Computing Geodesics on a Riemannian Manifold

David Eberly
Geometric Tools, LLC
http://www.geometrictools.com/
Copyright © 1998-2008. All Rights Reserved.

Created: September 16, 2005
Last Modified: February 14, 2008

Contents

1 Introduction **2**
1.1 Basic Terminology 2
1.2 The Geodesic Equations 2

2 Shooting Methods **5**
2.1 The Standard Algorithm 6
2.2 A Modified Algorithm for Geodesics 7
2.3 Another Modified Algorithm for Geodesics 8

3 Continuous Relaxation by Partial Differential Equations **9**

4 Discrete Relaxation by Dijkstra's Algorithm **11**

5 Hierarchical Relaxation **11**

1 Introduction

This document provides various numerical approaches for computing geodesic curves on an n-dimensional Riemannian manifold. Such curves arise naturally as shortest-length paths between points on a surface; the path itself is required to lie on the surface. The classical surfaces are planes and spheres. The shortest-length path between two points on a plane is the line segment which connects the points. The shortest-length path between two points on a sphere is the shortest great-circle arc which connects the points. In the plane example, the path is unique. In the sphere example, the path is not unique when the points are antipodal; for example, there are infinitely many great-circle arcs connecting the North and South Poles on a sphere, all such arcs having the same length. The shortest-length paths are examples of geodesic curves. Not all geodesic curves need be shortest-length paths. In the sphere example, two nonantipodal points are connected by two great-circle arcs. Both are geodesic curves, one of them the shortest-length path.

1.1 Basic Terminology

The Riemannian manifold has a metric tensor $G = [g_{ij}]$, which is a symmetric $n \times n$ matrix. The differential form associated with the metric is

$$ds^2 = \mathbf{dx}^{\mathrm{T}}\, G\, \mathbf{dx} \tag{1}$$

where $\mathbf{x} = (x^1, \ldots, x^n)$ may be thought of as the parameters which define the manifold. The superscript indices are used to denote contravariant quantities. Subscript indices denote covariant quantities. The variable s is the arc length parameter.

It is assumed that G is positive definite to avoid null geodesics. The inverse of G is $G^{-1} = [g^{ij}]$. The Christoffel symbols of the first kind are

$$\Gamma_{ijk} = (g_{jk,i} + g_{ki,j} - g_{ij,k})/2 \tag{2}$$

where $g_{ij,k} = \partial g_{ij}/\partial x^k$. The Christoffel symbols of the second kind are

$$\Gamma_{ij}^{k} = \Gamma_{ij\ell}\, g^{\ell k} \tag{3}$$

The standard convention is used for repeated indices, in this case a summation is implied for the index ℓ, where $1 \le \ell \le n$.

For a parametrically defined manifold $\mathbf{P}(\mathbf{x}) \in \mathbb{R}^m$ for some dimension $m > n$, the metric tensor is

$$g_{ij} = \frac{\partial \mathbf{P}}{\partial x^i} \cdot \frac{\partial \mathbf{P}}{\partial x^j} \tag{4}$$

and the Christoffel symbols of the first kind are

$$\Gamma_{ijk} = \frac{\partial^2 \mathbf{P}}{\partial x^i \partial x^j} \cdot \frac{\partial \mathbf{P}}{\partial x^k} \tag{5}$$

1.2 The Geodesic Equations

Various methods are employed to derive a set of ordinary differential equations in $\mathbf{x}$ whose solutions are the geodesic curves. These include variational principles (selecting a curve to minimize the length integral), embedding of curves in a parametrically defined manifold (curves are required to have zero curvature), and purely differential geometric methods (absolute differentiation of ordinary arc-length derivatives). The result in all cases is the system of equations

$$\frac{d^2x^k}{ds^2} + \Gamma^k_{ij}\frac{dx^i}{ds}\frac{dx^j}{ds} = 0, \quad 1 \le k \le n \tag{6}$$

Once again, the repeated indices imply summations, in this case summations over i and j.

In the remainder of the document, I ignore the contravariant index convention and write the x indices as subscripts. The superscripts look like powers on x and are somewhat confusing.

Example 1. Consider the (x_1, x_2) plane with metric tensor $G = I$, the 2×2 identity matrix. The Christoffel symbols are all zero, $\Gamma_{ijk} = 0$ and $\Gamma^k_{ij} = 0$. The geodesic equations are $d^2x_1/ds^2 = 0$ and $d^2x_2/ds^2 = 0$. These integrate to produce geodesic curves

$$(x_1, x_2) = (a_1 + b_1 s, a_2 + b_2 s)$$

for constants a_1, b_1, a_2, and b_2. The geodesic curves are straight lines. The shortest-length path connecting two points is a straight line segment. $\bowtie$

Example 2. Consider the unit sphere centered at the origin in three dimensions, a manifold of two dimensions. If $\mathbf{P}$ is a point on the sphere, a parameterization is

$$\mathbf{P}(x_1, x_2) = (\cos(x_1)\sin(x_2), \sin(x_1)\sin(x_2), \cos(x_2))$$

for $x_1 \in [0, 2\pi)$ and $x_2 \in [0, \pi]$. The metric tensor is

$$g_{ij} = \begin{bmatrix} \sin^2(x_2) & 0 \\ 0 & 1 \end{bmatrix}$$

The Christoffel symbols of the first kind are

$$\Gamma_{ij,1} = \begin{bmatrix} 0 & \sin(x_2)\cos(x_2) \\ \sin(x_2)\cos(x_2) & 0 \end{bmatrix}, \quad \Gamma_{ij,2} = \begin{bmatrix} -\sin(x_2)\cos(x_2) & 0 \\ 0 & 0 \end{bmatrix}$$

