Ordered Line Integral Methods for Solving the Eikonal Equation

Samuel F. Potter† Maria K. Cameron§

February 20, 2019

Abstract

The eikonal equation is used to model high-frequency wave propagation and solve a variety of applied problems in computational science. We present a family of fast and accurate Dijkstra-like solvers for the eikonal equation and factored eikonal equation, which compute solutions on a regular grid by solving local variational minimization problems. Our methods converge linearly but compute significantly more accurate solutions than competing linear methods, due to improved directional coverage and the use of more accurate quadrature rules. In 3D, we present two different families of algorithms which significantly reduce the number of FLOPs needed to obtain an accurate solution to the eikonal equation. One method employs a fast search using local characteristic directions to prune unnecessary updates, and the other uses the theory of constrained optimization to achieve the same end. The proposed solvers are more efficient than the standard fast marching method in terms of the relationship between error and CPU time. We also modify our method for use with the additively factored eikonal equation, which can be solved locally around point sources to maintain linear convergence. We conduct extensive numerical simulations and provide theoretical justification for our approach. A library that implements the proposed solvers is available online.

Keywords. ordered line integral method, eikonal equation, factored eikonal equation, simplified midpoint rule, semi-Lagrangian method, fast marching method

AMS subject classifications. 65N99, 65Y20, 49M99

1 Introduction

We develop fast, memory efficient, and accurate solvers for the eikonal equation, a nonlinear hyperbolic PDE encountered in high-frequency wave propagation [17] and the modeling of a wide variety of problems in computational and applied science [43], such as photorealistic rendering [20], constructing signed distance functions in the level set method [33], solving the shape from shading problem [26, 36, 16], traveltime computations in numerical modeling of seismic wave propagation [44, 34, 24, 52, 53], and others. We are motivated primarily by problems in high-frequency acoustics [37], which are key to enabling a higher degree of verisimilitude in virtual reality simulations (see [39, 40] for a cutting-edge time-domain approach which is useful up to moderate frequencies). Current approaches to acoustics simulations rely on methods whose complexity depends on the highest frequency of the sound being simulated. For moderately high-frequency wave propagation problems, the eikonal equation comes about as the first term in an asymptotic WKB
expansion of the Helmholtz equation and corresponds to the first arrival time of rays propagating under geometric optics, although approaches for computing multiple arrivals exist [19].

In this work, we develop direct solvers for the eikonal equation which are fast and accurate, particularly in 3D. We develop two separate families of algorithms which approach the problem of efficiently computing updates in 3D in separate ways. These algorithms are analyzed and extensive numerical studies are carried out. Both approaches use knowledge of the problem gleaned from its Lagrangian representation to reduce the amount of computational work necessary to compute an accurate result. These approaches are competitive with existing direct solvers for the eikonal equation and readily generalize to related equations (the static Hamilton-Jacobi equation) and higher dimensions. This research was conducted in tandem with research into ordered line integral methods for computing the quasipotential of nongradient stochastic differential equations (SDEs) [13, 12, 54]; however, because of the comparative simplicity of the eikonal equation, we were led to develop two separate families of algorithms (one based on the original quasipotential ordered line integral method, but modified) that were more amenable to analysis, which allowed us to obtain theoretical results that justify our experimental findings.

1.1 Results

Different numerical methods have been proposed for the solution of the eikonal equation; generally, there are direct solvers and iterative solvers. The most popular direct solvers are based on Dijkstra’s algorithm (we refer to these as “Dijkstra-like” from now on) [51, 42], and the most popular iterative method is the fast sweeping method [50, 56]. In this work, we develop a family of Dijkstra-like solvers for the eikonal equation in 2D and 3D, similar to the fast marching method (FMM) or ordered upwind methods [42, 46]. These solvers come about by discretizing and minimizing the action functional for the eikonal equation (see appendix A). We investigate three different quadrature rules for discretizing this line integral: a righthand rule (rhr), a simplified midpoint rule (mp0), and a midpoint rule (mp1). We also consider different ways of organizing a grid point’s neighborhood into triangles and tetrahedra, simplifying and accelerating our solvers by avoiding redundant and unnecessary computations, particularly in 3D. Additionally, we modify our algorithm to solve the additively factored eikonal equation [27]: to enhance accuracy, we solve the locally factored eikonal equation near point sources, which recovers the global $O(h)$ error convergence expected from a first-order method, where $h > 0$ is the uniform spacing between grid points. This fixes the degraded $O(h \log h^{-1})$ convergence often associated with point source eikonal problems [38] (see [56] for a proof of this error bound).
Our main results follow:

- For 3D problems, we develop two separate algorithms: a bottom-up (olim3d) algorithm, and a top-down algorithm (olimK, where $K = 6, 18, 26$ is the size of neighborhood used). Each algorithm locally updates a grid point by performing a minimal number of triangle or tetrahedron updates. Depending on the quadrature rule, each update is calculated by solving a system of nonlinear equations either directly (rhr and mp0) or iteratively (mp1).

- We note that this work was done in tandem with research on ordered line integral methods for computing the quasipotential 3D for nongradient SDEs $[13, 54, 12]$. Unlike the quasipotential, the eikonal equation is simple enough to allow us to analyze and justify our algorithms. We are also able to obtain simpler solution methods and established performance guarantees. We prove theorems relating our quadrature rules, rigorously justifying the mp0 rule, establishing it as superior to mp1.

- We conduct extensive numerical tests on a variety of problems, including point source problems for different slowness (index of refraction) functions, and multiple point source problems with a linear speed function. These problems have analytical solutions, which we use as a ground truth.

- We show that a significant improvement in accuracy is gained over the equivalent of the standard fast marching method in 3D (olim6_rhr), and that only modest slowdown is incurred, so that our algorithms with improved directional coverage and quadrature rules are competitive. See fig. 1 to see the improvement of olim3d_mp0 over olim6_rhr, and see section 5 for more details.

- We use Valgrind [31] to profile our implementation and show that the time spent sorting the heap used to order nodes on the front is negligible for all practical problem sizes. Since our solvers otherwise run in $O(N^n)$ time, where $n$ is the dimension of the domain, we suggest that the $O(N^n \log N)$ cost of the algorithm is pessimistic. Memory access patterns play a significant role in scaling.

1.2 Accessing our library

We carefully implement the algorithms described in this paper using the C++ language [49]. Our library, libolim, can be accessed from S. Potter’s website [35]. The project website includes instructions for downloading and running basic examples [1]. The code used to generate plots in this paper is also available with instructions [2].

2 Background

We now provide a brief overview of the eikonal equation and its numerical solution on a regular grid. We then review the different algorithms available for its solution, and sketch a generic Dijkstra-like algorithm which we will refer to throughout the paper to organize our results. Note that the FMM could be described by this generic algorithm equally well. Since we focus on Dijkstra-like algorithms here, our explanation of the fast sweeping and other iterative methods will be brief.

2.1 The eikonal equation

With $n \geq 2$, and given a domain $\Omega \subset \mathbb{R}^n$, the eikonal equation is:

$$\|\nabla u(x)\| = s(x), \quad x \in \Omega,$$

where $\|\cdot\|$ denotes the $\ell_2$ norm unless otherwise stated, and $s : \Omega \to (0, \infty)$ is a fixed, positive slowness function, which forms part of the problem data. Hence, we solve for $u : \Omega \to \mathbb{R}_+$. The rest of the problem data is a subset $D \subset \Omega$ where $u$ has been fixed; i.e., $u|_D = g$ for some $g : D \to \mathbb{R}_+$. As an example, if $s \equiv 1$ and $g \equiv 0$, then the solution of eq. (1) is:

$$u(x) = d(x, D) = \min_{y \in D} \|x - y\|.$$  

(2)
That is, $u$ is the distance to $D$ at each point in $\Omega$.

To numerically solve eq. (1), first let $G = \{p_i\} \subseteq \Omega$ be the set or grid of nodes where we would like to approximate the true solution $u$ with a numerical solution $U : G \to \mathbb{R}_+$. Additionally, for each node $p \in G$, define a set of neighbors, $\text{nb}(p) \subseteq G \setminus \{p\}$. Typically—for the FMM and olim6, for instance—$G$ is taken to be a subset of a lattice in $\mathbb{R}^n$ and $\text{nb}(p)$ to be each node’s $2n$ von Neumann neighbors. With $G$ defined, we also define the set of boundary nodes, $\text{bd} \subseteq G$. It may happen that the set $\text{bd}$ and $D$ do not coincide (e.g., $D$ could be a curve which does not intersect $G$); to reconcile this difference, the initial value of $U(p)$ for each $p \in \text{bd}$ must take $g = u|_D$ into account in the best way possible. This problem has been approached in different ways, and is not the focus of the present work [10].

Throughout, we make several simplifying assumptions.

- All boundary nodes coincide with grid points: $\text{bd} = D \subseteq G$.
- The grid $G$ is a regular, uniform grid (a subset of a regular, uniform square lattice in 2D or cubic lattice in 3D). We denote grid nodes by $x \in G$.
- When numerically computing a new value at a grid point $\hat{x} \in G$, we transform the neighborhood to the origin and scale the vertices so that they have integer values. The transformed update node is labeled $\hat{p}$. See section 3.1 for a detailed explanation.

2.2 Dijkstra-like algorithms

If we order nodes in $G$ so that new solution values are only computed using upwind nodes, the eikonal equation can be solved directly; i.e., without the use of an iterative solver. This means that new values of the solution are only involve nodes that have fixed smaller values. This is done using a variant of Dijkstra’s algorithm for finding shortest paths in a network. Other algorithms which solve similar network flow problems can also be used, but have different complexity guarantees [8]. In particular, Dijkstra’s algorithm is a type of label-setting method for finding shortest paths in a network; there are also label-correcting methods [5].
Using Dijkstra’s algorithm to solve a “continuous shortest path” problem has been discovered in several contexts. The earliest such development is a theoretical result in computational geometry due to Mitchell, Mount, and Papadimitriou, who used this idea to compute exact polyhedral shortest paths (“discrete geodesics”) on triangulated surfaces [30]. This was followed by Tsitsiklis who developed a first-order semi-Lagrangian method for solving isotropic optimal control problems on a uniform grid [51]. Finally, the fast marching method, which uses a first-order upwind finite difference scheme was developed by Sethian for isotropic front propagation [42]. Many variations of these methods have since been developed [46, 22]. Our own development resembles Tsitsiklis’s, but extends it past its original formulation.

To write down a generic Dijkstra-like algorithm, there are several pieces of information which need to be kept track of. A data structure we call front that stores an expanding wavefront is maintained throughout the algorithm’s execution. For each node \(p\), apart from the current value of \(U(p)\), the most salient piece of information is its state, written \(p.state \in \{\text{valid, trial, far}\}\). To fix ideas, consider the following high-level Dijkstra-like algorithm:

**Algorithm 1** A generic Dijkstra-like algorithm for solving the eikonal equation.

1. For each \(p \in G\), set \(p.state \leftarrow \text{far}\) and \(U(p) \leftarrow \infty\).
2. For each \(p \in \text{bd}\), set \(p.state \leftarrow \text{trial}\), and set \(U(p)\) to a user-defined value.
3. While there are trial nodes left in \(G\):
   1. Let \(p_{\text{new}}\) be the trial node in front with the smallest value \(U(p_{\text{new}})\).
   2. Set \(p_{\text{new}}.state \leftarrow \text{valid}\) and remove \(p_{\text{new}}\) from front.
   3. For each \(\hat{p} \in \text{nb}(p_{\text{new}})\), set \(\hat{p}.state \leftarrow \text{trial}\) if \(\hat{p}.state = \text{far}\).
   4. For each \(\hat{p} \in \text{nb}(p_{\text{new}})\) such that \(\hat{p}.state = \text{trial}\), update \(U = U(\hat{p})\) and merge \(\hat{p}\) into front.

Specifying how item 3d is to be performed is the crux of developing a Dijkstra-like algorithm and is left intentionally vague. This step involves indicating how nodes in \(\text{nb}(\hat{p})\) are used to compute \(U\), and how they are organized into the front data structure. The FMM uses an upwind finite difference scheme where only valid nodes are used to compute \(U\), and where nodes on the front are sorted using an array-based heap implementing a priority queue [42]. As an example, Tsitsiklis’s algorithm combines nodes in valid into sets whose convex hulls approximate the surface of the expanding wavefront and then solves local raytracing problems where rays emanate from these surfaces. The method presented here operates in the same way (see fig. 2). For specific details, a general reference should be consulted [43]. In addition to item 3d, algorithm 1 is generic in the following ways:

- As we mentioned before, there are different ways of computing bd and subsequently approximating the initial value of \(U\) for \(p \in \text{bd}\) using \(g = u|_D\) [10]
- How we keep track of the node with the smallest value is variable: most frequently, as in Dijkstra’s algorithm, a heap storing pointers to the nodes is used, leading to \(O(N^2 \log N)\) update operations overall, where \(N\) is the number of nodes. In fact, there are \(O(N^n)\) variations using Dial’s algorithm (a bucketed version of Dijkstra’s algorithm), but these have not been used as extensively as Dijkstra-like algorithms [51, 23, 55].
- The arrangement of the nodes into a grid or otherwise varies, as do the neighborhoods of each node. This affects the update procedure. A regular grid is simple to deal with, but Dijkstra-like methods have been extended to manifolds and unstructured meshes, where the situation is more involved [25, 45, 7].

Other problems can be solved using Dijkstra-like algorithms: the static Hamilton-Jacobi equation, an anisotropic generalization of the eikonal equation, can be solved using the ordered upwind method [46] and other recently introduced methods [29, 28]. Consequently, the quasipotential of a nongradient stochastic differential equation can be computed using the ordered line integral method, although the considerations may be more involved [13, 12, 54].
2.3 Fast sweeping methods

Another approach to solving a discretized version of eq. (1) is the fast sweeping method \cite{50, 56}. Unlike Dijkstra-like methods, which are direct solvers, the fast sweeping method is an iterative solver using an upwind scheme and rotating sweep directions. For problems where the characteristics of the solution change direction infrequently, the algorithm obtains $O(N^n)$ complexity. A drawback of the method is that the constant cannot be bound a priori and depends heavily on the geometry of the problem. Fast sweeping methods have been extended to Hamilton-Jacobi equations \cite{50, 22}, and hybrid methods combining the fast sweeping method with a Dijkstra-like method have been introduced recently \cite{8, 9}.

