{1}{2}(\lambda - \lambda_k)^\top \nabla_\lambda^2 F_i(\lambda_k; \theta)(\lambda - \lambda_k) + O(||\lambda - \lambda_k||^3), \tag{18}$$

then formulating our problem as an SQP requires that we form a sequence of iterates by approximating F_i using the first three terms of eq. (18) and solving the inequality-constrained quadratic

program (QP):

$$\begin{aligned} \text{minimize} \quad & F_i(\lambda_k;\theta) + \nabla_\lambda F_i(\lambda_k;\theta)^\top(\lambda-\lambda_k) + \frac{1}{2}(\lambda-\lambda_k)^\top \nabla^2_\lambda F_i(\lambda_k;\theta)(\lambda-\lambda_k) \\ \text{subject to} \quad & \lambda \in \Delta^d. \end{aligned}$$

The solution of this problem can be carried out in a variety of ways. Since we are interested in small values of d (in particular, $d = 2$), and since the number of constraints is $d + 1 = O(d)$, a suitable approach is to use the active set method. At each step of the active set method, an equality-constrained QP is solved. For F_0, this is straightforward, and can again be accomplished in a variety of ways; for F_1, the Hessian $\nabla^2_\lambda F_1$ can be indefinite, which requires us to perturb it to ensure descent.

An informal description of the algorithm. In this section, we sketch the update algorithm in broad strokes. Since Δ^d is a polytope, we can write $\lambda \in \Delta^d$ as a linear matrix inequality:

$$A\lambda \le b, \text{ where } A = \begin{pmatrix} -I \\ \mathbf{1}_{1\times d} \end{pmatrix} \text{ and } b = \begin{pmatrix} \mathbf{0}_{d\times 1} \\ 1 \end{pmatrix}.$$

The main procedures comprising the update algorithm follow.

1. **SQP iteration.** We start by computing $G_k = \nabla^2 F_i(\lambda_k)$. If G_k is indefinite, we perturb $G_k \leftarrow G_k + \epsilon I$ so that $\lambda_{\min}(G_k + \epsilon I) > 0$. Next, we compute the gradient $g_k = \nabla_\lambda F_i(\lambda_k)$ and the load vector for the iteration, $c_k = g_k - G_k\lambda_k$. If we write $F_k = F_i(\lambda_k)$, then our quadratic approximation to our cost function can be written $\frac{1}{2}\lambda^\top G_k\lambda + c_k^\top\lambda + C_k$, where C_k is a constant. Solving the problem:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\lambda^\top G_k\lambda + c_k^\top\lambda \\ \text{subject to} \quad & A\lambda \le b, \end{aligned} \tag{19}$$

   we obtain λ^*, and write $\lambda_{k+1} = \lambda_k + \alpha_k(\lambda^* - \lambda_k)$. The quantity $p_k = \lambda^* - \lambda_k$ is the descent direction for the step, and α_k is the step length. We note that our method of solving eq. (19) recovers $-p_k$ instead of $\lambda*$. To ensure descent, we use a backtracking procedure to select α_k, starting with $\alpha_k = 1$.

2. **Active set iteration.** We solve eq. (19) using the active set method, which is similar to the simplex algorithm. Our method follows the procedures outlined in Chapter 16 of Nocedal and Wright [2] and Chapter 2 of Bertsekas [1]. The key detail here is that, at each iteration of the active set method, an equality-constrained QP is solved. The indices of the constraints (i.e., the rows of $A\lambda \le b$) are $\{1,\dots,d+1\}$. A constraint is active for an iterate if it holds with equality. Let $I \subseteq \{1,\dots,d+1\}$ be the index of active constraints for a given iterate, and let A_I denote the submatrix consisting of the rows of A indexed by I; define b_I likewise. Then, the equality-constrained QP is:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\lambda^\top G_k\lambda + c_k^\top\lambda \\ \text{subject to} \quad & A_I\lambda = b_I. \end{aligned} \tag{20}$$

   There are several direct methods for solving equality-constrained QPs. Their relative merits depend on the problem being solved. We discuss our choice next.

