# DIRECT SEARCH ALGORITHMS OVER RIEMANNIAN MANIFOLDS*

David W. Dreisigmeyer†
Los Alamos National Laboratory
Los Alamos, NM 87545

January 8, 2007

**Abstract**

We generalize the Nelder-Mead simplex and LTMADS algorithms and, the frame based methods for function minimization to Riemannian manifolds. Examples are given for functions defined on the special orthogonal Lie group $\mathcal{SO}(n)$ and the Grassmann manifold $\mathcal{G}(n,k)$. Our main examples are applying the generalized LTMADS algorithm to equality constrained optimization problems and, to the Whitney embedding problem for dimensionality reduction of data. A convergence analysis of the frame based method is also given.

**Key words:** Direct search methods, Riemannian manifolds, Grassmann manifolds, equality constraints, geodesics

**AMS subject classifications:** 90C56, 53C21

# 1 Introduction

Direct search methods are function minimization algorithms that attempt to minimize a scalar function $f(\mathbf{x})$, $\mathbf{x} \in \mathbb{R}^n$, using only evaluations of the function $f(\mathbf{x})$ itself. That is, they are implicitly and explicitly derivative-free techniques. Typically these algorithms are easy to code and may be robust to problems where $f(\mathbf{x})$ is discontinuous or noisy [21]. Three popular direct search methods are the Nelder-Mead simplex algorithm [22], the mesh adapted direct search algorithms (MADS) [3] and, the frame based methods [9, 25]. Here we will extend these algorithms to Riemannian manifolds. Two important applications of these extensions are to equality constrained optimization problems and, the dimensionality reduction of data via the Whitney projection algorithm [7, 8].

The Nelder-Mead simplex algorithm is a direct search method for function minimization over $\mathbb{R}^n$. (See [18] for a detailed treatment of the algorithm). At each iteration,

*LA-UR-06-7416
†email: dreisigm@lanl.gov

the algorithm starts with an initial simplex in $\mathbb{R}^n$ with $n+1$ vertices and, returns a new simplex that is (hopefully) in some sense closer to the point $\mathbf{y} \in \mathbb{R}^n$ that (locally) minimizes our function $f(\mathbf{x})$. To accomplish this, one or more of the vertices of the simplex are modified during each iteration. In order to modify the vertices, we need to be able to compute the centroid (mean) of $n$ vertices, connect two points with a straight line, and move along the calculated line a specified distance. All of these operations are trivial in $\mathbb{R}^n$ but, become significantly less so when we want to minimize a function over a Riemannian manifold. The Nelder-Mead algorithm remains popular despite potential shortcomings. For example, convergence results are generally lacking. Indeed, it is possible for the algorithm to converge to non-stationary points even if the function to be minimized is strictly convex [21]. On the positive side, the algorithm is easy to program and, requires no derivative information. We briefly examine the simplex algorithm in section 2.

The MADS algorithms [3] proceed by implicitly constructing a mesh around candidate solutions for our minimization problem. The function $f(\mathbf{x})$ is evaluated at certain points on this mesh. If $f(\mathbf{x})$ is decreasing as we evaluate it at the mesh points, we 'expand' the mesh, allowing ourselves the ability to search further away from our current candidate solution. However, as the MADS algorithms proceed and, we approach a (local) minimum, the mesh is refined. In this way we can zoom in on the value $\mathbf{y} \in \mathbb{R}^n$ that (locally) minimizes $f(\mathbf{x})$. Typically, the MADS algorithm will include a search and a poll step. The (optional) search step will attempt to find a new potential candidate solution to our minimization problem. The user has nearly complete freedom in designing the search step. The poll step will look around our current candidate solution for a better point on our mesh. Unlike the Nelder-Mead algorithm, there are general convergence results for the MADS algorithms [1, 3]. Since the convergence results for the MADS algorithms depend on the poll step, the choices for this step of the algorithms are much more restricted. A particular example of the MADS algorithms, the lower-triangular or LTMADS algorithm, was given in [3]. It is the LTMADS that we will deal with in this paper, examining the standard LTMADS in section 6.

Frame based methods include the MADS algorithms as a special case [3, 9, 25]. They allow more general meshes than the MADS algorithms and, searches off of the mesh points. Of particular interest to us is the fact that the mesh can be freely rotated and scaled during each iteration of an algorithm. The price to be paid for the increased generality of the frame based methods, if one wants to show convergence, is the requirement that a frame based algorithm guarantee that the frame is infinitely refined [9, 25].

Somewhat recently, there has been an increased interest in optimizing functions defined over some Riemannian manifold $\mathcal{M}$ other than $\mathbb{R}^n$ [2, 5, 11, 19, 20, 29]. We will generalize the Nelder-Mead and LTMADS algorithms to Riemannian manifolds. In section 3, we will find that there are three main items we need to address when doing this generalization for the Nelder-Mead algorithm. First, we need to be able to find the straight lines (geodesics) on our manifold $\mathcal{M}$ connecting two points $p, q \in \mathcal{M}$. Secondly, we have to be able to find the Karcher mean on $\mathcal{M}$. This will replace the centroid used in the Nelder-Mead algorithm. Finally, while the simplex is allowed

to grow without bound in $\mathbb{R}^n$, when dealing with a general Riemannian manifold the simplex must be restricted to some well-defined neighborhood of $\mathcal{M}$ at each step. After addressing the difficulties in section 3, we present the final algorithm in section 4. Some examples are examined in section 5. There we will minimize a function over the special orthogonal group $\mathcal{SO}(n)$ and the Grassmann manifold $\mathcal{G}(n,k)$.