3 Ordered line integral methods for the eikonal equation

The fast marching method \cite{42} discretizes the eikonal equation, eq. (1), and solves this discretization in an upwind fashion to compute $\hat{U}$. Throughout, we distinguish between the exact solution $u$ and the numerical solution $\hat{U}$, where $\hat{U}$ will always denote the current value to be computed; likewise, any quantity with a hat (‘) will denote a quantity evaluated at the node being updated. The ordered line integral method locally and approximately minimizes the minimum action integral of eq. (1):

$$\hat{u} = \min_\alpha \left\{ u_0 + \int_\alpha s(x) dt \right\},$$

where $\alpha$ is a ray parametrized by arc length, $\hat{x}$ is a target point, $\hat{u} = u(\hat{x})$, and $u_0 = u(\alpha(0))$ (see appendix A for a derivation of eq. (3)). By contrast, Lagrangian methods (i.e., raytracing methods) trace a bundle of rays from a common locus by integrating eq. (3) for different initial conditions. In this section, we describe our approach to discretizing and approximately minimizing eq. (3). As we mentioned in section 2.2, to compute $\hat{U} = U(\hat{p})$ in item 3d of algorithm 1, we need to approximately minimize several instances of eq. (3); details of this procedure will be discussed in section 4. In this section, we focus on a single instance of the discretized version of eq. (3), presenting our notation, deriving preliminary results, and describing the different quadrature rules we use, detailing useful exact solutions to the approximate minimization of eq. (3). We then present theoretical results.

3.1 Notation

When numerically minimizing eq. (3), we minimize a line integral whose starting point is constrained to lie on the base of a simplex (lines, triangles, and tetrahedra in 2D and 3D). This is depicted in fig. 3. Since we solve eq. (1) on a regular grid, there are only a small number of distinct simplexes that we need to treat in our implementation, each with fixed geometry but varying data. Because of this, it is beneficial for us to rescale the vertices of each simplex to lie in $\mathbb{Z}^n$ and translate each so that the point being updated lies at the origin.

Let $n$ and $d$ be integers such that $n \geq 2$ and $0 \leq d < n$. We think of $n$ as the ambient dimension and $d$ as the affine dimension of the facet of the simplex where the line integral originates (e.g., $d = 2$ for a tetrahedron). Let $p_0, \ldots, p_d \in \mathbb{Z}^n$ be linearly independent. For each update, we shift the coordinates so that the update node $\hat{p}$ coincides with the origin: $\hat{p} = 0$. Then, the vectors $\hat{p}, p_0, \ldots, p_d$ are the vertices of a simplex with affine dimension $d$. In this work, we only use uniform square or cubic grids; hence, we only consider vectors $p_i$ which satisfy $\|p_i\|_{\infty} = 1$ (larger neighborhoods are required for a Hamilton-Jacobi solver). Our results here are straightforwardly generalized to nonuniform rectangular grids. For each $i = 0, \ldots, d$, we also define $x_i$ to be the preimage of $p_i$ before scaling and translation is performed (i.e., $x_i$ lies in $h\mathbb{Z}^n$). The numerical solution $U$ and the slowness function $s$ are evaluated at the points $x_i$; we write $U_i = U(x_i)$ and $s_i = s(x_i)$. For a fixed update, we “forget” $x_i$ and think of $U_i$ and $s_i$ being “evaluated” at $p_i$. For a given $n$, it is necessary to do updates for simplexes with affine dimension $d = 0, \ldots, n - 1$. We will cover this in more detail in section 4: presently, we will derive the updates for general dimension since there is no need to specialize. So, for this section, we simply assume that $d = n - 1$. 

6
(a) A depiction of the different quantities related to \( s^0 \) and \( s^\lambda \) for the case of \( d = 2 \), a tetrahedron update. Both of \( s^0 \) and \( s^\lambda \) live on the \( \theta \)-section of the simplex \( \text{conv}\{\hat{p}, p_0, p_1, p_2\} \). The function \( s^\lambda \) is a linear combination of \( \hat{s}, s_0, s_1, \) and \( s_2 \); the value \( s^0 \) is \( s^\lambda \) evaluated at the centroid of the \( \theta \)-section.

(b) An example showing an update interval for \( d = 2 \) (a tetrahedron update).

Figure 3: Characteristics emanate from \( p_0 + \delta P \Delta^d \), which approximates the front of the solution. These figures depict the quantities involved in a tetrahedron update.

We define the set \( \Delta^d \), the parameter space for the simplex, by:

\[
\Delta^d = \{(\lambda_1, \ldots, \lambda_d) : \lambda_i \geq 0 \text{ for } i = 1, \ldots, d \text{ and } \lambda_1 + \cdots + \lambda_d \leq 1 \}. \tag{4}
\]

We will also occasionally write \( \Delta^d \) as a linear matrix inequality:

\[
\lambda \in \Delta^d \iff A\lambda \leq b, \quad \text{where } A = \begin{bmatrix} -I_d & 1_d \end{bmatrix} \in \mathbb{R}^{(d+1) \times d}, \quad \text{and } b = \begin{bmatrix} 0_d \\ 1 \end{bmatrix} \in \mathbb{R}^{d+1}. \tag{5}
\]

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 denote this vector by \( p_\lambda \) (see fig. 3). Defining \( \delta p_i = p_i - p_0 \), we can also write \( p_\lambda = p_0 + \sum_{i=1}^d \lambda_i \delta p_i \) and define the matrix:

\[
\delta P = [\delta p_1 \cdots \delta p_d] \in \mathbb{R}^{n \times d}, \tag{6}
\]

so that we can write \( p_\lambda = p_0 + \delta P \lambda \). Along the same lines, \( U_\lambda = U_0 + \delta U^\top \lambda \), where \( \delta U_i = U_i - U_0 \); we also define \( s_\lambda = s_0 + \delta s^\top \lambda \), where \( \delta s_i = s_i - s_0 \). Although the slowness function \( s \) may be available to us in analytic form, we assume that it is only available at the nodes in \( \mathcal{G} \) to reflect use in real-world applications (e.g., \( s \) will most likely be presented to us as gridded data of some sort).

### 3.2 Quadrature

To approximately minimize eq. (3), we assume that \( \alpha \) is a line segment parametrized by arc length and then apply several different one-point quadrature rules to the resulting integral. If \( \alpha \) is a straight line segment connecting \( p_\lambda \) and \( \hat{p} = 0 \) and \( l_\lambda = \|p_\lambda\| \), then:

\[
\hat{U} = U(\hat{p}) = \min_{\lambda \in \Delta^d} \left\{ U_\lambda + h \int_0^{l_\lambda} s(\alpha(t)) dt \right\}. \tag{7}
\]
We consider two approximations to eq. (7). For \( \theta \) such that \( 0 \leq \theta \leq 1 \), we define:

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

\[
F_1(\lambda) = F_1^\theta(\lambda) = U_\lambda + s^\theta h l_\lambda := U_\lambda + \left[\left(1 - \theta\right) \hat{s} + \theta s_\lambda \right] h l_\lambda,
\]

where we omit the superscript in \( F_\theta \) when the context is clear. See fig. 3a for depiction of this setup. The geometric sense of \( \theta \) and the aggregates in eqs. (8) and (9) is as follows:

- The parameter \( \theta \) is a convex coefficient that selects a section of \( \Delta^d \) parallel to its base. It is used to linearly interpolate between \( \hat{s} \) and the interpolated value of \( s \) at the base of \( \Delta^d \).

- The slowness is interpolated differently for \( F_0 \) and \( F_1 \). For \( F_0 \), we use the sample mean of \( s_0, \ldots, s_d \); for \( F_1 \), we linearly interpolate between \( s_0, \ldots, s_d \) using the convex coefficient vector \( \lambda \).

We primarily concern ourselves with \( F_0 \) and \( F_1 \) where \( \theta = 0 \) and \( \theta = 1/2 \). If \( \theta = 0 \), then \( F_0 \equiv F_1 \), so we define a single quadrature rule to capture this case, which we call \( \text{rrr} \). If \( \theta = 1/2 \), then \( F_0 \neq F_1 \) in general, leading us to define separate midpoint quadrature rules for each of \( F_0 \) and \( F_1 \), named \( \text{mp0} \) and \( \text{mp1} \), respectively. The case of \( \text{mp0} \) requires special care and is handled in section 3.4.

### 3.3 The minimization problem

With \( F_0 \) and \( F_1 \) so defined, the minimization problem which approximates eq. (7) to compute \( \hat{U} \) for a fixed update simplex is:

\[
\hat{U} = \min_{\lambda \in \Delta^d} F_i(\lambda).
\]

This is a nonlinear, constrained optimization problem with linear inequality constraints and no equality constraints (cf. eq. (5)). We require the gradient and Hessian of \( F_0 \) and \( F_1 \) for our algorithms and analysis. These are easy to compute, but we have found a particular form for them to be convenient for both implementation and analysis. The proofs of all propositions and lemmas in this section can be found in appendix B.

**Proposition 1.** The gradient and Hessian of \( F_0(\lambda; \theta) \) are given by:

\[
\nabla F_0(\lambda) = \delta U + s^\theta h P^\perp \nu_\lambda,
\]

\[
\nabla^2 F_0(\lambda) = \frac{s^\theta h}{l_\lambda} P^\perp P_{\nu_\lambda} P^\perp,
\]

where \( \nu_\lambda = p_\lambda/l_\lambda \) is the unit vector in the direction of \( p_\lambda \), \( P_{\nu_\lambda} = \nu_\lambda \nu_\lambda^\perp \) denotes the orthogonal projector onto \( \text{span}(p_\lambda) \) and \( P_{\nu_\lambda}^\perp = I - \nu_\lambda \nu_\lambda^\perp \) is the orthogonal projector onto \( \text{span}(p_\lambda)^\perp \), the orthogonal complement of \( \text{span}(p_\lambda) \).

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

\[
\nabla F_1(\lambda) = \delta U + \theta h l_\lambda \hat{s} + s^\theta h P^\perp \nu_\lambda,
\]

\[
\nabla^2 F_1(\lambda) = \left\{ \delta P^\perp \nu_\lambda, \theta h \delta s \right\} + \frac{s^\theta h}{l_\lambda} P^\perp P_{\nu_\lambda} P^\perp P_{\nu_\lambda} P^\perp,
\]

where \( \{a, b\} = ab^\perp + ba^\perp \) is the anticommutator of two vectors.

Our task is to minimize \( F_0 \) and \( F_1 \) over the convex set \( \Delta^d \); so, we need to determine whether \( F_0 \) and \( F_1 \) are convex functions. The next two lemmas address this point.

**Lemma 1.** Let \( p_0, \ldots, p_d \) form a nondegenerate simplex (i.e., \( p_0, \ldots, p_d \) are linearly independent) together with \( \hat{p} \) and assume that \( s \) is positive. Then, \( \nabla^2 F_0 \) is positive definite and \( F_0 \) is strictly convex.
For $F_1$, we can only obtain convexity (let alone strict convexity) for $h$ sufficiently small. For large enough $h$, we will encounter nonconvex updates. To obtain convexity, we need to stipulate that the slowness function $s$ be Lipschitz continuous on $\Omega$ with a Lipschitz constant that is independent of $h$. In practice, we have not found this to be a particularly stringent restriction.

**Lemma 2.** In the setting of lemma 1, additionally assume that $s$ is Lipschitz continuous with Lipschitz constant $K \leq C$ on $\Omega$, for some constant $C > 0$ independent of $h$. Then, $\nabla^2 F_1$ is positive definite (hence, $F_1$ is strictly convex) for $h$ small enough.

We have found that all $mp1$ updates become strictly convex problems rapidly as $h \to 0$. The reason for this is discussed at the end of section 3.7.

### 3.4 Validation of $mp0$

If we consider an update simplex defined by the nodes $p_0, \ldots, p_{n-1}$, each subset of these nodes defines another, lower-dimensional update simplex, which lies on the boundary of the original simplex. For example, three triangle updates form the boundary of a single tetrahedron update. We expect the values of $F_i$ to be the same on the surface of the neighborhood without regard to the simplex under consideration (note that the update simplexes share common lower-dimensional boundaries). Likewise, we expect $F_i$ to be continuous as we transfer between adjacent simplexes. For $F_1$ this is the case, but for $F_0$ with $\theta \neq 0$, the function $F_0$ may not be well-defined or continuous at boundaries. This problem can lead to inconsistent and divergent solvers. A heuristic fix is to first minimize eq. (10) for $F_0$ with nonzero $\theta$ to obtain the minimizing argument $\lambda_i^*$, then set $\hat{U} = F_1(\lambda_i^*)$; i.e., to replace $F_0$ with $F_1$ for the purposes of evaluation, or to use $F_0$ as a surrogate when minimizing $F_1$.

If we use Newton’s method to minimize $F_1$, starting from $\lambda_i^*$ and letting $\lambda_i^*$ denote the optimum of eq. (10) for $F_1$, then we can use the convergence theory of Newton’s method to bound the distance between $\lambda_i^*$ and $\lambda_i^*$, thereby bounding the error incurred by using $mp0$ instead of $mp1$.