The Christoffel symbols of the second kind are

$$\Gamma^1_{ij} = \begin{bmatrix} 0 & \cos(x_2)/\sin(x_2) \\ \cos(x_2)/\sin(x_2) & 0 \end{bmatrix}, \quad \Gamma^2_{ij} = \begin{bmatrix} -\sin(x_2)\cos(x_2) & 0 \\ 0 & 0 \end{bmatrix}$$

The geodesic equations are

$$\frac{d^2x_1}{ds^2} + 2\frac{\cos(x_2)}{\sin(x_2)}\frac{dx_1}{ds}\frac{dx_2}{ds} = 0, \quad \frac{d^2x_2}{ds^2} - \sin(x_2)\cos(x_2)\left(\frac{dx_1}{ds}\right)^2 = 0$$

The first equation is trivially satisfied when x_1 is a constant ($dx_1/ds = 0$ and $d^2x/ds^2 = 0$). The second equation becomes $d^2x_2/ds^2 = 0$, in which case a solution is $x_2 = s$ for $s \in [-\pi, \pi]$. The equation $x_1 = c$ for a constant c defines a plane through the origin of space. This plane intersects the unit sphere in a great circle. The value $x_2 = s$ moves you around the great circle with unit speed.

The second equation is trivially satisfied when $x_2 = \pi/2$. The first equation becomes $d^2x_1/ds^2 = 0$, in which case a solution is $x_1 = s$ for $s \in [0, 2\pi]$. The equation $x_2 = \pi/2$ defines a plane through the origin of space. This plane intersects the unit sphere in a great circle. The value $x_1 = s$ moves you around the great circle with unit speed.

Generally, great circles are defined by $\mathbf{P}(s) = \cos(s)\mathbf{U} + \sin(s)\mathbf{V}$, where $\mathbf{U} = (u_1, u_2, u_3)$ and $\mathbf{V} = (v_1, v_2, v_3)$ are unit-length vectors that are perpendicular, and where $s \in [0, 2\pi)$. The parameterization moves you around the great circle with unit speed. The equation

$$(\cos(x_1)\sin(x_2), \sin(x_1)\sin(x_2), \cos(x_2)) = \cos(s)\mathbf{U} + \sin(s)\mathbf{V}$$

has the solutions

$$x_1 = \tan^{-1}\left(\frac{u_2\cos(s) + v_2\sin(s)}{u_1\cos(s) + v_1\sin(s)}\right), \quad x_2 = \cos^{-1}\left(u_3\cos(s) + v_3\sin(s)\right)$$

These may be verified as solutions to the geodesic differential equations. ⋈

Example 3. A slight variation on Example 2 is to consider an ellipsoid $\mathbf{P}^{\mathrm{T}}D\mathbf{P} = 1$, where $D = \mathrm{Diag}(d_1, d_2, d_3)$. The origin is the center of the ellipsoid. The unit sphere centered at the origin is a special case when $D = I$, the identity matrix. A parameterization is

$$\mathbf{P}(x_1, x_2) = (d_1\cos(x_1)\sin(x_2), d_2\sin(x_1)\sin(x_2), d_3\cos(x_2))$$

for $x_1 \in [0, 2\pi)$ and $x_2 \in [0, \pi]$. For simplicity of notation, define $s_i = \sin(x_i)$ and $c_i = \cos(x_i)$ for $i = 1, 2$. The metric tensor is

$$g_{ij} = \begin{bmatrix} [d_1^2 s_1^2 + d_2^2 c_1^2] s_2^2 & (d_2^2 - d_1^2) s_1 c_1 s_2 c_2 \\ (d_2^2 - d_1^2) s_1 c_1 s_2 c_2 & [d_1^2 c_1^2 + d_2^2 s_1^2] c_2^2 + d_3^2 s_2^2 \end{bmatrix}$$

The inverse metric tensor is

$$g^{ij} = \frac{1}{\Delta}\begin{bmatrix} [d_1^2 c_1^2 + d_2^2 s_1^2] c_2^2 + d_3^2 s_2^2 & -(d_2^2 - d_1^2) s_1 c_1 s_2 c_2 \\ -(d_2^2 - d_1^2) s_1 c_1 s_2 c_2 & [d_1^2 s_1^2 + d_2^2 c_1^2] s_2^2 \end{bmatrix}$$

where

$$\Delta = s_2^2\{d_1^2 d_2^2 c_2^2 + d_3^2[d_1^2 s_1^2 + d_2^2 c_1^2] s_2^2\}$$

The Christoffel symbols of the first kind are

$$\Gamma_{ij,1} = \begin{bmatrix} (d_1^2 - d_2^2) s_1 c_1 s_2^2 & [d_1^2 s_1^2 + d_2^2 c_1^2] s_2 c_2 \\ [d_1^2 s_1^2 + d_2^2 c_1^2] s_2 c_2 & (d_1^2 - d_2^2) s_1 c_1 s_2^2 \end{bmatrix}$$

and

$$\Gamma_{ij,2} = \begin{bmatrix} -[d_1^2 c_1^2 + d_2^2 s_1^2] s_2 c_2 & (d_2^2 - d_1^2) s_1 c_1 c_2^2 \\ (d_2^2 - d_1^2) s_1 c_1 c_2^2 & [d_3^2 - d_1^2 c_1^2 - d_2^2 s_1^2] s_2 c_2 \end{bmatrix}$$