3. **Solving the equality-constrained quadratic subproblem.** If a basis for the so-called nullspace of A_I is easily obtained, then one method that is of practical use is the nullspace method, described in section 16.2 of Nocedal and Wright [2]. A brief description follows. The problem given by eq. (20) is equivalent to the system:

$$\begin{pmatrix} G_k & A_I^\top \\ A_I & \end{pmatrix} \begin{pmatrix} -p_k \\ \gamma^* \end{pmatrix} = \begin{pmatrix} g_k \\ h_I \end{pmatrix}, \tag{21}$$

where λ^*, γ^* are the solution and Lagrange multipliers for eq. (20), $p_k = \lambda^* - \lambda_k$, g_k is as given above, and $h_I = A_I\lambda_k - b_k$. If we let Z be a nullspace matrix for A_I (i.e. such that $A_I Z = 0$), and let Y be a matrix such that $\begin{pmatrix} Y & Z \end{pmatrix}$ is full rank, blocking p_k so that:

$$p_k = \begin{pmatrix} Y & Z \end{pmatrix} \begin{pmatrix} p_Y \\ p_Z \end{pmatrix} = Yp_Y + Zp_Z$$

and substituting this into eq. (21) yields the system:

$$\begin{aligned} A_I Y p_Y &= -h_I, \\ Z^\top G_k Z p_Z &= -Z^\top G_k Y p_Y - Z^\top g_k. \end{aligned}$$

The key to the efficiency of this method in our case is that the simplicity of A leads to simple expressions for Z and Y, which allows these two systems to be solved rapidly.

In the following lemma we show that Y and Z can take very simple forms (although these forms are not unique). Using these forms, multiplication by Y, Z, and $Z^\top$ amount to indexing operations, and—if $d{+}1 \in I$—taking simple differences of elements. **TODO**: *MATLAB experiments tell me that indexing written in matrix form is compatible with solving a system using a Cholesky factorization or a QR decomposition. This is extremely useful! I don't think I've checked if it also works for the case when $d+1 \in I$, but I think something straightforward should be possible.*

Lemma 3. *Let $I \subseteq \{1, \ldots, d+1\}$, let $m = |I|$, and let A be the constraint matrix for Δ^d. Then a nullspace matrix $Z \in \{0, -1, 1\}^{d \times d-m}$ for A is given explicitly by the following two cases. In both cases, let $j_1 < \cdots < j_{d-m}$ be the sorted elements of $\{1, \ldots, d\} \setminus I$:*

1. *If $I \subseteq \{1, \ldots, d\}$, then $Z = \begin{pmatrix} e_{j_1} & \cdots & e_{j_{d-m}} \end{pmatrix}$.*
2. *If $d+1 \in I$, then $Z = \begin{pmatrix} e_{j_2} - e_{j_1} & \cdots & e_{j_{d-m}} - e_{j_1} \end{pmatrix}$.*

Proof. **TODO**: easy. □

Corollary 2. *For each Z in lemma 3, a matrix Y such that $\begin{pmatrix} Y & Z \end{pmatrix}$ is full rank can be written such that its columns are distinct columns of the identity matrix. In particular, for each case of lemma 3:*

1. *If $I \subseteq \{1, \ldots, d\}$, let $i_1 < \cdots < i_m$ be the sorted elements of I. Then, choose $Y = \begin{pmatrix} e_{i_1} & \cdots & e_{i_m} \end{pmatrix}$.*
2. *If $d+1 \in I$, let $i_1 < \cdots < i_{m-1}$ be the sorted elements of $I \setminus \{d+1\}$. Then, $Y = \begin{pmatrix} e_1 & e_{i_1} & \cdots & e_{i_{m-1}} \end{pmatrix}$ suffices.*

7 Enumerating Tetrahedra

References

[1] D.P. Bertsekas. *Nonlinear programming.* Athena Scientific, 1999.

[2] J. Nocedal and S. Wright. *Numerical optimization.* Springer, 2006.

[3] J.A. Sethian. *Level set methods and fast marching methods: evolving interfaces in computational geometry, fluid mechanics, computer vision, and materials science*, volume 3. Cambridge University Press, 1999.

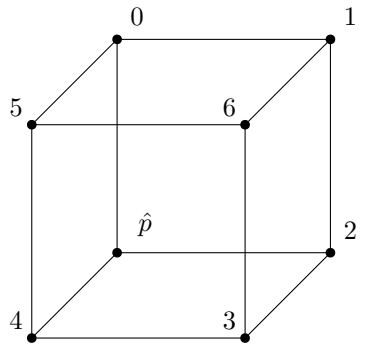


Figure 3: Numbering scheme for an update octant. The node $\hat{p}$ is having its value updated. The diagonally opposite node is the sixth (last) node, with the other six nodes numbered 0–5 cyclically.