**Theorem 1.** Using lemma 2, let $h$ be sufficiently small so that $F_1$ is strictly convex. Then, the error $\lambda_i^* = \lambda_i^* - \lambda_i^*$ satisfies $\|\lambda_i^*\| = O(h)$. Further, if we let $\lambda_0 = \lambda_i^*$ in the following Newton iteration:

$$
\lambda_{k+1} \leftarrow \lambda_k - \nabla^2 F_1(\lambda_k)^{-1} \nabla F_1(\lambda_k), \quad k = 0, 1, \ldots, (15)
$$

then this iteration is well-defined, and converges quadratically to $\lambda_i^*$. This immediately implies that the error incurred by $mp0$ is $O(h^3)$ per update compared to $mp1$; i.e.:

$$
|F_1(\lambda_i^*) - F_1(\lambda_i^*)| = O(h^3). (16)
$$

**Proof.** The proof of theorem 1 is detailed in appendix C. \qed

We can provide some intuition for why this bound is satisfactory. If we assume that our domain is spanned along a diameter by $O(N)$ nodes, and that $h \sim N^{-1}$, then we can anticipate $O(N)$ downwind updates, starting from $bd$ and extending to the boundary of $G$ in any direction. Accumulating the error over these nodes, we can expect the maximum pointwise error between a solution to eq. (1) computed by using $mp0$ and $mp1$ to be $O(h^2)$, which is dominated by the $O(h)$ discretization error coming from the linear convergence of the method itself. Hence, using $mp0$ instead of $mp1$ to find the parameter $\lambda$ (i.e., determine the local characteristic) should introduce no significant error when evaluating $F_1$.

### 3.5 Exact solution for $rhr$ and $mp0$

Since $F_0$ is strictly convex, $\nabla F_0(\lambda) = 0$ is sufficient for the optimality of $\lambda$, if we ignore the constraint $\lambda \in \Delta^d$. The unconstrained system of nonlinear equations defined by $\nabla F_0(\lambda) = 0$ can be solved exactly without an iterative solver. We can compute the solution using the reduced QR decomposition of $\delta P$ and by considering the problem’s geometry (see fig. 17). This is captured in the following theorem. We will discuss how to use this theorem efficiently in a solver in section 4.4.
Theorem 2. Let $\delta P = QR$ be the reduced QR decomposition of $\delta P$; i.e., where $Q \in \mathbb{R}^{n \times d}, R \in \mathbb{R}^{d \times d}, Q^T Q = I_d$, and with $R$ upper triangular. For $s^0, h$, and $U$ fixed, if $\lambda^* = \arg\min_{\lambda \in \mathbb{R}^n} F_0^0(\lambda)$, then:

\[
l_{\lambda^*} = \sqrt{\frac{p_0^T (I - QQ^T) p_0}{1 - \|R^{-1} \frac{\partial U}{\partial s} \|^2}}, \tag{17}
\]

\[
\lambda^* = -R^{-1} \left( Q^T p_0 + l_{\lambda^*} R^{-\top} \frac{\delta U}{s^0 h} \right), \tag{18}
\]

\[
\hat{U} = U_0 + \frac{s^0 h}{l_{\lambda^*}} p_0^\top p_{\lambda^*}. \tag{19}
\]

Proof. See appendix D.

3.6 Equivalence of the upwind finite difference scheme and $F_0$

If we linearly approximate $U$ near $\hat{p}$, then for $i = 0, \ldots, n - 1$, we find that $\hat{U}$ approximately satisfies:

\[
U_i - \hat{U} = \nabla \hat{U}^\top p_i. \tag{20}
\]

This finite difference approximation to eq. (1) can be solved exactly and is a known generalization of the upwind finite difference scheme used by the fast marching method to an unstructured mesh [25, 45]. Computing $\hat{U}$ using this approximation is equivalent to solving:

\[
\hat{U} = \min_{\lambda \in \Delta^n} F_0(\lambda) \tag{21}
\]

in a sense made precise by the following theorem.

Theorem 3 (Equivalence of upwind finite difference scheme and $F_0$). Let $\hat{U}$ by the solution of eq. (20) and let $\hat{U}' = \min_{\lambda \in \mathbb{R}^n} F_0(\lambda)$. Then, $\hat{U}$ exists if and only if $\|R^{-\top} \delta U\| \leq s^0 h$, and can be computed from:

\[
\hat{U} = U_i - p_i^\top Q R^{-\top} \delta U + \|p_{\min}\| \sqrt{(s^0 h)^2 - \|R^{-\top} \delta U\|^2}, \tag{22}
\]

where $p_{\min} = (I - QQ^T) p_i$ (for any $i$, see fig. 17 in appendix D). Additionally, the following hold:

1. The finite difference solution and line integral solution coincide: i.e., $\hat{U} = \hat{U}'$ can be computed from:

\[
\hat{U} = U_i + s^0 h p_i^\top \nu_{\lambda^*}, \tag{23}
\]

where $\lambda^* = \arg\min_{\lambda \in \mathbb{R}^n} F_0(\lambda)$ and $\nu_{\lambda^*} = p_{\lambda^*}/l_{\lambda^*}$.

2. The characteristics found by solving the finite difference problem and minimizing $F_i$ coincide and are given by $[p_{\lambda^*}, \hat{p}] = [p_{\lambda^*}, 0]$.

3. The approximated characteristic passes through $\text{conv}(\{p_0, \ldots, p_{n-1}\})$ if and only if $\lambda^* \in \Delta^n$.

Proof. See appendix E.

3.7 Causality

Dijkstra-like methods are based on the idea of monotone causality, similar to Dijkstra’s method itself. To compute shortest paths in a network, Dijkstra’s method uses dynamic programming to compute globally optimal shortest paths using local information [15]. In this way, the distance to each downwind vertex must be greater than its upwind neighboring vertices. To ensure convergence to the correct viscosity solution, our scheme must be consistent and monotone [11]. Our OLIMs using the rhr quadrature rule inherit the
consistency and causality of the finite difference methods which they are equivalent to if they use the same 4 (in 2D) or 6 (in 3D) point neighborhoods. Since we consider many different update neighborhoods involving distinct simplexes, we provide a simple way of checking whether each simplex is causal.

The causality of an update depends on the underlying simplex and the problem data. In particular, an update is causal for $F_i$ if:

$$\hat{U} = F_i(\lambda^*_i) \geq \max_i U_i.$$  \hspace{1cm} (24)

It is enough to determine whether or not each type of update simplex admits only causal updates, which relates to whether the simplex is acute.

We also consider something we refer to here as the "update gap": the difference $\hat{U} - \max_i U_i$. As discussed in Tsitsiklis’s original paper [51], an alternative to Dijkstra’s algorithm is Dial’s algorithm—a bucketed version of Dijkstra’s algorithm which runs in $O(N^n)$ time, where the constant depends on the bucket size [14, 23]. In this case, the size of the buckets is determined by the update gap. It is unclear whether there is any real advantage of a Dial-like solver (see [21] for a discussion); indeed, we present numerical evidence suggesting that this is not the case in section 5.1. Despite this, the update gap is of fundamental importance and limits the number of nodes that can be processed in parallel without violating causality.

**Theorem 4.** For $\nu_i = p_i / \|p_i\|$, an update simplex is causal for $F_0$ if and only if $\nu_i^\top \nu_j \geq 0$ for all $i$ and $j$ such that $0 \leq i < n$ and $0 \leq j < n$. If we assume that $s$ is Lipschitz continuous, for $h$ small enough, the simplex is also causal for $F_1$, and the term in $F_1$ which prevents an update from being causal decays with order $O(h^2)$. Furthermore, the update gap is given explicitly by:

$$\hat{U} - \max_i U_i = s^0 h \min_{i,j} \frac{\nu_i^\top \nu_j}{\|p_i\|} \hspace{1cm} (25)$$

**Proof.** See appendix F. \hfill $\square$

The fact that our methods are causal for all practical problem sizes follows from the fact that the term preventing causality decays rapidly—see eq. (89). This can be seen easily by rewriting $F_1(\lambda)$ as $F_0(\lambda) + s$ a small perturbation (which is $O(h^2)$) and using the Lipschitz continuity of $s$.

### 3.8 Local factoring

Near rarefaction fans, for example if $D$ is a point source or the domain contains obstacles with corners, the rate of convergence of the eikonal equation is diminished. For the eikonal equation with point source data and constant slowness, this degrades the rate of convergence to $O(h \log h^{-1})$ [38, 56]. Different factored eikonal equations which treat this problem have been developed [18, 27]. In this section, we show how the ordered line integral method can be easily adapted to additive factoring, and provide numerical tests that show that it recovers the expected linear rate of convergence for factored problems. Our focus is locally factored point sources, but this approach can be applied to the globally factored equation and other types of rarefaction fans occurring at corners or discontinuities [38].

Let $x^o \in \Omega$ be the location of a point source so that $\partial D = \{x^o\}$ and let the grid $\mathcal{G} \subseteq \mathbb{Z}^n$ be aligned so that $p^o \in \mathbb{Z}^n$; that is, we let $p^o$ be the image of $x^o$ in the update neighborhood, in the same way that each $p_i$ is the image of a grid point $x \in \mathcal{G}$ (hence, $p^o$ varies from update to update, but $x^o$ remains the same). For a point $p_\lambda$, define $l^o_\lambda = \|p_\lambda - p^o\|$ and let $s^o = s(x^o)$. The additive factorization of $U$ around $x^o$ is [27, 38]:

$$U(x) = T(x) + \tau(x), \quad \text{where} \quad T(x) = s^o \|x - x^o\|, \hspace{1cm} (26)$$

i.e. $u_\lambda = T_\lambda + \tau_\lambda$ where $T_\lambda = s^o h l^o_\lambda$. Our original definition of $F_\lambda^\theta$ was such that $\hat{U} = F_\lambda^\theta (\lambda^*)$. We will define $G_\lambda^\theta$ analogously. Letting $\tau_\lambda = \tau_0 + \delta \tau^\top \lambda$, where $\tau_0$ and $T_0$ are the values of $\tau$ and $T$ at $p_i$ for each $i$, we define:

$$G_0(\lambda) = G_0^\theta(\lambda) = \tau_\lambda + T_\lambda + s^0 h l_\lambda, \hspace{1cm} (27)$$

$$G_1(\lambda) = G_1^\theta(\lambda) = \tau_\lambda + T_\lambda + s^h h l_\lambda. \hspace{1cm} (28)$$
Like with $F_0$ and $F_1$, the only difference between $G_0$ and $G_1$ is between the terms containing $s^\theta$ and $s^\phi$.

To solve the factored eikonal equation, we choose a factoring radius $r^\circ$, replacing $F_i$ with $G_i$ in eq. (10) for nodes which lie within a distance $r^\circ$ of $x^\circ$. For constant slowness, the effect of this is to solve eq. (1) exactly inside of the locally factored region. For clarity, this is depicted in fig. 4. Algorithm 2 can be applied to solve eq. (10) for factored nodes. The gradient and Hessian of $G_i$ are simple modifications of the gradient and Hessian for $F_i$.

**Lemma 3.** The gradient and Hessian of $G_i$, for $i = 0, 1$, are given by:

$$
\nabla G_i(\lambda) = \nabla F_i(\lambda) - \delta \tau + \frac{s^\theta}{r^\circ} \delta P^T (p_\lambda - p^\circ),
$$

(29)

$$
\nabla^2 G_i(\lambda) = \nabla^2 F_i(\lambda) + \frac{s^\theta}{r^\circ} \delta P^T \delta P^\perp_{p_\lambda - p^\circ}.
$$

(30)

## 4 Implementation of the ordered line integral method

In this section, we describe our “top-down” and “bottom-up” algorithms. We emphasize the 3D solver, since in 2D, the distinction between the two is less important. Each algorithm reduces the number of updates that are done without degrading solution accuracy by using an efficient enumeration or search of the neighboring simplexes. The difference between the algorithms is in how this is done.

In section 4.1, we start by showing how to enumerate update tetrahedra and put them into separate groups of congruent tetrahedra. By only performing updates from these groups, we obtain top-down algorithms with stencils of different sizes. Our numerical tests (section 5) will show how neighborhoods of different sizes lead to different patterns of directional coverage, which can significantly affect the error. We also discuss the update gaps attainable using these tetrahedron groups in section 4.2.

Following this, we describe our bottom-up algorithm in section 4.3, which involves first finding the minimal update of smallest dimension ($d = 0$, a line update), then finding the minimal update of the next highest dimension ($d = 1$, a triangle update) which is incident to the original update, and so on. In 3D, this means finding the minimal line update, doing neighboring triangle updates which contain the minimal line update, and then neighboring tetrahedron updates which contain the minimal triangle update. We can think of this algorithm as a fast search for the first arrival characteristic.

To minimize the number of updates that are done, it is important to take advantage of the structure of the underlying constrained optimization problems that are being solved in order to skip unnecessary lower
Figure 5: Neighborhoods for the top-down family of algorithms. Algorithms olim4 and olim8 are 2D solvers and the rest are 3D solvers. The color coding of tetrahedron updates is the same for this figure and fig. 6 below.

Figure 6: Numbering scheme for update groups for the top-down solvers. In this diagram, $\hat{p}$ is being updated. The diagonally opposite node is the sixth (last) node, with the other six nodes numbered 0–5 cyclically.

Figure 7: Tables of update groups. These tables should be scanned columnwise: each column of dots selects a different tetrahedron. Tetrahedra (0, 1, 2), (2, 3, 4), and (4, 5, 0) in group I and all tetrahedra in group VII are degenerate and can be omitted.
or higher dimensional updates. We describe this procedure in section 4.4. How this is done varies depending on the choice of quadrature rule (mp0, mp1, or rhr) and type of algorithm (top-down or bottom-up).

4.1 Simplex enumeration for the top-down algorithm

When a node is first removed from front and has just become valid (item 3a), an isotropic solver must do updates involving, at the very least, the node’s $2n$ von Neumann neighbors. We can use larger neighborhoods to improve the accuracy of the result. Doing so does not necessarily improve the order of convergence of the solver, but can significantly improve the accuracy of the solution. For all of the solvers considered in this paper, in 3D, we only ever consider neighborhoods with at most 26 neighbors.