The Christoffel symbols of the second kind are

$$\Gamma^1_{ij} = \frac{1}{\Delta}\begin{bmatrix} d_3^2(d_1^2 - d_2^2) s_1 c_1 s_2^4 & s_2 c_2(d_1^2 d_2^2 c_2^2 + d_3^2(d_1^2 s_1^2 + d_2^2 c_1^2) s_2^2) \\ s_2 c_2(d_1^2 d_2^2 c_2^2 + d_3^2(d_1^2 s_1^2 + d_2^2 c_1^2) s_2^2) & d_3^2(d_1^2 - d_2^2) s_1 c_1 s_2^4 \end{bmatrix}$$

and

$$\Gamma^2_{ij} = \frac{1}{\Delta} \begin{bmatrix} -d_1^2 d_2^2 s_2^3 c_2 & 0 \\ 0 & s_2^3 c_2 (d_3^2 (d_1^2 s_1^2 + d_2^2 c_1^2) - d_1^2 d_2^2) \end{bmatrix}$$

The geodesic equations are somewhat complicated expressions. Naturally, this is not a problem when computing numerically, but the point is that even for something as simple as an ellipsoid, the expressions can be daunting. The geodesic equations for an ellipsoid generally do not have a closed-form solution, so you *must* resort to numerical methods to solve them. Practitioners in the geosciences will have the need for this when computing shortest paths on the surface of the earth, which is ellipsoidal rather than spherical. $\bowtie$

The remaining sections discuss various methods for numerically solving the equations (6) to obtain a geodesic path, and hopefully the shortest-length path, between two points on the Riemannian manifold.

2 Shooting Methods

The equations (6) are written concisely in a vector-valued form as

$$\mathbf{x}''(s) = \mathbf{F}(\mathbf{x}(s), \mathbf{x}'(s)) \tag{7}$$

where $\mathbf{x} = (x_1, \ldots, x_n)$, $\mathbf{x}'(s) = d\mathbf{x}/ds$, and $\mathbf{x}'' = d^2\mathbf{x}/ds^2$. Again, the subscripts are used rather than superscripts to avoid confusing them with power operations. The vector-valued function $\mathbf{F}$ is just a short notation for the right-hand side of the geodesic equations.

Given two points $\mathbf{a}$ and $\mathbf{b}$ on the manifold, the shortest-path geodesic connecting them has a length L, which at the moment is unknown. Let's revisit this issue later, but for the sake of the motivation for shooting methods, just treat L as if it is known. The requirement is that a solution $\mathbf{x}(s)$ of equation (7) satisfy the boundary conditions

$$\mathbf{x}(0) = \mathbf{a}, \ \ \mathbf{x}(L) = \mathbf{b} \tag{8}$$

Taken together, equations (7) and (8) are referred to as a *two-point boundary value problem.* Numerical methods for solving such problems are typically more difficult to implement and make robust than for an *initial value problem.* The latter problem involves specifying initial conditions,

$$\mathbf{x}(0) = \mathbf{a}, \ \ \mathbf{x}'(0) = \mathbf{v} \tag{9}$$

The starting position and starting velocity are specified for the differential equation. Initial value problems for systems of first-order ordinary differential equations are easily solved by standard methods such as the Runge-Kutta Fourth-Order method. Equations (7) are second-order equations, but they may be reduced to first-order equations in the usual manner by introducing new variables $\mathbf{y}(s) = \mathbf{x}'(s)$, in which case the first-order system has twice as many equations as the second-order system,

$$\begin{bmatrix} \mathbf{x}'(s) \\ \mathbf{y}'(s) \end{bmatrix} = \begin{bmatrix} \mathbf{y}(s) \\ \mathbf{F}(s, \mathbf{x}(s), \mathbf{y}(s)) \end{bmatrix}, \ \begin{bmatrix} \mathbf{x}(0) \\ \mathbf{y}(0) \end{bmatrix} = \begin{bmatrix} \mathbf{a} \\ \mathbf{v} \end{bmatrix} \tag{10}$$

If you were really lucky, you might guess the initial direction $\mathbf{v}$ for which the solution of the initial value problem (7)-(9) is also the solution for the boundary value problem (7)-(8). The probability of guessing correctly is essentially zero. *Shooting methods* are designed to allow you to start with a presumably incorrect initial guess for $\mathbf{v}$, and then iteratively refine it to the correct velocity. The idea is one of shooting an arrow at a distant target, where the initial elevation and speed of the arrow are key aspects for hitting the target. If you miss the target on the first shot, you adjust the arrow's elevation and/or speed slightly and try again. The adjustments use feedback from the previous attempt to hit the target. If you were close to the target the first time, the adjustment of the elevation and/or speed will be only slight modifications from the previous ones.