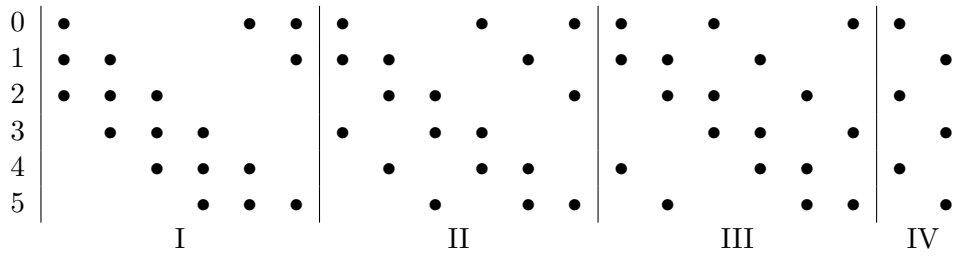


Figure 4: Enumeration of tetrahedra that do not involve the sixth node. These are OLIM18's only tetrahedra, and are common to OLIM26. The tetrahedra (0, 1, 2), (2, 3, 4), and (4, 5, 0) in group I are degenerate and can be omitted; the other three must be included. All of the tetrahedra in groups II and III have a pairwise distance that is greater than $h\sqrt{2}$ and can be omitted. The remaining tetrahedra in group IV must be included.

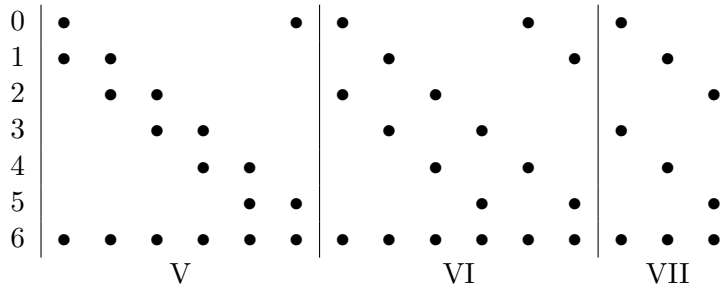
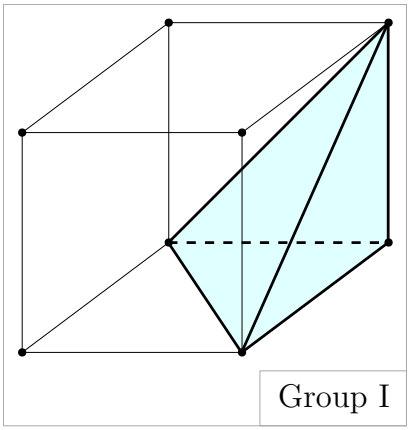
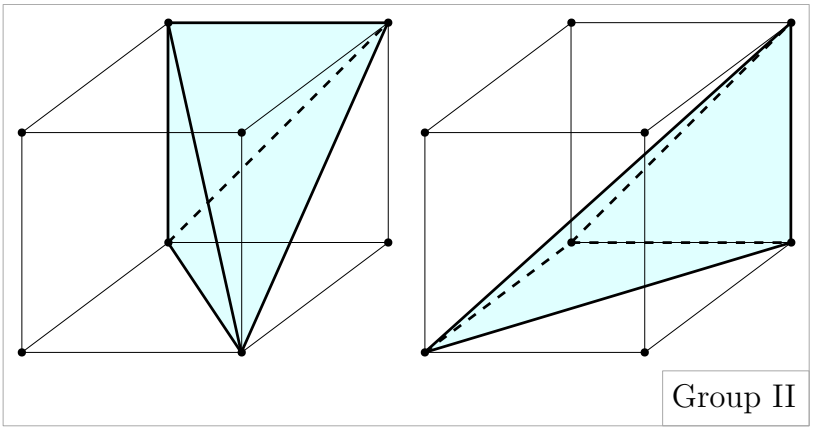


Figure 5: Enumeration of tetrahedra involving the node diagonally opposite $\hat{p}$—i.e., OLIM26-only tetrahedra. All of the tetrahedra in groups V and VI have pairwise distances no greater than $h\sqrt{2}$ and must be included. The tetrahedra in group VII are all degenerate and can be omitted.



Group I



Group II