For the top-down solver, we simplify things by treating a node’s neighboring octants separately. That is, we iterate over each octant and do all updates that lie inside that octant before moving onto the next. To do this, we enumerate all update tetrahedra with vertices $p \in \{0, 1\}^3$ in a symmetric fashion. Since we assume our update tetrahedra have been translated so that $\hat{p} = 0 \in \mathbb{Z}^n$ (see section 3.1), this means enumerating $\binom{7}{3} = 35$ choices of vertices. Some choices lead to degenerate tetrahedra (i.e., such that $p_0, p_1, p_2$ are linearly dependent), so the number of nondegenerate update tetrahedra is fewer than 35 per octant. This makes it reasonable to write out the update procedure as straight-line code.

We enumerate the tetrahedra in a type of “shift-order” (see, e.g., [4])—that is, we start with an unseen bit pattern, and group this pattern together with all of its shifts (with rotation). This groups the tetrahedra into sets that are rotationally symmetric about the diagonal of the octant. In our implementation, we conditionally compile different groups so that no unnecessary branching is done. This is done using C++ templates [49]. Example stencils for the versions of olim6, olim18, and olim26 that are used for our numerical test are shown in fig. 5. The tetrahedron groups are shown in figs. 6 and 7.

4.2 Update gaps for tetrahedron groups

If we apply theorem 4 to the tetrahedron groups enumerated in figs. 6 and 7, we get the following update gaps (ignoring the $s^\theta h$ factor):

| Group     | Update Gap |
|-----------|------------|
| I         | $1/\sqrt{2}$ |
| II        | $1/\sqrt{2}$ |
| III       | $1/\sqrt{2}$ |
| IVa       | 0           |
| V         | $1/\sqrt{3}$ |
| VIa       | 0           |
| VIb       | $2/\sqrt{3}$ |
| IVb       | $1/\sqrt{2}$ |

The idea of the update gap is first explored in Tsitsiklis’s original paper [51]; in this work, the fact that Group IVa has no update gap and that the update gap of Group V is $1/\sqrt{3}$ is noted and an $O(N^n)$ algorithm based on Dial’s algorithm is presented using Group V for the update tetrahedra. This same observation is made in a more recent paper explicitly detailing a method based on Dial’s algorithm [23]. A method based on a combination of tetrahedra groups will have as its update gap the minima of each of the individual groups’ update gaps. We note here that a solver based on a combination of Groups I and VIb has a larger update gap than a solver based on Group V. This should have a positive impact on the performance of any parallel Dijkstra-like method.

4.3 The search procedure used by the bottom-up algorithm

Another approach is to take advantage of the fact that lower dimensional updates provide some information about the likely direction of arrival of the first arrival time characteristic. If we know where the minimum line update is, then the characteristic is nearby. Starting with the minimum line update, we enumerate neighboring vertices and perform the corresponding triangle updates, then enumerate vertices which are sufficiently close to the minimum triangle update, doing any relevant tetrahedron updates along the way.

While the top-down algorithm is parametrized by the choice of tetrahedron groups to include, the bottom-up algorithm is parametrized by the norms used to search for neighboring vertices as well as the permitted search radii. In 3D, let $p_0$ be the vertex that admits the minimum line update; then, $p_1$ must satisfy
Figure 8: The three types of neighborhoods for the bottom-up algorithm with $q_1 = 1 = q_2$, $d_1 = 1$, and $d_2 = 2$. The yellow and blue regions indicate where triangle and tetrahedron updates may be performed, respectively. For instance, with $p_0$ the minimizing line update vertex, candidates for $p_1$ consist of the yellow nodes: triangle updates involving these candidates and $p_0$ will be performed. Once a yellow node ($p_1$) has been selected, tetrahedron updates involving the neighboring blue nodes (candidates for $p_2$) will be performed. Note that the updates performed correspond roughly to a combination of groups I, V, VIa, and VIb.

4.4 Minimization algorithms and skipping updates

Performing an update is the same as solving eq. (10) for fixed problem data. When $F_i = F_0$, we can use theorem 2 or theorem 3 to compute $\lambda^*_0$. In this case, $\lambda^*_0$ may lie outside $\Delta^d$. On the other hand, if $F_i = F_1$, we need to use an algorithm that can solve the constrained optimization problem defined by eq. (10). Our approach has been to use sequential quadratic programming (SQP), although there are many other options. It is possible to skip some updates; and, indeed, the performance of our algorithms depends on this happening. We skip updates in three different ways. The first two strategies for skipping updates are used by the top-down algorithms, and the third is used by the bottom-up family of algorithms.

Top-down constrained skipping When computing an update using a constrained solver, we can rule out all incident lower-dimensional updates, since we have computed the global constrained optimum on $\Delta^d$.

Top-down unconstrained skipping If we do an update using an unconstrained solver, then depending on where the optimum $\lambda^*_0$ lies, we can skip some lower-dimensional updates. The idea is simple: since $F_0$ is strictly convex, if we consider a straight line starting at $\lambda^*_0$ and extending in some direction, then $f$ restricted to that line is monotonically increasing as we move away from $\lambda^*_0$. Hence, for a tetrahedron update, if $\lambda^*_0 \notin \Delta^2$, then we can skip all updates which are not “visible” (in parameter space) from $\lambda^*_0$. This is illustrated in the fig. 9.

Bottom-up KKT skipping We can also skip higher-dimensional updates. For example, if we do the three triangle updates on the boundary of a tetrahedron update, we can use the Karush-Kuhn-Tucker necessary conditions for optimality of a constrained optimization problem to determine if the minimizer
on the boundary is also a global minimizer for the constrained minimization problem given by eq. (10). Let \( L(\lambda, \mu) = F_i(\lambda) + (A \lambda - b)^\top \mu \) be the Lagrangian function, where \( \mu \in \mathbb{R}^{d+1} \) is the vector of Lagrange multipliers. Since \( F_0 \) is strictly convex and since we assume \( h \) is small enough for \( F_1 \) to be strictly convex, if \( \lambda^* \) lies on the boundary of \( \Delta^d \), we only need to check that the optimum Lagrange multipliers \( \mu^* \) are dual feasible; i.e., whether \( \mu^* \geq 0 \) [6, 32].

For \( \lambda^* \in \partial \Delta^d \), letting \( I = \{ i : (A \lambda - b)_i = 0 \} \) be the set of active constraints’ indices, stationarity requires:

\[
A^\top I \mu^*_I = \nabla F_i(\lambda). \tag{31}
\]

See eq. (5) for the definition of \( A \); the notation \( A_I \) refers to the submatrix consisting of rows of \( A \) indexed by \( I \). For a tetrahedron update in 3D, \( A \in \mathbb{R}^{3 \times 2} \) and \( |I| \leq 2 \) (not all three constraints be active simultaneously). In particular, if \( i \notin I \), then \( \mu^*_i = 0 \), and otherwise, \( \mu^*_i \) can be computed easily from eq. (31). Once the full vector of Lagrange multipliers has been computed, if \( \mu^* \geq 0 \), then the update may be skipped. A modified version of this strategy for skipping updates was used in our work on computing the quasipotential for nongradient SDEs in 3D [54].

### 4.5 The top-down and bottom-up algorithms

To describe our top-down algorithm (see algorithm 2), we define:

\[
\mathcal{V}_d = \{ \{ p_0, \ldots, p_d \} : p_i \text{ state = valid for } i = 0, \ldots, d, \\
\text{and } \{ p_0, \ldots, p_d \} \text{ belongs to the selected update group,} \\
\text{and } p_{\text{new}} \in \{ p_0, \ldots, p_d \} \}
\tag{32}
\]

for \( d = 0, \ldots, n - 1 \). These sets collect all possible simplex updates: i.e., updates which both belong to a group as defined in section 4.1 and are valid. The third condition is an important optimization. To see why it is correct, fix an update set \( \mathcal{V}_d \). If \( \{ p_0, \ldots, p_d \} \) satisfies the first two conditions but not the third, we can see that \( \hat{p} \) would have already been updated from it in a previous iteration. All new information affecting \( \hat{U} \) during this iteration must be computed from an update involving \( p_{\text{new}} \).
### Algorithm 2 The top-down hierarchical algorithm for computing \( U(\hat{p}) \) (item 3d of algorithm 1).

**Input:** the neighboring update points \((p_0, \ldots, p_d)\), and, for \(i = 0, \ldots, d\), the downwind solution value \(U_i = U(p_i)\) and the slowness \(s_i = s(p_i)\).

**Output:** a new solution value \( \hat{U} = U(\hat{p}) \), where \( \hat{U} > U_i \) for \(i = 0, \ldots, d\).

1. Set \( \hat{U} \leftarrow \infty \).
2. Initialize \( V_d \) according to eq. (32) for each \(d = 0, \ldots, n - 1\).
3. For \(d = n - 1\) down to 0:
   (a) For each \((p_0, \ldots, p_d) \in V_d\):
      i. If \( F_i = F_0 \) (mp0 or rhr):
         A. Compute \( U \) for \((p_0, \ldots, p_d)\) using theorem 2 or theorem 3.
         B. Remove updates from \( V_0, \ldots, V_{d-1} \) by visibility (see fig. 9).
      ii. Otherwise, if \( F_i = F_1 \) (mp1):
         A. Compute \( U \) by solving eq. (10) numerically (we use SQP).
         B. Remove all lower-dimensional updates from \( V_0, \ldots, V_{d-1} \).
      iii. Set \( \hat{U} \leftarrow \min(\hat{U}, U) \).

### Algorithm 3 The bottom-up hierarchical algorithm for computing \( U(\hat{p}) \) (item 3d of algorithm 1).

**Input:** for \(i = 0, \ldots, d\), the update point \(p_i\), the values \(U_i = U(p_i)\), and \(s_i = s(p_i)\).

**Output:** the new solution value \( \hat{U} = U(\hat{p}) \).

1. Set \( \hat{U} \leftarrow \infty \) and \(p_0 \leftarrow p_{\text{new}}\).
2. For \(i = 1, \ldots, n - 1\):
   (a) For each valid \(p_i\) close enough to \(p_0, \ldots, p_{i-1}\) (see section 4.3), do the update corresponding to \((p_0, \ldots, p_i)\) and keep track of the minimizing \(\lambda^* \in \Delta^i\). This update can optionally be skipped by first computing \(\mu^* \) corresponding to the optimum of the incident lower-dimensional update \((p_0, \ldots, p_{i-1})\) and checking if \(\mu^* \geq 0\).
   (b) Let \(p_i\) be the node which forms the update with the minimum value.
   (c) If \( F_i = F_0 \) (mp0 or rhr), compute \( U \) for \((p_0, \ldots, p_i)\) using theorem 2 or theorem 3.
   (d) Otherwise, if \( F_i = F_1 \) (mp1), compute \( U \) for \((p_0, \ldots, p_i)\) by solving eq. (10).
   (e) Set \( \hat{U} \leftarrow \min(\hat{U}, U) \).

The bottom-up algorithm (algorithm 3) builds up each update \((p_0, \ldots, p_d)\) one vector at a time by searching for adjacent minimizing updates of higher dimension. The optimization involving \( p_{\text{new}} \) described above can be incorporated by initially setting \(p_0 \leftarrow p_{\text{new}}\).

## 5 Numerical Results

We test on several different slowness functions with available exact solutions for point source data, and a linear speed function (i.e., \(1/s\)) which has been shown to be amenable to local factoring. For each quadrature rule described in section 3.2 (mp0, mp1, or rhr), we have two 2D algorithms, olim4 and olim8, corresponding to 4- and 8-point stencils, respectively. Since there is no advantage in 2D, we don’t apply the top-down or bottom-up approaches. In 3D, we have three top-down algorithms: olim6 (group IVa), olim18 (groups I, IVa, and IVb), and olim26 (group V). We also test the bottom-up algorithm olim3d (see fig. 8).
Figure 10: Slowdown incurred by using olim6_rhr instead of the FMM. From left to right: 1) the ratio of runtimes versus $N$, 2) the total CPU runtime of each solver. We compare results on two different computers: “2.2 GHz” is a 2015 MacBook Air with a 2.2 GHz Intel Core i7 CPU, 8 GB of 1600 MHz DDR3 RAM, a 256 KiB L2 cache, and a 4 MiB L3 cache; “4.6 GHz” is a custom built workstation running Linux with a 4.6 GHz Intel Core i7 CPU, 64 GB of 2133 MHz DDR4 RAM, a 1536 KiB L2 cache, and 12 MiB L3 cache. Both computers have 32 KiB L1 instruction caches and data caches. The plots here use our standard $\Omega = [-1, 1]^3$ domain discretized into $N = 2^p + 1$ nodes in each direction, with $s = 1$ and a point source at the origin.

Figure 11: Percentage of time spent on different tasks as determined by profiling. The “update” tasks and “heap” tasks are clearly defined, while the “logic” task contains a variety of things related to control-flow, finding neighbors, and memory movement—basically, the parts of algorithm 1 that don’t clearly pertain to computing new $\hat{U}$ values or keeping front updated. From these plots, it is clear that memory speed plays a large role in determining efficiency. To some extent, even though the more complicated update procedures are slower, their slowness is hidden somewhat by memory latency as problem sizes grow. For large $N$ and some solvers (the middle and right plots), “heap” takes too little time, and is not picked up by the profiler.
5.1 Implementation Notes

Before describing our numerical tests, we briefly comment on our implementation and make some observations about its performance. A discussion of some of the choices that we made in our implementation follows:

- We precompute and cache all values of \( s \) on the grid \( \mathcal{G} \), as opposed to reevaluating \( s \), because we assume that \( s \) will be provided as gridded data (consider, e.g., the shape from shading problem [26], where the input data is an image).

- We maintain \texttt{front} using a priority queue implemented using an array-based heap, which is updated using the \texttt{sink} and \texttt{swim} functions described in Sedgewick and Wayne [41].

- We store \texttt{front} as a dense grid of states: for each node in \( p \in \mathcal{G} \), we track \( p.\text{state} \) for all time for every node. We could implement a sparse \texttt{front} using a hash map or a quadtree or octree, which would save space, but would also be much slower to update. In fact, since updating these data structures frequently takes \( O(\log n) \) time, using a sparse front has the potential to degrade the overall complexity to \( O(n \log^2 n) \).