2.1 The Standard Algorithm

The mathematical description of a shooting method is now given. Let the solution to the initial value problem be written to indicate its dependence on the initial values as well as on the independent variable s; that is, write the solution as $\mathbf{x}(s, \mathbf{a}, \mathbf{v})$. For each choice of $\mathbf{a}$ and $\mathbf{v}$, you get a different solution to the initial value problem. The initial velocity $\mathbf{v}$ of interest is the one for which $\mathbf{x}(L, \mathbf{a}, \mathbf{v}) = \mathbf{b}$. Consider this condition written as

$$\mathbf{R}(\mathbf{v}) = \mathbf{x}(L, \mathbf{a}, \mathbf{v}) - \mathbf{b} = \mathbf{0} \tag{11}$$

The length L, the initial position $\mathbf{a}$, and the final position $\mathbf{b}$ never change during the numerical construction. The only variable here is the initial velocity $\mathbf{v}$. The equation is set up as a root-finding problem: Construct a root $\mathbf{v}$ to the equation $\mathbf{R}(\mathbf{v}) = \mathbf{0}$.

The function $\mathbf{R}$ is generally not known in closed form, and is obtained only through numerically solving the initial value problem. However, we may still formally seek a root of the equation using Newton's Method (or some other multidimensional root-finding algorithm). Let $D\mathbf{R}$ denote the matrix of first-order partial derivatives of $\mathbf{R} = (R_1, \ldots, R_n)$ with respect to the components of $\mathbf{v} = (v_1, \ldots, v_n)$. That is, the (i, j) entry of the matrix is the partial derivative $\partial R_i / \partial v_j$. Newton's method is

$$\mathbf{v}_{k+1} = \mathbf{v}_k - D\mathbf{R}^{-1}(\mathbf{v}_k)\mathbf{R}(\mathbf{v}_k), \quad k \geq 0 \tag{12}$$

where $\mathbf{v}_0$ is an initial guess to the root. The matrix $D\mathbf{R}^{-1}$ is the inverse of the matrix $D\mathbf{R}$. As k k approaches infinity, the limiting iteration is (hopefully) a root to the equation. In practive, the iterations are computed until either $\mathbf{R}(\mathbf{v}_k)$ is sufficiently close to zero or until consecutive iterates have a difference that is sufficiently close to zero.

The technical challenge is to compute the matrix $D\mathbf{R}$. This is formally written as

$$D\mathbf{R}(\mathbf{v}) = \mathbf{x}_{\mathbf{V}}(s, \mathbf{a}, \mathbf{v}) \tag{13}$$

The subscript notation $\mathbf{x}_{\mathbf{V}}$ is suggestive of taking derivatives of components of $\mathbf{x}$ with respect to components of $\mathbf{v}$. Now consider equation (7) written to show its dependence on other parameters,

$$\mathbf{x}''(s, \mathbf{a}, \mathbf{v}) = \mathbf{F}(s, \mathbf{x}(s, \mathbf{a}, \mathbf{v}), \mathbf{x}'(s, \mathbf{a}, \mathbf{v})) \tag{14}$$

Formally compute the derivative with respect to $\mathbf{v}$ to obtain the following, where the arguments are suppressed for clarity:

$$\mathbf{x}''_{\mathbf{V}} = \mathbf{F}_{\mathbf{X}}\, \mathbf{x}_{\mathbf{V}} + \mathbf{F}_{\mathbf{X}'}\, \mathbf{x}'_{\mathbf{V}} \tag{15}$$

The construction uses the Chain Rule from Calculus. The quantity $\mathbf{F}_{\mathbf{X}}$ is the matrix of first-order partial derivatives of the components of $\mathbf{F}$ with respect to the components of $\mathbf{x}$ and the quantity $\mathbf{F}_{\mathbf{X}'}$ is the matrix of first-order partial derivatives of the components of $\mathbf{F}$ with respect to the components of $\mathbf{x}'$. The matrix $\mathbf{x}_{\mathbf{V}}$ was defined earlier. Its first-order derivative with respect to s is $\mathbf{x}'_{\mathbf{V}}$ and its second-order derivative with respect to s is $\mathbf{x}''_{\mathbf{V}}$.

Equation (15) is referred to as the *first variational equation* for (7). It is a matrix-valued differential equation which may be written as a system of first-order equations in a manner similar to how equation (10) was derived. The initial values may also be formally differentiated to produce

$$\mathbf{x}_{\mathbf{V}}(0) = Z, \quad \mathbf{x}'_{\mathbf{V}}(0) = I \tag{16}$$

where Z is the $n \times n$ zero matrix and I is the $n \times n$ identity matrix.

The initial value problem for $\mathbf{x}_{\mathbf{V}}(s, \mathbf{a}, \mathbf{v})$, namely, (15)-(16), is solved using standard methods, such as the Runge-Kutta Fourth-Order method. The solution is evaluated at $s = L$, and then inverted so that the Newton's iterate in equation (12) may be evaluated; that is,

$$\mathbf{v}_{k+1} = \mathbf{v}_k - [\mathbf{x}_{\mathbf{V}}(L, \mathbf{a}, \mathbf{v}_k)]^{-1}[\mathbf{x}(L, \mathbf{a}, \mathbf{v}_k) - \mathbf{b}] \tag{17}$$

This is quite an intensive algorithm to implement. Moreover, I indicated earlier that we would assume that the geodesic length L is *known*, which it is not!