We use a policy-based design [3] written in the C++ programmed language which makes heavy use of templates. This allows us to conditionally compile different features and reuse logic to implement different Dijkstra-like algorithms. In particular, we implement the standard FMM [42] and make a direct comparison between it and the ordered line integral method which it is equivalent to, \texttt{olim6rhr} (see fig. 10). We have found that only a modest slowdown is incurred by using \texttt{olim6rhr} for problems of moderate size. The disparity between the two is greater for smaller problem sizes, which is due to cache effects.

Using Valgrind [31], we profiled running our solver on the numerical tests below for different problem sizes and categorized the resulting profile data. See fig. 11. The “update” task corresponds to time spent actually computing updates, the “logic” task is a grab bag category for time spent on program logic, and “heap” corresponds to updating the array-based heap which implements \texttt{front}. Since the asymptotic complexity of the “update” and “logic” sections is \( O(N^n) \), and since “heap” is \( O(N^n \log N) \), we can see from fig. 11 that since so little time is spent updating the heap, the algorithm’s runtime is better thought of as \( O(N^n) \) for practical problem sizes. This is a consequence of using an array-based heap, which is cheap to update, and a dense grid of states, which can be read from and written to in \( O(1) \) time.

5.2 Slowness functions with an analytic solution for a point source

Using eq. (1) directly, a simple recipe to create pairs of slowness functions and solutions is to prescribe a continuous function \( u \) with level sets homeomorphic to balls and compute \( s(x) = ||\nabla u(x)||_2 \) analytically, which is valid for a single point source at the origin. Such tests allow us to observe the effect of local factoring, and to see how \texttt{mp0}, \texttt{mp1}, and \texttt{rhr} compare. The following table lists our test functions:

| Name | \( u(x) \) | \( s(x) \) |
|------|-------------|-------------|
| s1   | \( \cos(r) + r - 1 \) | \( 1 - \sin(r) \) |
| s2   | \( \frac{r^2}{2} \) | \( r \) |
| s3   | \( S(x)^\top AS(x) \) | \( \alpha \|\text{diag}(C(x))(A + A^\top)S(x)\| \) |
| s4   | \( \frac{1}{2}x^\top A^{1/2}x \) | \( \|x\|_A = \sqrt{x^\top Ax} \) |

We assume that \( x \in \Omega = [-1, 1]^3 \). We also define \( r = ||x|| \), and vector fields \( S(x) = (\sin(\alpha x_i))^3_{i=1} \) and \( C(x) = (\cos(\alpha x_i))^3_{i=1} \); we take \( \alpha = \pi/3 \). For \texttt{s3} and \texttt{s4}, we assume that \( A \) is symmetric positive definite. In 3D, the matrices we use for \texttt{s3} and \texttt{s4} are:

\[
A_{s3} = \begin{bmatrix}
1 & 1/4 & 1/8 \\
1/4 & 1 & 1/4 \\
1/8 & 1/4 & 1
\end{bmatrix} = A_{s4}^{1/2} \tag{33}
\]
Figure 12: Relative $\ell_\infty$ error plotted against CPU runtime in seconds. The domain is $\Omega = [-1,1]^3$ discretized uniformly in each direction into $N = 2^p + 1$ points, where $p = 3, \ldots, 9$, so that there are $N^3$ points overall. The slowness functions used are listed in section 5.2. We note that the horizontal and vertical axes of each subplot are the same.
Figure 13: Relative $\ell_\infty$ error plotted versus $N$. The setup is the same as in fig. 12, except that $p = 3, \ldots, 8$, so that the largest $N$ is 257 instead of 513. For olim26 and olim3d, we can see that mp0 is initially less accurate than mp1 but quickly attains parity, in accordance with theorem 1. For olim6 and olim18, the error is the same between mp0 and mp1 for all slowness functions, so these plots overlap.
Our results are displayed in figs. 12 and 13. We include plots of relative $\ell_\infty$ error plotted versus problem size and time, as well as $\ell_\infty$ error plotted versus $N$. We summarize our observations:

- For rhr, as we increase directional coverage ($\text{olim6\_rhr} \rightarrow \text{olim18\_rhr} \rightarrow \text{olim26\_rhr}$), the error constant does not improve; in fact, for $s_1$, $s_3$, and $s_4$, increasing the directional coverages causes the accuracy to deteriorate (see fig. 13). This can be due to the fact that the errors due to quadrature and due to linear interpolation have different signs, and may partially compensate each other (e.g., in $\text{olim6}$), and that this balance may worsen with increased directional coverage, leading to a reduction in interpolation error. On the other hand, using one of the midpoint rules allows improved directional coverage to translate into an improved error constant.

- If we scan each graph horizontally, we can see that the difference in error between $\text{mp0}$ and $\text{mp1}$ is minimal. For each $\text{mp1}$ graph, the corresponding $\text{mp0}$ graph has the same error, but is shifted to the left, reflecting the fact that the $\text{mp0}$ OLIMs are substantially faster. This is consistent with theorem 1, which justifies the use of $\text{mp0}$.

- With respect to the choice of neighborhood, $\text{olim6}$ is the fastest; and, for each choice of neighborhood, $\text{mp0}$ provides the best combination of speed and accuracy. If we are willing to pay somewhat in speed, we can dramatically improve the error constant by improving the directional coverage and using a solver like $\text{olim3d\_mp0}$. This tradeoff is more pronounced for smaller problem sizes. A theme running through this work is that, as the problem size increases, memory access patterns come to dominate the runtime, and the disparity between the faster and slower neighborhoods becomes less pronounced. To see this, compare the start of each graph in the top-left of the plots, and their ends in the bottom-right. We can observe, e.g., that the maximum horizontal distance between starting points and ending points has decreased significantly, which confirms this observation.

- Our high accuracy algorithms allow us to obtain a better solution on rough grids: this is helpful since opportunities to refine the mesh are limited in 3D. Discretizing $\Omega = [-1,1]^2$ in each direction into $N = 2^{14} + 1$ nodes requires about as much memory as discretizing $\Omega = [-1,1]^3$ with $N = 2^9 + 1$, which leads to $h$ being 32 times smaller in 2D than in 3D.

5.3 A linear speed function

We consider a problem that has a known analytical solution and has been used as a test problem for other factored eikonal equation solvers before\footnote{We thank D. Qi for helpful discussions regarding this problem.} [47, 18, 38]. For a single point source at $x_i$ and a vector $v$, we define:

$$
\frac{1}{s(x)} = \frac{1}{s(x_i)} + v^\top (x - x_i), \quad (34)
$$

where $s_i = s(x_i)$. The analytic solution to eq. (1) for a single source and slowness function given by eq. (34) is [47]:

$$
u_i(x) = \frac{1}{\|v\|} \cosh^{-1}\left(1 + \frac{s_i}{2} s(x) \|v\| \|x - x_i\|^2 \right). \quad (35)$$

If we shift the point source from $x_i$ to another location $x_j$, we find:

$$
\frac{1}{s_i} + v^\top (x - x_j + x_j - x_i) = \frac{1}{s_i} + v^\top (x_j - x_i) + v^\top (x - x_j) = \frac{1}{s_j} + v^\top (x - x_j). \quad (36)
$$

That is, the slowness function $s$ remains unchanged as it is rewritten with respect to a different source.

If $\{x_i\}$ is a set of point sources and $u_i$ is the solution of the eikonal equation for the single point source problem with point source given by $x_i$, then the solution for the multiple point source problem with sources $\{x_i\}$ is:

$$u(x) = \min_i u_i(x). \quad (37)$$
Figure 14: Comparing different ways of selecting factored nodes. For the test problem, $\Omega = [-1,1]^n$, with $n = 2$ (left) and $n = 3$ (right). The domain is discretized into $N^3$ nodes, where $N = 2^p + 1$, so that $h = 2/(N-1)$. The slowness function is constant ($s \equiv 1$). For the 2D problem, olim8rhr is used; olim26rhr is used for the 3D problem. Solutions for the unfactored problem are plotted, along with solutions using a disk/sphere neighborhood with constant factoring radius given by $r^{\circ} = 0.05, 0.1, 0.15, 0.2$. We note that for this problem the choice $r^{\circ} = \sqrt{n}$ results in an exact solution. This only applies to the constant slowness function, $s \equiv 1$. 
Numerical results for section 5.3. Problem sizes are \( N = 2^p + 1 \), where \( p = 3, \ldots, 14 \) in 2D and \( p = 3, \ldots, 9 \) in 3D. The total number of nodes is \( N^n \), where \( n = 2, 3 \). See section 5.3 for least squares fits.
also hold for the multiple point source problem. Addi-

Table 1: Least-squares fits of the runtime and relative $\ell_\infty$ error for OLMs in 2D and 3D. We denote the time for a given $N$ by $T_N$; likewise, $E_N$ denotes the relative $\ell_\infty$ error for a specific $N$. We fit $T_N$ to a power $C_T N^\alpha$. In 2D, we expect $\alpha \approx 2$; in 3D, $\alpha \approx 3$. In 3D, we fit $E_N$ to $C_E h^\beta$, and expect $\beta \approx -1$ in all cases, due to the use of local factoring. In fact, for olim26 and olim3d using either mp0 or mp1, we find that the situation is better than expected, with $\beta \approx -1.3$.

We use this formula to compare relative $\ell_\infty$ errors for each of our OLMs in 2D and olim26 and olim3d in 3D for this slowness function with a pair of point sources, $x_1 = (0,0)$ and $x_2 = (0.8,0)$ in 2D, and $x_1 = (0,0,0)$ and $x_2 = (0.8,0,0)$ in 3D. We set the domain of the problem to be $\Omega = [0,1]^n$ and discretize it into $N = 2^n + 1$ points, so that $h = (N - 1)^{-1}$.

For this choice of slowness function, we plot the CPU runtime versus $N$ (see fig. 16), along with the relative $\ell_\infty$ error versus $N$ (see fig. 15). We also do least squares fits for these plots to get an overall sense of the accuracy and speed (see table 1). Since we have established that olim8, olim26, and olim3d are the best choice of neighborhoods in the previous section, we do not plot these results for olim4, olim6, and olim8.

We can see that our conclusions from section 5.2 also hold for the multiple point source problem. Additionally, our least-squares fits indicate to us that our algorithms’ runtimes are accurately described by the fit $T_N \sim C_T N^\alpha$ with $\alpha \approx n$, and the error by $E_N \sim C_E h^\beta$, with $\beta \approx 1$ (here, $E_N$ is the relative $\ell_\infty$ error). In fact, for olim26 and olim3d with mp0 or mp1, the power $\beta$ is improved beyond 1 to $\beta \approx 1.3$.

### 6 Conclusion

We have presented two families of fast and accurate direct solvers for the eikonal equation. One of these relies on enumerating valid update simplexes, while the other employs a fast search for the first arrival characteristic. These methods use different quadrature rules: a simplified midpoint rule (mp0), a midpoint rule (mp1), and a righthand rule (rhr). We analyze the relationship between these quadrature rules and prove error bounds. We conduct numerical experiments measuring runtime and relative $\ell_\infty$ error. We make a comparison with the standard fast marching method which is equivalent to our olim6_rhr, showing that they are comparable in terms of speed and accuracy, with olim6_rhr being slightly slower. To determine the relative time spent on different tasks, we profile our C++ implementation using Valgrind, separating time spent into several coarse-grained categories. From this, we show that for practical problem sizes, the runtime of Dijkstra-like algorithms behaves like $CN^n$, where $n = 2, 3$, and $N^n$ is the total number of gridpoints (even if this is not strictly true from a computational complexity viewpoint); we also emphasize that memory access patterns play a large role in algorithm runtime, especially for large $N$.

Overall, we conclude that ordered line integral methods are a powerful approach to obtaining a higher degree of accuracy when solving the eikonal equation in 3D. With an appropriate choice of quadrature rule, we are able to exploit improved directional coverage to drive down the error constant. The improved accuracy more than makes up for the modest price paid in speed, and we fully expect it to be possible to find ways to

| Neighborhood | $C_T$ | $C_E$ | $\beta$ |
|--------------|-------|-------|--------|
| olim4        | 7.779 x $10^{-8}$ | $1.0785$ |       |
| olim8        | 1.971 x $10^{-7}$ | $1.0515$ |       |
| olim6        | 2.968 x $10^{-7}$ | $1.085$ |       |
| olim18       | 2.984 x $10^{-6}$ | $1.018$ |       |
| olim26       | 4.649 x $10^{-6}$ | $1.0103$ |       |
| olim3d       | 3.923 x $10^{-6}$ | $1.013$ |       |

(a) $T_N \sim C_T N^\alpha$

| Neighborhood | $C_E$ | $\beta$ |
|--------------|-------|--------|
| olim8_mp0    | 0.4077 | 0.98744 |
| olim8_mp1    | 0.3683 | 0.993 |
| olim8_rhr    | 1.511 | 0.9728 |
| olim26_mp0   | 2.328 | 1.3135 |
| olim26_mp1   | 1.949 | 1.2888 |
| olim26_rhr   | 1.772 | 0.90394 |
| olim3d_mp0   | 2.268 | 1.3141 |
| olim3d_mp1   | 1.865 | 1.2855 |
| olim3d_rhr   | 1.77 | 0.90353 |

(b) $E_N \sim C_E h^\beta$
optimize this family of algorithms further. We have also tried to demonstrate that memory access patterns dominate both update time and time spent maintaining the front data structure, from which we can conclude two things: 1) the exact time spent updating a node is important but not paramount (improving accuracy is more important than improving speed), 2) using memory optimally will lead to a substantial speed-up for large problems.

7 Acknowledgements

We thank Prof. A. Vladimirsky for valuable discussions during the course of this project.

A Minimum actional integral for 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 $\sigma$ and denote $p \equiv \nabla u$, the corresponding Hamiltonian is:

$$H(p, x) = \frac{\|p\|^2}{2} - \frac{s(x)^2}{2} = 0. \tag{38}$$

Since $H = 0$, eq. (38) implies $L = \sup_p (\langle p, x' \rangle - H) = s(x)\|x'\|$. Since $x' = \partial_p H = p$ and $\|p\| = s(x)$ can be expressed as:

$$L(x, x') = \langle p, x' \rangle = \langle x', x' \rangle = \langle \nabla u, x' \rangle = \frac{du}{d\sigma}. \tag{39}$$

Let $x(\sigma)$ be a characteristic arriving at $\hat{x} = x(\hat{\sigma})$ from $x_0 = x(0)$, which lies on the expanding front. Integrating from 0 to $\hat{\sigma}$ and letting $\hat{u} = u(\hat{x})$ and $u_0 = u(x_0)$:

$$\hat{u} - u_0 = \int_0^{\hat{\sigma}} L(x, x')d\sigma = \int_0^{\hat{\sigma}} s(x)\|x'\|d\sigma = \int_0^{L} s(x)dl, \tag{40}$$

where $L$ is the length of the characteristic from $x_0$ to $\hat{x}$ and $dl$ is the length element. A characteristic of eq. (1) minimizes eq. (40) over admissible paths. Then, if $\hat{x}$ is fixed and $\alpha$ is an arc-length parametrized curve with $\alpha(L) = \hat{x}$, eq. (40) is equivalent to:

$$\hat{u} = u(\hat{x}) = \min_{\alpha} \left\{ u(\alpha(0)) + \int_{\alpha} s(x)dl \right\}. \tag{41}$$

Our update procedure is based on eq. (41). This problem may have multiple local minima—$\hat{u}$ above corresponds to the first arrival, which is what interests us primarily in this work. While the standard finite difference method effectively discretizes the Hamiltonian, the family of methods presented here discretizes eq. (41).

B Proofs for section 3.3

Proof of proposition 1. For the gradient, we have:

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

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

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

26
from which the result follows.

\[\text{Proof of proposition 2.}\] Since \(F_1(\lambda; \theta) = u_\lambda + hs_\lambda^2l_\lambda\), for the gradient we have:

\[
\nabla F_1(\lambda; \theta) = \delta U + h \left( \theta l_\lambda \delta s + \frac{s_\lambda^2}{2l_\lambda} \nabla_p^\top \nabla \right) = \delta U + \frac{h}{l_\lambda} \left( \theta p_\lambda^\top p_\lambda \delta s + s_\lambda^2 \delta p^\top p_\lambda \right),
\]

and for the Hessian:

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

Simplifying this gives us the result.

\[\text{Proof of lemma 1.}\] Let \(\nu_\lambda = p_\lambda/l_\lambda \in \mathbb{R}^n\) be the unit vector in the direction of \(p_\lambda\), and assume that \(Q = [\nu_\lambda \ U] \in \mathbb{R}^{n \times n}\) is orthonormal. Then:

\[
\delta p^\top \mathcal{P}_\lambda^\perp \delta p = \delta p^\top (I - \nu_\lambda \nu_\lambda^\top) \delta p = \delta p^\top (QQ^\top - \nu_\lambda \nu_\lambda^\top) \delta p = \delta p^\top UU^\top \delta p. \tag{42}
\]

Hence, \(\delta p^\top \mathcal{P}_\lambda^\perp \delta p\) is a Gram matrix and positive semidefinite.

Next, since \(\Delta^n\) is nondegenerate, the vectors \(p_i\) for \(i = 0, \ldots, n - 1\) are linearly independent. Since the \(i\)th column of \(\delta p\) is \(\delta p_i = p_i - p_0\), we can see that the vector \(p_0\) is not in the range of \(\delta p\); hence, there is no vector \(\mu\) such that \(\delta p_\mu = 0\), for any \(\alpha \neq 0\). What’s more, by definition, \(\ker(\mathcal{P}_\lambda^\perp) = \langle p_\lambda \rangle\). So, we can see that \(\mathcal{P}_\lambda^\perp \delta p_\mu = 0\) only if \(\mu = 0\), from which we can conclude \(\delta p_\perp^\top \mathcal{P}_\lambda^\perp \delta p > 0\). Altogether, bearing in mind that \(s_{\min}\) is assumed to be positive, we conclude that \(\nabla^2 F_1\) is positive definite.

\[\text{Proof of lemma 2.}\] To show that \(\nabla^2 F_1\) is positive definite for \(h\) small enough, note from eq. (14) that \(\nabla F_1\) is the sum of a positive definite matrix and a small, indefinite perturbation. To use this fact, note that since \(\delta p^\top \mathcal{P}_\lambda^\perp \delta p\) is symmetric positive definite, it has an eigenvalue decomposition \(Q^\top A Q\) where \(A_{ii} > 0\) for all \(i\). Since \(\delta p^\top \mathcal{P}_\lambda^\perp \delta p\) doesn’t depend on \(h\), for a fixed set of vectors \(p_0, \ldots, p_n\), we can expect its eigenvalues to be constant with respect to \(h\). So, if we write:

\[
A = \frac{s_\lambda^2 h}{l_\lambda} \delta p^\top \mathcal{P}_\lambda^\perp \delta p = Q \left( \frac{s_\lambda^2 h}{l_\lambda} \Lambda \right) Q^\top \tag{43}
\]

we can expect this matrix’s eigenvalues to be \(\Theta(h)\); in particular, \(\lambda_{\min} \geq Ch\) for some constant \(C\), provided that \(s > s_{\min} > 0\), as assumed. This gives us a bound for the positive definite part of \(\nabla^2 F_1\).

The perturbation \(B = \left\{ \delta p^\top \nu_\lambda, \theta h \delta s \right\}\) is indefinite. Since \(\|\delta s\| = O(h)\), we find that:

\[
|\lambda_{\max}(B)| = \left\| \left\{ \delta p^\top \nu_\lambda, \theta h \delta s \right\} \right\|_2 \leq \theta h \sqrt{n} \left\| \left\{ \delta p^\top \nu_\lambda, \delta s \right\} \right\|_\infty = O(h^2), \tag{44}
\]

where we use the fact that the Lipschitz constant of \(s\) is \(K \leq C\), so that:

\[
|\delta s_i| = |s_i - s_0| \leq K |x_i - x_0| \leq K h \sqrt{n} \leq C h \sqrt{n}, \tag{45}
\]

for each \(i\). Letting \(z \neq 0\), we compute:

\[
z^\top \nabla^2 F_1 z = z^\top A z + z^\top B z \geq \lambda_{\min}(A) z^\top z + z^\top B z \geq Chz^\top z + z^\top B z. \tag{46}
\]

Now, since \(z^\top B z \leq |\lambda_{\max}(B)| z^\top z \leq Dh^2 z^\top z\), where \(D\) is some positive constant, we can see that for \(h\) small enough, it must be the case that \(Ch z^\top z + z^\top B z > 0\); i.e., that \(\nabla^2 F_1\) is positive definite; consequently, \(F_1\) is strictly convex in this case.
C Proofs for section 3.4

In this section, we establish some technical lemmas that we will use to validate the use of $mp0$. Lemmas 4 to 6 set up the conditions for theorem 5 of Stoer and Bulirsch [18], from which theorem 1 readily follows.

**Lemma 4.** There exists $\beta = O(h^{-1})$ s.t. $\|\nabla^2 F_1(\lambda)^{-1}\| \leq \beta$ for all $\lambda \in \Delta^n$.

*Proof of lemma 4.* To simplify eq. (14), we temporarily define:

$$ A = \frac{s^\theta h}{l^\lambda} \delta P^\top \mathcal{P}_{\lambda}^\bot \delta P \quad \text{and} \quad B = \frac{\theta h}{l^\lambda} \left\{ \delta P^\top p_{\lambda}, \delta s \right\}. $$

(47)

Observe that $\|A\| = O(h)$ and $\|B\| = O(h^2)$, since $\|\delta s\| = O(h)$ and since all other factors involved in $A$ and $B$ (excluding $h$ itself) are independent of $h$. Hence:

$$ \|A^{-1} B\| = \frac{\theta}{s^\lambda} \left\| \left( \delta P^\top \mathcal{P}_{\lambda}^\bot \delta P \right)^{-1} \left\{ \delta P^\top p_{\lambda}, \delta s \right\} \right\| = O(h), $$

(48)

since $1/s \leq 1/s_{\text{min}}$ and $\|\delta s\| = O(h)$. Hence, $\|A^{-1}B\| < 1$ for $h$ small enough, and we can Taylor expand:

$$ \nabla^2 F_1(\lambda)^{-1} = (A + B)^{-1} = (I + A^{-1} B)^{-1} A^{-1} $$

$$ = \left( I - A^{-1} B + (A^{-1} B)^2 - \ldots \right) A^{-1} $$

(49)

$$ = A^{-1} - A^{-1} B A^{-1} + (A^{-1} B)^2 A^{-1} - \ldots, $$

which implies $\|\nabla^2 F_1(\lambda)^{-1}\| = O(h^{-1})$. Note that when we Taylor expand, $\|A^{-1}B\| = O(h)$, so that $\|A^{-1}B\| < 1$ for $h$ small enough. To define $\beta$, let:

$$ \beta = \max_{\lambda \in \Delta^n} \|\nabla^2 F_1(\lambda)^{-1}\| = O(h^{-1}), $$

(50)

completing the proof.

**Lemma 5.** There exists $\alpha = O(h)$ s.t. $\|\nabla^2 F_1(\lambda^*_h) - \nabla F_1(\lambda_0^*)\| \leq \alpha$.

*Proof of lemma 5.* From lemma 4 we have $\|F_1(\lambda^*_h)\| = O(h^{-1})$, so to establish the result we only need to show that $\|\nabla F_1(\lambda^*_h)\| = O(h^2)$. To this end, let $\lambda = (n + 1)^{-1} 1_{n \times 1}$ (i.e., the centroid of $\Delta^n$, where $s^\theta$ is evaluated). Then, recalling fig. 3a, $s^\lambda = s^\theta + \delta s^\top (\lambda - \lambda)$ so that, for a general $\lambda$:

$$ \nabla F_1(\lambda) = l^\lambda h \delta s + \delta U + \frac{s^\theta + \delta s^\top (\lambda - \lambda)}{l^\lambda} h \delta P^\top p_{\lambda} $$

$$ = l^\lambda h \delta s + \nabla F_0(\lambda) + \frac{\delta s^\top (\lambda - \lambda)}{l^\lambda} h \delta P^\top p_{\lambda}. $$

(51)

Since $\nabla F_0(\lambda_0^*) = 0$ by optimality, we can conclude using eq. (51) and $\|\delta s\| = O(h)$ that:

$$ \|\nabla F_1(\lambda_0^*)\| = h \left\| l^\lambda h \delta s + \frac{\delta s^\top (\lambda - \lambda)}{l^\lambda} h \delta P^\top p_{\lambda} \right\| = O(h^2), $$

(52)

which proves the result.

**Lemma 6.** The Hessian $\nabla^2 F_1$ is Lipschitz continuous with $O(h)$ Lipschitz constant. That is, there is some constant $\gamma = O(h)$ so that for two points $\lambda$ and $\lambda'$:

$$ \|\nabla^2 F_1(\lambda) - \nabla^2 F_1(\lambda')\| \leq \gamma \|\lambda - \lambda'\|. $$

28
Proof of lemma 6. If we restrict our attention to $\Delta^n$, we see that $l_\lambda^{-1}\delta P^T P_\lambda \delta P$ is Lipschitz continuous function of $\lambda$ with $O(1)$ Lipschitz constant and $\theta\{\delta P^T p_\lambda, \delta s\}/l_\lambda$ is Lipschitz continuous with $O(h)$ Lipschitz constant since $\|\delta s\| = O(h)$. Then, since $s_\lambda^h$ is $O(1)$ Lipschitz, it follows that:

$$A(\lambda) = \frac{s_\lambda^h}{l_\lambda} \delta P^T P_\lambda \delta P$$

has a Lipschitz constant that is $O(h)$ for $\lambda \in \Delta^n$, using the notation of lemma 4. Likewise,

$$B(\lambda) = \frac{\theta h}{l_\lambda} \{\delta P^T p_\lambda, \delta s\} = O(h^2),$$

since it is a sum of two terms involving products of $h$ and $\delta s$. Since $\nabla^2 F_1(\lambda) = A(\lambda) + B(\lambda)$, we can see immediately that it is also Lipschitz on $\Delta^n$ with a constant that is $O(h)$. \hfill \square

Proof of theorem 1. Our proof of theorem 1 relies on the following theorem on the convergence of Newton’s method, which we present for convenience.

**Theorem 5** (Theorem 5.3.2, Stoer and Bulirsch). Let $C \subseteq \mathbb{R}^n$ be an open set, let $C_0$ be a convex set with $C_0 \subseteq C$, and let $f : C \rightarrow \mathbb{R}^n$ be differentiable for $x \in C_0$ and continuous for $x \in C$. For $x_0 \in C_0$, let $r, \alpha, \beta, \gamma$ satisfy $S_\lambda(x_0) = \{x : \|x - x_0\| < \epsilon\} \subseteq C_0$, $\mu = \alpha \beta \gamma < 2$, $r = \alpha (1 - \mu)^{-1}$, and let $f$ satisfy:

(a) for all $x, y \in C_0$, $\|Df(x) - Df(y)\| \leq \gamma \|x - y\|$;

(b) for all $x \in C_0$, $(Df(x))^{-1}$ exists and satisfies $\|Df(x))^{-1}\| \leq \beta$,

(c) and $\|(Df(x))^{-1} f(x_0)\| \leq \alpha$.

Then, beginning at $x_0$, each iterate:

$$x_{k+1} = x_k - Df(x_k)^{-1} f(x_k), \quad k = 0, 1, \ldots, \quad (55)$$

is well-defined and satisfies $\|x_k - x_0\| < \epsilon$ for all $k \geq 0$. Furthermore, $\lim_{k \rightarrow \infty} x_k = \xi$ exists and satisfies $\|\xi - x_0\| \leq \epsilon$ and $f(\xi) = 0$.