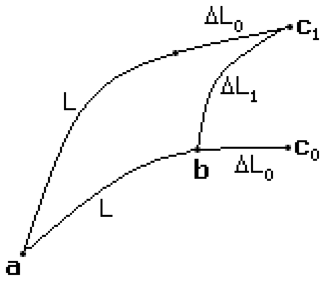
2.2 A Modified Algorithm for Geodesics

As indicated, the standard shooting method requires knowing the final parameter $s = L$ for the boundary point $\mathbf{b}$, but we do not know L since it is the length of the solution we seek. We could attempt to estimate L, shoot once, and then reestimate L, but be aware that if L is *underestimated*, the shooting can never succeed. All the shots must fall short because the estimated L is smaller than the length of the smallest path connecting $\mathbf{a}$ and $\mathbf{b}$.

An *overestimate* of L apparently has similar problems. Too large an estimate means we will always overshoot the target no matter what the initial velocity. The flaw in this thinking is that we want to reach the target *exactly* at the estimated parameter. This goal is overly restrictive. If L' is the overestimate of L, that is $L' > L$, then our hope is that a computed geodesic $\mathbf{x}(s, \mathbf{a}, \mathbf{v})$ of length L' passes through $\mathbf{b}$. Thus, we are satisfied if $\mathbf{x}(s, \mathbf{a}, \mathbf{v}) = \mathbf{b}$ for $s = L < L'$, even though the computed curve terminates at a point different from $\mathbf{b}$.

The standard shooting method must be modified to deal with the new goals. In particular, we will require a velocity vector $\mathbf{v}$ such that the **geodesic** distance between $\mathbf{x}(L', \mathbf{a}, \mathbf{v})$ and $\mathbf{b}$ is minimized. Figure 2.1 illustrates.

Figure 2.1 Overshooting in hopes of finding the boundary point $\mathbf{b}$.



The true distance from $\mathbf{a}$ to $\mathbf{b}$ is L, but is unknown. The overestimated distance is $L' = L + \Delta L_0$. The geodesic path from $\mathbf{a}$ to $\mathbf{c}_0$ is what we want to compute, because it contains $\mathbf{b}$. The numerical algorithm first computes the geodesic from $\mathbf{a}$ to some other point $\mathbf{c}_1$. Any path the numerical method constructs will have length L'. The claim is that the geodesic distance from $\mathbf{b}$ to $\mathbf{c}_1$ must be larger or equal to the geodesic distance from $\mathbf{b}$ to $\mathbf{c}_0$. That is, we claim $\Delta L_1 \geq \Delta_0$. The direct path from $\mathbf{a}$ to $\mathbf{c}_1$ is the shortest-length path connecting the two points. The path from $\mathbf{a}$ to $\mathbf{c}_1$ which goes through $\mathbf{b}$ can only be longer (or same length), which forces $\Delta L_1 \geq \Delta L_0$.

The conclusion here is that the guiding factor in refining the estimate on the initial velocity $\mathbf{v}$ is to minimize the geodesic distance from $\mathbf{b}$ to the boundary point of any numerically constructed curve. However, this problem is itself a two-point boundary value problem because it requires computing geodesic paths from $\mathbf{b}$ to other specified points. An approximation to the situation might be fruitful. The geodesic lengths of the line segments connecting $\mathbf{b}$ to the $\mathbf{c}(\mathbf{v}) = \mathbf{x}(L', \mathbf{a}, \mathbf{v})$ points can be used instead to control the refinement of $\mathbf{v}$. Specifically, the line segment distance is

$$\ell(\mathbf{v}) = \int_0^1 \sqrt{(\mathbf{c}(\mathbf{v}) - \mathbf{b})^{\mathrm{T}} G\left((1-t)\mathbf{b} + t\mathbf{c}(\mathbf{v})\right)(\mathbf{c}(\mathbf{v}) - \mathbf{b})}\, dt \tag{18}$$

This function may be minimized using Steepest Descent or a Conjugate Gradient method. In either case, the gradient vector $\ell_{\mathbf{V}}$ must be computed, and it involves the derivative matrix $\mathbf{c}_{\mathbf{V}}$, which is just the matrix $\mathbf{x}_{\mathbf{V}}$ described earlier in the discussion about the standard shooting method. Thus, the modified shooting algorithm will itself require numerical solution of $\mathbf{x}_{\mathbf{V}}$, just as the standard algorithm did.

One last numerical requirement is producing the overestimate L'. The length of any path connecting $\mathbf{a}$ and $\mathbf{b}$ will do since L is the minimum length over all paths connecting these points. It is sufficient to choose L' to be the geodesic length of the line segment connect $\mathbf{a}$ and $\mathbf{b}$. Equation (18) provides the equation, but with $\mathbf{a}$ replacing $\mathbf{c}(\mathbf{v})$.

2.3 Another Modified Algorithm for Geodesics

The geodesic equations use arc length parameter s for the independent variable. An alternate parameter t may be used. Specifically, think of $x_i(s) = y_i(t)$ for $1 \leq i \leq n$. Differentiating leads to

$$\frac{dx_i}{ds} = \frac{dy_i}{ds} = \frac{dy_i}{dt}\frac{dt}{ds} = \frac{y_i'}{s'} \tag{19}$$

where $s' = ds/dt$ and $y_i' = dy_i/dt$. Differentiating again,

$$\frac{d^2x_i}{ds^2} = \frac{d}{ds}\left(\frac{y_i'}{s'}\right) = \frac{d}{dt}\left(\frac{y_i'}{s'}\right)\frac{dt}{ds} = \frac{s'y_i'' - s''y_i'}{(s')^3} \tag{20}$$

where $s'' = d^2s/dt^2$ and $y_i'' = d^2y_i/dt^2$.

The differential form in equation (1) gives us the relationship between s and t. Written using tensor notation, using the summation convention for repeated indices, and using subscripts instead of superscripts for the y terms,

$$s' = \sqrt{\mathbf{y}'(t)^{\mathrm{T}} G(\mathbf{y}(t))\mathbf{y}(t)} = \sqrt{g_{ij}y_i'y_j'} \tag{21}$$

The derivative is

$$s'' = \frac{g_{ij}(y_i'y_j'' + y_i''y_j') + g_{ij,\ell}y_i'y_j'y_\ell'}{2\sqrt{g_{ij}y_i'y_j'}} \tag{22}$$

An identity that relates the partial derivatives of the metric tensor to the Christoffel symbols is

$$g_{ij,\ell} = \Gamma_{i\ell j} + \Gamma_{j\ell i} \tag{23}$$

The geodesic equations (6) are converted to t and $\mathbf{y}(t)$ using equations (19) through (23), where a multiplication by $(s')^2$ was performed to isolate the first of the second-derivative terms,

$$y_k'' - \left(\frac{g_{ij}(y_i' y_j'' + y_i'' y_j') + (\Gamma_{i\ell j} + \Gamma_{j\ell i}) y_i' y_j' y_\ell'}{2 g_{ij} y_i' y_j'} \right) y_k' + \Gamma^k_{ij} y_i' y_j' = 0. \ \ 1 \le k \le n \tag{24}$$

Notice that the equations must be manipulated to solve explicitly for the second-derivative terms. This step may be performed numerically. We may choose the domain for t as we like, let's say $t \in [0, 1]$. The boundary conditions for the two-point boundary value problem are then $\mathbf{y}(0) = \mathbf{a}$ and $\mathbf{y}(1) = \mathbf{b}$. The standard shooting method may be applied to this problem, but at the expense of many more calculations to evaluate the differential equation terms.

3 Continuous Relaxation by Partial Differential Equations

As we have seen, shooting methods are quite difficult to formulate for the geodesic equations because of the lack of knowledge of the total length L. An alternative approach is to compute the geodesics by relaxation, in particular as the steady-state solution to a system of parabolic partial differential equations corresponding to equations (6):

$$\frac{\partial x_k}{\partial t} = \frac{\partial^2 x_k}{\partial s^2} + \Gamma^k_{ij} \frac{\partial x_i}{\partial s} \frac{\partial x_j}{\partial s} = 0, \ \ 1 \le k \le n \tag{25}$$

Any solution depends on the arc length s and the introduced time parameter t, call it $\mathbf{x}(s, t)$. Parabolic PDEs require initial conditions and boundary conditions. The initial condition for this system is chosen as some curve connecting the points $\mathbf{a}$ and $\mathbf{b}$. That is,

$$\mathbf{x}(s, 0) = \mathbf{c}(s), \ \ 0 \le s \le L \tag{26}$$

The boundary conditions are

$$\mathbf{x}(0, t) = \mathbf{a}, \ \ \mathbf{x}(L, t) = \mathbf{b}, \ \ t \ge 0 \tag{27}$$

Theoretically, one hopes that there is a unique solution to the PDE such that

$$\mathbf{x}(s, \infty) = \lim_{t \to \infty} \mathbf{x}(s, t)$$

is a solution to the original geodesic equations (6).

As formulated, the parabolic PDE has the same problem as the shooting method. We do not know L. The problem needs to be slightly modified to introduce a *moving boundary.* The left boundary remains $s = 0$, but the right boundary is some curve $s = \ell(t)$. The initial curve is $\mathbf{c}(s)$ for $0 \le s \le L' = \ell(0)$ for an overestimate L' of the geodesic length. The line segment connecting the boundary points is as good a guess as any. The boundary conditions are now

$$\mathbf{x}(0, t) = \mathbf{a}, \ \ \mathbf{x}(\ell(t), t) = \mathbf{b}, \ \ t \ge 0 \tag{28}$$

This still begs the question in that it is not clear how one could choose $\ell(t)$, especially since you need $\ell(\infty) = \lim_{t \to \infty} \ell(t) = L$, and L is unknown. Regardless, a numerical scheme based on finite difference approximations to partial derivatives may be used. A polyline is used to approximate $\mathbf{x}(s,t)$, and the vertices of the polyline are updated over time. The process itself creates the moving boundary $\ell(t)$, but it is not important what that boundary is, only that in the limit you get a geodesic curve and the moving boundary approaches L.