For our situation, Theorem 5.3.2 of Stoer and Bulirsch [48] indicates that if:

$$\|\nabla F_1(\lambda)^{-1}\| \leq \beta, \text{ where } \beta = O(h^{-1}), \quad (56)$$

$$\|\nabla F_1(\lambda_0^*)^{-1}\nabla F_1(\lambda_0^*)\| \leq \alpha, \text{ where } \alpha = O(h), \text{ and } \quad (57)$$

$$\|\nabla F_1(\lambda) - \nabla F_1(\lambda')\| \leq \gamma \|\lambda - \lambda'\| \quad \text{for each } \lambda, \lambda' \in \Delta^n, \text{ where } \gamma = O(h), \quad (58)$$

then with $\lambda_0 = \lambda_0^*$, the iteration eq. (15) is well-defined, with each iterate satisfying $\|\lambda_k - \lambda_0\| \leq \epsilon$, where $\epsilon = \alpha/(1 - \alpha \beta \gamma/2)$. Additionally, the limit of this iteration exists, and the iteration converges to it quadratically; we note that since $F_1$ is strictly convex for $h$ small enough, the limit of the iteration must be $\lambda_1^*$, so the theorem also gives us $\|\delta \lambda^*\| = \|\lambda_1^* - \lambda_0^*\| \leq \epsilon$.

Now, we note that eqs. (56) to (58) correspond exactly to lemmas 4 to 6, which gave us values for $\alpha, \beta,$ and $\gamma$. All that remains is to compute $r$. Since the preceding lemmas imply $\alpha \beta \gamma = O(h)$, hence $\alpha \beta \gamma/2 < 1$ for $h$ small enough. We have:

$$r = \frac{\alpha}{1 - \frac{\alpha \beta \gamma}{2}} = \alpha \left(1 + \frac{\alpha^2 \beta^2 \gamma^2}{2} + \frac{\alpha^4 \beta^4 \gamma^4}{4} + \cdots \right) = O(h), \quad (59)$$

so that $\|\delta \lambda^*\| = O(h)$, and the result follows.

To obtain the $O(h^3)$ error bound, from theorem 1, we have $\|\delta \lambda^*\| = O(h)$. Then, Taylor expanding $F_1(\lambda_0^*)$, we get:

$$F_1(\lambda_0^*) = F_1(\lambda_1^* + \delta \lambda^*) = F_1(\lambda_1^*) - \nabla F_1(\lambda_1^*)^T \delta \lambda^* + \frac{1}{2} \delta \lambda^* \nabla^2 F_1(\lambda_1^*) \delta \lambda^* + R,$$
where $|R| = O(\|\delta \lambda^*\|^3)$. Since $\lambda_1^*$ is optimum, $\nabla F_1(\lambda_1^*) = 0$. Hence:

$$|F_1(\lambda_1^*) - F_1(\lambda_0^*)| \leq \frac{1}{2} \|\nabla F_1^2(\lambda_1^*)\| \|\delta \lambda^*\|^2 + O(\|\delta \lambda^*\|^3) = O(h^3),$$

which proves the result.

\[ \square \]

## D Proofs for section 3.5

**Proof of theorem 2.** We proceed by reasoning geometrically; fig. 17 depicts the geometric setup. First, letting $\delta P = QR$ be the reduced QR decomposition of $\delta P$, and writing $\nu_{\lambda^*} = p_{\lambda^*}/l_{\lambda^*}$, we note that since:

$$\nabla F_0(\lambda^*) = \delta U + s^0 h \delta P^T \nu_{\lambda^*} = 0,$$

the optimum $\lambda^*$ satisfies:

$$-R^{-\top} \frac{sU}{s^0 h} = Q^\top \nu_{\lambda^*}.$$

Let $P_{\delta P} = QQ^\top$ denote the orthogonal projector onto range $\delta P$, and $P_{\perp\delta P} = I - QQ^\top$ the projector onto its orthogonal complement. We can try to write $p_{\lambda^*}$ by splitting it into a component that lies in range($\delta P$) and
one that lies in range($\delta P$)$^\perp$. Letting $p_{\text{min}}$ be the point in $p_0 + \text{range}(\delta P)$ with the smallest 2-norm, we write:

$$p_{\lambda^*} = (p_{\lambda^*} - p_{\text{min}}) + p_{\text{min}},$$

(62)

where $p_{\lambda^*} - p_{\text{min}} \in \text{range}(\delta P)$ and $p_{\text{min}} \in \text{range}(\delta P)^\perp$. The vector $p_{\text{min}}$ corresponds to $p_{\lambda_{\text{min}}}$ where $\lambda_{\text{min}}$ satisfies:

$$0 = \delta P^T(\delta P \lambda_{\text{min}} + p_0) = R^T R \lambda_{\text{min}} + R^T Q^T p_0,$$

(63)

hence $\lambda_{\text{min}} = -R^{-1}Q^T p_0$, giving us:

$$p_{\text{min}} = p_0 + \delta P \lambda_{\text{min}} = P_{\delta P}^\perp p_0.$$  

(64)

This vector is easily obtained. For $p_{\lambda^*} - p_{\text{min}}$, we note that $P_{\delta P} \nu_{\lambda^*}$ is proportional to $p_{\lambda^*} - p_{\text{min}}$, suggesting that we determine the ratio $\alpha$ satisfying $p_{\lambda^*} - p_{\text{min}} = \alpha P_{\delta P} \nu_{\lambda^*}$. In particular, from the similarity of the triangles $(\hat{p}, \nu_{\lambda^*}, P_{\delta P} \nu_{\lambda^*})$ and $(\hat{p}, p_{\lambda^*}, p_{\text{min}})$ in fig. 17, we have, using eqs. (61) and (64):

$$\alpha = \frac{\|p_{\text{min}}\|}{\|P_{\delta P} \nu_{\lambda^*}\|} = \sqrt{\frac{p_0^T P_{\delta P}^\perp p_0}{1 - \|Q^T \nu_{\lambda^*}\|^2}} = \sqrt{\frac{p_0^T P_{\delta P}^\perp p_0}{1 - \|R\delta P_{\delta P}^\perp\|^2}}.$$  

(65)

At the same time, since:

$$\nu_{\lambda^*}^T P_{\delta P} \nu_{\lambda^*} = \frac{(P_{\delta P}^\perp p_{\lambda^*})^T (P_{\delta P}^\perp p_{\lambda^*})}{l_{\lambda^*}^2} = \frac{p_{\text{min}}^T p_{\text{min}}}{l_{\lambda^*}^2} = \frac{p_0^T P_{\delta P}^\perp p_0}{l_{\lambda^*}^2},$$

(66)

we can conclude that:

$$l_{\lambda^*} = \alpha = \sqrt{\frac{p_0^T P_{\delta P}^\perp p_0}{1 - \|R\delta P_{\delta P}^\perp\|^2}},$$

(67)

giving us eq. (17), proving the first part of theorem 2.

Next, combining eqs. (61), (62), (64) and (65), we get:

$$p_{\lambda^*} = P_{\delta P}^\perp p_0 - \frac{p_0^T P_{\delta P}^\perp p_0}{1 - \|R\delta P_{\delta P}^\perp\|^2} QR^T \delta U \frac{\delta \theta h}{\delta \theta h}.$$  

(68)

This expression for $p_{\lambda^*}$ can be computed from our problem data and $\delta P$. Now, note that $p_{\lambda^*} = p_0 + \delta P \lambda^*$ implies:

$$\lambda^* = R^{-1}Q^T (p_{\lambda^*} - p_0).$$

(69)

Substituting eq. (68) into eq. (69), we obtain eq. (18) after making appropriate cancellations, establishing the second part of theorem 2.

To establish eq. (19), we note that by optimality of $\lambda^*$, our expression for $\nabla F_0$ (eq. (11) of proposition 1) gives:

$$\delta U = -s^\theta h \frac{\delta P^\top p_{\lambda^*}}{l_{\lambda^*}}.$$  

(70)

This lets us write:

$$\delta U^\top \lambda^* = -s^\theta h p_{\lambda^*}^\top \delta P^\top \lambda^* = s^\theta h p_{\lambda^*}^\top (p_0 - p_{\lambda^*}).$$

(71)

Combining eq. (71) with our definition of $F_0$ yields:

$$\hat{U} = F_0(\lambda^*) = U_0 + \delta U^\top \lambda^* + s^\theta h l_{\lambda^*} = U_0 + s^\theta h \frac{p_{\lambda^*}^\top p_{\lambda^*}}{l_{\lambda^*}} (p_0 - p_{\lambda^*}) + s^\theta h \frac{p_{\lambda^*}^\top p_{\lambda^*}}{l_{\lambda^*}} p_{\lambda^*},$$

(72)

which gives eq. (19), completing the final part of the proof.
E Proofs for section 3.6

Proof of theorem 3. We assume that \( U \) is a linear function in the update simplex; hence, \( \nabla U \) is constant. By stacking and subtracting eq. (20) for different values of \( i \), we obtain, for \( i = 0, \ldots, n - 1 \):

\[
\begin{bmatrix}
\delta U \\
p_i^T
\end{bmatrix} = \begin{bmatrix}
\delta U \\
U_0 - \hat{U}
\end{bmatrix}.
\]  
(73)

The inverse of the matrix in the left-hand side of eq. (73) is:

\[
\begin{bmatrix}
(I - \frac{\nu_{\min} P_i^T}{\nu_{\min} P_i}) QR^{-T}, & \frac{\nu_{\min} P_i^T}{\nu_{\min} P_i}
\end{bmatrix},
\]  
(74)

which can be checked. This gives us:

\[
\nabla U = \left( I - \frac{\nu_{\min} P_i^T}{\nu_{\min} P_i} \right) QR^{-T} \delta U + \frac{U_i - \hat{U}}{\nu_{\min} P_i} \nu_{\min}.
\]  
(75)

Hence, \( \|\nabla U\|^2 \) is a quadratic equation in \( \hat{U} - U_i \). Expanding \( \|\nabla U\|^2 \), a number of cancellations occur since \( Q^T \nu_{\min} = 0 \). We have:

\[
\delta U^T R^{-1} Q^T \left( I - \frac{\nu_{\min} P_i^T}{\nu_{\min} P_i} \right) \left( I - \frac{\nu_{\min} P_i^T}{\nu_{\min} P_i} \right) QR^{-T} \delta U = \|R^{-T} \delta U\|^2 + \left( \frac{p_i^T QR^{-T} \delta U}{\|p_{\min}\|^2} \right)^2,
\]  
(76)

so that, written in standard form:

\[
(\hat{U} - U_i)^T + 2p_i^T QR^{-T} \delta U(\hat{U} - U_i) + \left( \frac{p_i^T QR^{-T} \delta U}{\|p_{\min}\|^2} \right)^2 = 0.
\]  
(77)

Solving for \( \hat{U} - U_i \) gives:

\[
\hat{U} = U_i - p_i^T QR^{-T} \delta U + \|p_{\min}\| \sqrt{(s^o h)^2 - \|R^{-T} \delta U\|^2},
\]  
(78)

establishing eq. (22).

Next, to show that \( \hat{U}' = \hat{U} \), we compute:

\[
\hat{U}' = U_0 + \delta U^T \lambda^* + s^o h \lambda^*
\]  
(8)

\[
= U_0 - \left( Q^T p_0 + l_\lambda R^{-T} \frac{\delta U}{s^o h} \right)^T R^{-T} \delta U + s^o h \lambda^*
\]  
(18)

\[
= U_0 - p_0^T QR^{-T} \delta U + s^o h \lambda^* \left( 1 - \left( \frac{R^{-T} \delta U}{s^o h} \right)^2 \right)
\]

\[
= U_0 - p_0^T QR^{-T} \delta U + \|p_{\min}\| \sqrt{(s^o h)^2 - \|R^{-T} \delta U\|^2} = \hat{U}.
\]  
(17)

To establish eq. (23), first note that \( -R^{-T} \delta U = s^o h Q^T \nu_{\lambda^*} \) by optimality. Substituting this into eq. (22), we first obtain:

\[
\hat{U} = U_i + \frac{s^o h}{l_\lambda} \left( p_i^T \mathcal{P}_{\delta \nu} \nu_{\lambda^*} + \|p_{\min}\| \sqrt{p_{\lambda^*} \mathcal{P}_{\delta \nu} \nu_{\lambda^*}} \right).
\]  
(79)

Now, using the notation for weighted norms and inner products, we have:

\[
p_i^T \mathcal{P}_{\delta \nu} \nu_{\lambda^*} + \|p_{\min}\| \sqrt{p_{\lambda^*} \mathcal{P}_{\delta \nu} \nu_{\lambda^*}} = (p_i, \nu_{\lambda^*})_{\nu_{\lambda^*}} + \|p_i\|_{\mathcal{P}_{\delta \nu}} \|p_{\lambda^*}\|_{\mathcal{P}_{\delta \nu}}.
\]  
(80)
Since $\mathcal{P}_{\tilde{g}_a}$ orthogonally projects onto range($\delta P$)$^\perp$, and since the dimension of this subspace is 1, $\mathcal{P}_{\tilde{g}_a}p_i$ and $\mathcal{P}_{\tilde{g}_a}p_{\lambda^*}$ are multiples of one another and their directions coincide (see fig. 17); furthermore, the angle between them is since our simplex is nondegenerate. So, by Cauchy-Schwarz:

$$\|p_i\|_{\mathcal{P}_{\tilde{g}_a}} \|p_{\lambda^*}\|_{\mathcal{P}_{\tilde{g}_a}} = (p_i, p_{\lambda^*})_{\mathcal{P}_{\tilde{g}_a}}.$$  \hspace{1cm} (81)

Combining eq. (81) with eq. (80) and cancelling terms yields:

$$p_i^\top \mathcal{P}_{\tilde{g}_a}p_{\lambda^*} + \|p_{\min}\| \sqrt{p_{\lambda^*}^\top \mathcal{P}_{\tilde{g}_a}p_{\lambda^*}} = p_i^\top p_{\lambda^*}.$$  \hspace{1cm} (82)

Equation eq. (23) follows.