The finite difference estimates included a forward difference in time,

$$\frac{\partial x_k(s,t)}{\partial t} \doteq \frac{x_k(s,t+\Delta t) - x_k(s,t)}{\Delta t} \tag{29}$$

and a centered difference in space,

$$\frac{\partial x_k(s,t)}{\partial s} \doteq \frac{x_k(s+\Delta s,t) - x_k(s-\Delta s,t)}{2\Delta s} \tag{30}$$

and

$$\frac{\partial^2 x_k(s,t)}{\partial s^2} \doteq \frac{x_k(s+\Delta s,t) - 2x_k(s,t) + x_i(s-\Delta s,t)}{(\Delta s)^2} \tag{31}$$

Replacing these in equation (25) and solving for the $x_k(s,t+\Delta t)$ term produces

$$\begin{aligned} x_k(s,t+\Delta t) \quad &= \quad x_k(s,t) + \tfrac{\Delta t}{(\Delta s)^2}\left[x_k(s+\Delta s,t) - 2x_k(s,t) + x_k(s-\Delta s,t)\right. \\ &\qquad \left. +0.25\Gamma^k_{ij}\left(x_i(s+\Delta s,t) - x_i(s-\Delta s,t)\right)\left(x_j(s+\Delta s,t) - x_j(s-\Delta s,t)\right)\right] \end{aligned} \tag{32}$$

The initial curve is a line segment, decomposed into a polyline with equally spaced points (Euclidean distance for spacing, not geodesic). The end points are the boundary points and are never modified. An interior polyline point has two adjacent points. The interior point is represented by $\mathbf{x}(s,0)$, the two neighbors by $\mathbf{x}(s \pm \Delta s, 0)$. The interior points are modified and stored in temporary memory (rather than a replacement in current memory) according to the difference equation (32). The point $\mathbf{x}(s,0)$ is updated to $\mathbf{x}(s,\Delta t)$. After all interior points are updated, they are copied back into the original memory locations and the process is repeated for the next time step.

The numerical stability of the algorithm will depend on the size of $\Delta t/(\Delta s)$. The smaller the value, the more likely the algorithm is stable. However, the smaller the value, the longer it takes to converge (if it does at all). This is the usual trade off with numerical methods for parabolic PDEs. Another issue is that Δs is an approximation to the geodesic distance between adjacent polyline points. During the evolution, if adjacent points move far apart, Δs is no longer a good estimate and the finite difference approximations (30) and (31) are no longer accurate. In this case your alternatives are (1) to insert more polyline vertices between the two that have moved far apart or (2) use a different approximation for the derivatives. Choice (1) is preferable, but increases the computational time for the algorithm.

This approach should work reasonably well when the initial curve is a good approximation to the geodesic curve. The line segment I proposed is not always a good choice. Moreover, if you choose a line segment and the polyline representing it has a large number of vertices, the updates of the vertices are rather minor and the method generally fails to converge to a solution. My instinct originally was that I want to start with a large number of vertices, but experiments verified the polyline really is stiff–the vertices cannot move far during the relaxation. An attempt for one vertex to move far from the others increases the total geodesic length of the polyline, which the evolution is designed to inhibit!

4 Discrete Relaxation by Dijkstra's Algorithm

A computationally expensive approach, but one guaranteed to give you good approximations to the geodesic curves, is based on generating a grid of samples. The grid points are the nodes in a weighted graph and pairs of adjacent grid points are the arcs in that graph. Each arc weight is the geodesic distance between the adjacent grid points of the arc. Naturally, this is a circular problem because we are trying to compute the geodesic path (and therefore distance) between points. The idea, though, is to have a high-resolution grid so that the geodesic lengths of line segments connecting adjacent grid points are good approximations to the true geodesic distances. The grid is assumed to have the fullest topology possible. For example, in 2D, the center point of a 3×3 lattice of grid points has 8 adjacent neighbors. In 3D, the center point of a $3 \times 3 \times 3$ lattice has 26 neighbors. In n dimensions, the center point of a 3^n-lattice has $3^n - 1$ neighbors.

Once the weighted graph is established, Dijkstra's Algorithm is used to compute the minimum-weight path connecting the two manifold points of interest. The algorithm is discussed in standard computer science textbooks, for example, "Introduction to Algorithms" by Cormen et al. The usual implementation processes *all pairs* of points. For the problem at hand, this can be quite expensive, especially when the dimension of the Riemannian manifold is large (the grid grows exponentially in size with linear increase in dimension). Instead, the full grid does not have to be physically allocated in memory and the arc weights are computed only when needed. A Bresenham's integer-based line algorithm is used to identify a sequence of grid points that act as a polyline connecting $\mathbf{a}$ and $\mathbf{b}$. This set of grid points is dilated by one unit. In 2D, a grid point is dilated to include its 3×3 neighborhood. In 3D, a grid point is dilated to include its $3 \times 3 \times 3$ neighborhood. In n dimensions, a grid point is dilated to include its 3^n-lattice neighborhood.

The dilated polyline obtained by the method described is treated as a weighted graph and all the arc weights are calculated. Dijkstra's algorithm is applied to compute the minimum-weight path through the graph. If all the path points are interior to the weighted graph, that is, if all $3^n - 1$ neighbors of each path point are in the graph, then this path is the approximation to the geodesic path connecting $\mathbf{a}$ and $\mathbf{b}$. Otherwise, some of the path points are on the boundary of the current set of grid points. The next set of grid points is obtained by dilating the current path by one unit. The update of the grid point set may be done in a manner to preserve weights already computed. A boundary point that is not on the current path and is not an immediate neighbor of a path point may be discarded from the current set of grid points. A boundary point that is on the current path is dilated and the weights computed with the newly added neighbors. Once the next grid point set is known, the process is repeated.