To parametrize the characteristic found by solving the finite difference problem, first note that the characteristic arriving at $\hat{p}$ is colinear with $\nabla \hat{U}$. If we let $\hat{\nu}$ be the normal pointing from $\hat{p}$ in the direction of the arriving characteristic, let $\hat{p}$ be the point of intersection between $p_0 + \text{range}(\delta P)$ and span($\hat{\nu}$), and let $\hat{l} = \|\hat{p}\|$, then, since $\hat{p} - p_0 \in \text{range}(\delta P)$:

$$\nu_{\min}^\top (\hat{p} - p_0) = 0.$$  \hspace{1cm} (83)

Rearranging this and substituting $\hat{p} = \hat{l} \hat{\nu}$, we get:

$$\hat{l} = \frac{\nu_{\min}^\top p_0}{\nu_{\min}^\top \hat{\nu}}.$$  \hspace{1cm} (84)

Now, if we assume that we can write $\hat{p} = \delta P \hat{\lambda} + p_0$ for some $\hat{\lambda}$, then:

$$\hat{\lambda} = R^{-1} Q^\top (\hat{p} - p_0) = - R^{-1} Q^\top \left(I - \frac{\hat{\nu} \nu_{\min}^\top}{\nu_{\min}^\top \hat{\nu}}\right)p_0.$$  \hspace{1cm} (85)

To see that $\hat{p} = p_{\lambda^*}$, note that since $\hat{\nu} = -\nabla \hat{U}/\|\nabla \hat{U}\| = -\nabla \hat{U}/(s^\theta h)$:

$$\mathcal{P}_{\delta P} \hat{\nu} = \frac{-\mathcal{P}_{\delta P} \nabla \hat{U}}{s^\theta h} = \frac{-QR^{-\top} \delta U}{s^\theta h} = \mathcal{P}_{\delta P} \nu_{\lambda^*}.$$  \hspace{1cm} (86)

Since $\hat{\nu}$ and $\nu_{\lambda^*}$ each lie in the unit sphere on the same side of the hyperplane spanned by $\delta P$, and since $\mathcal{P}_{\delta P}$ orthogonally projects onto range($\delta P$), we can see that in fact $\hat{\nu} = \nu_{\lambda^*}$. Hence, $\hat{p} = p_{\lambda^*} \in p_0 + \text{range}(\delta P)$. The second and third parts of theorem 3 follow.

F Proofs for section 3.7

Proof of theorem 4. For causality of $F_0$, we want $\hat{U} \geq \max_i U_i$, which is equivalent to $\min_i (\hat{U} - U_i) \geq 0$. From eq. (22), we have:

$$\min_i (\hat{U} - U_i) = s^\theta h \min_i \min_{\lambda \in \Delta^\alpha} \frac{\nu_i^\top \nu_{\lambda}}{\|p_i\|} = s^\theta h \min_{i,j} \frac{\nu_i^\top \nu_j}{\|p_i\|} \geq 0.$$  \hspace{1cm} (87)

The last equality follows because minimizing the cosine between two unit vectors is equivalent to maximizing the angle between them; since $\lambda$ is restricted to lie in $\Delta^\alpha$, this clearly happens at a vertex since the minimization problem is a linear program.

For $F_1$, first rewrite $s^\theta_{\lambda^*}$ as follows:

$$s^\theta_{\lambda^*} = s^\theta + \theta (s_0 + \delta s^\top \lambda - \vec{s}),$$  \hspace{1cm} (88)

where $\vec{s} = n^{-1} \sum_{i=0}^{n-1} s_i$. If $\lambda^*_{\theta}$ and $\lambda^*_1$ are the minimizing arguments for $F_0$ and $F_1$, respectively, and if $\delta \lambda^* = \lambda^*_1 - \lambda^*_0$, then we have:

$$F_1^\theta (\lambda^*_1) = F_0^\theta (\lambda^*_0) + \theta \left(s_0 + \delta s^\top \lambda^*_1 - \vec{s}\right) h \lambda^*_1.$$  \hspace{1cm} (89)
By the optimality of \( \lambda_1^* \) and strict convexity of \( F_0^\theta \) (lemma 2), we can Taylor expand and write:

\[
F_0^\theta(\lambda_1^*) = F_0^\theta(\lambda_0^*) + \nabla F_0^\theta(\lambda_0^*)^T \delta \lambda^* + \frac{1}{2} \delta \lambda^*^T \nabla^2 F_0^\theta(\lambda_0^*) \delta \lambda^* + R \geq R, \tag{90}
\]

where \( |R| = O(h^3) \) by theorem 1. Let \( \hat{U} = F_1^\theta(\lambda_1^*) \). Since \( F_0^\theta \) is causal, we can write:

\[
\hat{U} \geq \max_i U_i + R + \theta \left( s_0 + \delta s^T \lambda_1^* - \bar{\lambda} \right) h l_{\lambda_1^*}. \tag{91}
\]

Since \( s \) is Lipschitz, the last term is \( O(h^2) \)—in particular, \( \| \delta s \| = O(h) \) and \( \| s_0 - \bar{\lambda} \| = O(h) \) since \( s_0 \) and \( \bar{\lambda} \) lie in the same simplex. So, because the gap \( \min_i(\hat{U} - U_i) \) is \( O(h^2) \), we can see that \( \hat{U} \geq \max_i U_i \) for \( h \) sufficiently small. \( \square \)

References

[1] https://github.com/sampotter/olim/tree/sisc19. Project page for libolim on GitHub.

[2] https://github.com/sampotter/olim/tree/sisc19/plotting. Link to section of libolim project page containing Python plotting scripts and instructions.

[3] A. Alexandrescu, Modern C++ design: generic programming and design patterns applied, Addison-Wesley, 2001.

[4] J. Arndt, Matters Computational: ideas, algorithms, source code, Springer Science & Business Media, 2010.

[5] D. P. Bertsekas, Network optimization: continuous and discrete models, Citeseer, 1998.

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

[7] A. M. Bronstein, M. M. Bronstein, and R. Kimmel, Numerical geometry of non-rigid shapes, Springer Science & Business Media, 2008.

[8] A. Chacon and A. Vladimirsky, Fast two-scale methods for eikonal equations, SIAM Journal on Scientific Computing, 34 (2012), pp. A547–A578.

[9] A. Chacon and A. Vladimirsky, A parallel two-scale method for eikonal equations, SIAM Journal on Scientific Computing, 37 (2015), pp. A156–A180.

[10] D. L. Chopp, Some improvements of the fast marching method, SIAM Journal on Scientific Computing, 23 (2001), pp. 230–244.

[11] M. G. Crandall and P.-L. Lions, Viscosity solutions of Hamilton-Jacobi equations, Transactions of the American mathematical society, 277 (1983), pp. 1–42.

[12] D. Dahiya and M. Cameron, An ordered line integral method for computing the quasi-potential in the case of variable anisotropic diffusion, arXiv preprint arXiv:1806.05321, (2018).

[13] D. Dahiya and M. Cameron, Ordered line integral methods for computing the quasi-potential, Journal of Scientific Computing, 75 (2018), pp. 1351–1384.

[14] R. B. Dial, Algorithm 360: shortest-path forest with topological ordering, Communications of the ACM, 12 (1969), pp. 632–633.

[15] E. W. Dijkstra, A note on two problems in connexion with graphs, Numerische mathematik, 1 (1959), pp. 269–271.
[16] J.-D. Durou, M. Falcone, and M. Sagona, *Numerical methods for shape-from-shading: A new survey with benchmarks*, Computer Vision and Image Understanding, 109 (2008), pp. 22–43.

[17] B. Engquist and O. Runborg, *Computational high frequency wave propagation*, Acta numerica, 12 (2003), pp. 181–266.

[18] S. Fomel, S. Luo, and H. Zhao, *Fast sweeping method for the factored eikonal equation*, Journal of Computational Physics, 228 (2009), pp. 6440–6455.

[19] S. Fomel and J. A. Sethian, *Fast-phase space computation of multiple arrivals*, Proceedings of the National Academy of Sciences, 99 (2002), pp. 7329–7334.

[20] I. Ihrke, G. Ziegler, A. Tevs, C. Theobalt, M. Magnor, and H.-P. Seidel, *Eikonal rendering: Efficient light transport in refractive objects*, ACM Transactions on Graphics (TOG), 26 (2007), p. 59.

[21] W.-K. Jeong and R. T. Whitaker, *A fast iterative method for eikonal equations*, SIAM Journal on Scientific Computing, 30 (2008), pp. 2512–2534.

[22] C.-Y. Kao, S. Osher, and J. Qian, *Legendre-transform-based fast sweeping methods for static hamilton–jacobi equations on triangulated meshes*, Journal of Computational Physics, 227 (2008), pp. 10209–10225.

[23] S. Kim, *An O(N) level set method for eikonal equations*, SIAM journal on scientific computing, 22 (2001), pp. 2178–2193.

[24] S. Kim, *3-D eikonal solvers: First-arrival traveltimes*, Geophysics, 67 (2002), pp. 1225–1231.

[25] R. Kimmel and J. A. Sethian, *Computing geodesic paths on manifolds*, Proceedings of the national academy of Sciences, 95 (1998), pp. 8431–8435.

[26] R. Kimmel and J. A. Sethian, *Optimal algorithm for shape from shading and path planning*, Journal of Mathematical Imaging and Vision, 14 (2001), pp. 237–244.

[27] S. Luo and J. Qian, *Fast sweeping methods for factored anisotropic eikonal equations: multiplicative and additive factors*, Journal of Scientific Computing, 52 (2012), pp. 360–382.

[28] J.-M. Mirebeau, *Anisotropic fast-marching on cartesian grids using lattice basis reduction*, SIAM Journal on Numerical Analysis, 52 (2014), pp. 1573–1599.

[29] J.-M. Mirebeau, *Efficient fast marching with Finsler metrics*, Numerische mathematik, 126 (2014), pp. 515–557.

[30] J. S. Mitchell, D. M. Mount, and C. H. Papadimitriou, *The discrete geodesic problem*, SIAM Journal on Computing, 16 (1987), pp. 647–668.

[31] N. Nethercote and J. Seward, *Valgrind: a framework for heavyweight dynamic binary instrumenta- tion*, in ACM Sigplan Notices, ACM, 2007, pp. 89–100.

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

[33] S. Osher and R. Fedkiw, *Level set methods and dynamic implicit surfaces*, vol. 153, Springer Science & Business Media, 2006.

[34] A. M. Popovici and J. A. Sethian, *3-d imaging using higher order fast marching traveltimes*, Geophysics, 67 (2002), pp. 604–609.

[35] S. F. Potter. [http://umiacs.umd.edu/~sfp](http://umiacs.umd.edu/~sfp). Author’s personal webpage.
[36] E. Prados and O. Faugeras, *Shape from shading*, in Handbook of mathematical models in computer vision, Springer, 2006, pp. 375–388.

[37] R. Prislan, G. Veble, and D. Svenšek, Ray-trace modeling of acoustic Green’s function based on the semiclassical (eikonal) approximation, The Journal of the Acoustical Society of America, 140 (2016), pp. 2695–2702.

[38] D. Qi and A. Vladimirsky, Corner cases, singularities, and dynamic factoring, arXiv preprint arXiv:1801.04322, (2018).

[39] N. Raghuvanshi and J. Snyder, Parametric wave field coding for precomputed sound propagation, ACM Transactions on Graphics (TOG), 33 (2014), p. 38.

[40] N. Raghuvanshi and J. Snyder, Parametric directional coding for precomputed sound propagation, ACM Transactions on Graphics (TOG), 37 (2018), p. 108.

[41] R. Sedgewick and K. Wayne, *Algorithms*, Addison-Wesley Professional, 2011.

[42] J. A. Sethian, *A fast marching level set method for monotonically advancing fronts*, Proceedings of the National Academy of Sciences, 93 (1996), pp. 1591–1595.

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

[44] J. A. Sethian and A. M. Popovici, 3-D travelt ime computation using the fast marching method, Geophysics, 64 (1999), pp. 516–523.

[45] J. A. Sethian and A. Vladimirsky, Fast methods for the Eikonal and related Hamilton–Jacobi equations on unstructured meshes, Proceedings of the National Academy of Sciences, 97 (2000), pp. 5699–5703.

[46] J. A. Sethian and A. Vladimirsky, Ordered upwind methods for static Hamilton–Jacobi equations: theory and algorithms, SIAM Journal on Numerical Analysis, 41 (2003), pp. 325–363.

[47] M. Slotnick, Lessons in seismic computing, Soc. Expl. Geophys, 268 (1959).

[48] J. Stoer and R. Bulirsch, *Introduction to numerical analysis*, vol. 12, Springer Science & Business Media, 2013.

[49] B. Stroustrup, *The C++ programming language*, Pearson Education, 2013.

[50] Y.-H. R. Tsai, L.-T. Cheng, S. Osher, and H.-K. Zhao, Fast sweeping algorithms for a class of Hamilton–Jacobi equations, SIAM Journal on Numerical Analysis, 41 (2003), pp. 673–694.

[51] J. N. Tsitsiklis, Efficient algorithms for globally optimal trajectories, IEEE Transactions on Automatic Control, 40 (1995), pp. 1528–1538.

[52] J. Van Trier and W. W. Symes, Upwind finite-difference calculation of traveltimes, Geophysics, 56 (1991), pp. 812–821.

[53] J. E. Vidale, Finite-difference calculation of traveltimes in three dimensions, Geophysics, 55 (1990), pp. 521–526.

[54] S. Yang, S. F. Potter, and M. K. Cameron, Computing the quasipotential for nongradient SDEs in 3D, Journal of Computational Physics, 379 (2019), pp. 325–350.

[55] L. Yatziv, A. Bartesaghi, and G. Sapiro, O(N) implementation of the fast marching algorithm, Journal of Computational Physics, 212 (2006), pp. 393–399.
[56] H.-K. Zhao, A fast sweeping method for eikonal equations, Mathematics of computation, 74 (2005), pp. 603–627.