5 Hierarchical Relaxation

This section presents an algorithm I devised based on experimenting with the other methods and understanding their limitations. I have never had much success with shooting methods. The continuous relaxation does not appear to tie in differential geometric information into the updates, especially since the finite difference approximations are analytical and not geometrical. The continuous relaxation does not perform well when you start with a poorly approximating initial curve with even a moderate number of samples. Dijkstra's algorithm is robust, but is very computationally expensive. By design, it appears to need a lot of grid points, giving the convergence behavior a flavor of the continuous relaxation. What appears to be needed is a multiscale approach, one which quickly evolves the initial guess curve to produce a reasonable approximation to the geodesic, and then refines this result at a smaller scale.

The algorithm constructs a polyline approximation to the geodesic curve. Each segment of the polyline is subdivided into two segments. The common point of these segments is selected to reduce the geodesic length of the original segment. Given two points $\mathbf{a}$ and $\mathbf{b}$, the line segment connecting them is $(1-t)\mathbf{a}+t\mathbf{b}$ for $t \in [0,1]$. The geodesic length is

$$L = \int_0^1 \sqrt{(\mathbf{b}-\mathbf{a})^{\mathrm{T}} G\left((1-t)\mathbf{a}+t\mathbf{b}\right)(\mathbf{b}-\mathbf{a})}\, dt \tag{33}$$

The midpoint of the segment is $\mathbf{m} = (\mathbf{a}+\mathbf{b})/2$. Let L_0 denote the geodesic length of the segment connecting $\mathbf{x}_0$ and $\mathbf{m}$ and let L_1 denote the geodesic length of the segment connecting $\mathbf{m}$ and $\mathbf{x}_1$. The total length L is the sum of the lengths L_0 and L_1, which may be written in a form to suggest L is a function of $\mathbf{m}$,

$$\begin{array}{rcl} L(\mathbf{m}) & = & \int_0^1 \sqrt{(\mathbf{m}-\mathbf{a})^{\mathrm{T}} G\left((1-t)\mathbf{a}+t\mathbf{m}\right)(\mathbf{m}-\mathbf{a})}\, dt \\ & & + \int_0^1 \sqrt{(\mathbf{m}-\mathbf{b})^{\mathrm{T}} G\left((1-t)\mathbf{b}+t\mathbf{m}\right)(\mathbf{m}-\mathbf{b})}\, dt \\ & = & \int_0^1 \sqrt{(m_i-a_i)^{\mathrm{T}} g_{ij}\left((1-t)a_\ell+tm_\ell\right)(m_j-a_j)}\, dt \\ & & + \int_0^1 \sqrt{(m_i-b_i)^{\mathrm{T}} g_{ij}\left((1-t)b_\ell+tm_\ell\right)(m_j-a_j)}\, dt \end{array} \tag{34}$$

The goal now is to choose a new value for $\mathbf{m}$ which will minimize $L(\mathbf{m})$, or at least to decrease the initial value of L by a significant amount. Once a new $\mathbf{m}$ is chosen, the original line segment is now a polyline of two line segments. Each subsegment may be further subdivided using the same algorithm. The level of subdivision is at the user's discretion, perhaps automatically terminating when the geodesic length of a segment is smaller than a prescribed tolerance.

There are many methods for minimization. One that seems practical here is just a simple steepest descent. If $\mathbf{m}_0$ is the current choice for $\mathbf{m}$, the next choice is of the form

$$\mathbf{m}_1 = \mathbf{m}_0 - r\nabla L(\mathbf{m}_0) \tag{35}$$

for some $r > 0$. The vector $-\nabla L(\mathbf{m}_0)$ is the direction in which $L(\mathbf{m})$ (instantaneously) decreases the most at the point $\mathbf{m}_0$. Thus, we need to compute the gradient of L. Moreover, we need to do a 1-dimensional search of the ray of equation (35) to compute a minimum for $f(r) = L(\mathbf{m}_0 - r\nabla L(\mathbf{m}_0))$. Or for practical reasons, we just need a search to produce some r which makes L smaller than the value at $\mathbf{m}_0$.

The gradient of L, written using tensor notation, is

$$L_{,k} = \int_0^1 \frac{2g^0_{ij}(m_j-a_j)+tg^0_{ij,k}(m_i-a_i)(m_j-a_j)}{2\sqrt{(m_i-a_i)^{\mathrm{T}} g^0_{ij}(m_i-a_i)}} + \frac{2g^1_{ij}(m_j-b_j)+tg^1_{ij,k}(m_i-b_i)(m_j-b_j)}{2\sqrt{(m_i-b_i)^{\mathrm{T}} g^1_{ij}(m_i-b_i)}}\, dt \tag{36}$$

where

$$\begin{array}{rcl} g^0_{ij} & = & g_{ij}\left((1-t)\mathbf{a}+t\mathbf{m}\right) \\ g^0_{ij,k} & = & g_{ij,k}\left((1-t)\mathbf{a}+t\mathbf{m}\right) \\ g^1_{ij} & = & g_{ij}\left((1-t)\mathbf{b}+t\mathbf{m}\right) \\ g^1_{ij,k} & = & g_{ij,k}\left((1-t)\mathbf{b}+t\mathbf{m}\right) \end{array}$$

The identity (23) is used for evaluating the $g_{ij,k}$ terms.

The scheme which appears to be reasonable numerically is to do all the segment subdivisions first, each subdivision using the steepest descent algorithm once. Once you have a fine enough subdivision, then iterate over the polyline vertices and refine each one using the steepest descent algorithm. The number of iterations over the vertices is at the user's discretion. Also, the steepest descent algorithm may be applied multiple times at a vertex, not just once.