Then we will turn our attention to the LTMADS algorithm. The generalization of this algorithm to Riemannian manifolds is dealt with in section 7. We will see that, in comparison to the Nelder-Mead algorithm, the LTMADS algorithm can be extended to Riemannian manifolds in a rather straightforward way, provided the manifold is geodesically complete. In particular, we will only need to be able to find geodesics on our manifold with specified initial conditions and, parallel transport tangent vectors along some calculated geodesics. Section 8 gives a nontrivial example of the use of the generalized LTMADS algorithm. There we apply the LTMADS generalization to the Whitney embedding algorithm [7, 8, 10], which was our original motivation for extending direct search methods to Riemannian manifolds. The Whitney embedding problem is equivalent to minimizing a non-differentiable function over $\mathcal{G}(n,k)$. In section 9, we will show how to use the LTMADS algorithm on Riemannian manifolds to solve constrained optimization problems. The central idea is to treat the zero level set of our constraints as a Riemannain manifold.

The problem with proving convergence of the LTMADS algorithm given in section 7 is that it is more closely related to the frame based methods in [9, 25] than it is to the original LTMADS algorithm in [3]. In section 10 we remove this difficulty by modifying the generalized LTMADS algorithm to a slightly different frame based method. We can then use the results in [9] for our convergence analysis of this new algorithm. An example of the frame based method for a constrained optimization problem is also given in section 10. A discussion follows in section 11.

# 2 The Nelder-Mead Simplex Algorithm

Here we will briefly review the Nelder-Mead algorithm. A more complete discussion is given in [18]. The algorithm for minimizing a function $f(\mathbf{x})$, $\mathbf{x} \in \mathbb{R}^n$, proceeds as follows. First, fix the coefficients for *reflection*, $\rho > 0$, *expansion*, $\epsilon > 1$, *contraction*, $1 > \chi > 0$, and *shrinkage*, $1 > \sigma > 0$. The standard choices are $\rho = 1$, $\epsilon = 2$, $\chi = 1/2$ and $\sigma = 1/2$. Choose the initial simplex vertices $p_1, \ldots, p_{n+1} \in \mathbb{R}^n$ and, let $f_i \doteq f(p_i)$ for $i = 1, \ldots, n+1$. Each iteration of the algorithm is then given by Algorithm 2.1.

Various termination criterion can be used in the Nelder-Mead algorithm. For example, $MATLAB^{\textcircled{R}}$'s `fminsearch` function terminates the algorithm when the size of the simplex and the difference in the function evaluations are less than some specified tolerances. Additional tie-breaking rules for the algorithm are given in [18].

The simplex in the Nelder-Mead algorithm can be visualized as 'oozing' over the

**Algorithm 2.1** ***The Nelder-Mead Simplex Algorithm***

1. *Order the vertices such that* $f_1 \leq \cdots \leq f_{n+1}$.

2. *Try a reflection of* $p_{n+1}$ *through the mean of the other points. Let the reflection point* $p_r$ *be given by*

$$p_r \quad = \quad \bar{p} + \rho(\bar{p} - p_{n+1}),$$

*where* $\bar{p}$ *be given by*

$$\bar{p} \quad = \quad \frac{1}{n}\sum_{j=1}^{n} p_j.$$

*Let* $f_r = f(p_r)$. *If* $f_1 \leq f_r < f_n$, *replace* $p_{n+1}$ *with* $p_r$ *and go to Step 1, else, go to Step 3 or Step 4.*

3. *If* $f_r < f_1$, *try to expand the simplex. Find the expansion point* $p_e$

$$p_e \quad = \quad \bar{p} + \epsilon(p_r - \bar{p}).$$

*Let* $f_e = f(p_e)$. *If* $f_e < f_r$, *replace* $p_{n+1}$ *with* $p_e$ *and go to Step 1, else, replace* $p_{n+1}$ *with* $p_r$ *and go to Step 1.*

4. *If* $f_r \geq f_n$, *try to contract the simplex.*

   (a) *If* $f_n \leq f_r < f_{n+1}$, *do an outside contraction. Let*

$$p_{oc} \quad = \quad \bar{p} + \chi(p_r - \bar{p}).$$

   *Find* $f_{oc} = f(p_{oc})$. *If* $f_{oc} \leq f_r$, *replace* $p_{n+1}$ *with* $p_{oc}$ *and go to Step 1, else, go to Step 5.*

   (b) *If* $f_r \geq f_{n+1}$, *do an inside contraction. Let*

$$p_{ic} \quad = \quad \bar{p} - \chi(\bar{p} - p_{n+1}).$$

   *Find* $f_{ic} = f(p_{ic})$. *If* $f_{ic} < f_{n+1}$, *replace* $p_{n+1}$ *with* $p_{ic}$ *and go to Step 1, else, go to Step 5.*

5. *Shrink the simplex around* $p_1$. *Let*

$$p_i \quad \rightarrow \quad p_1 + \sigma(p_i - p_1),$$

*for* $i = 2, \ldots, n+1$. *Go to Step 1.*

function. This is achieved by reflecting the worst vertex at each iteration and, possibly, expanding the simplex. Once the simplex finds what appears to be a minimum, it will contract itself around the minimum until no further improvement is possible. This is the reason the algorithm is referred to as the 'amoeba algorithm' in [24].

One of the advantages of the simplex algorithm is that it does not use any derivative information. However, because of this, many function evaluations may be needed for the algorithm to converge satisfactorily. It is also possible that the algorithm can converge to a non-stationary point, even for strictly convex functions [21].

# 3 Generalizing the Nelder-Mead Simplex Algorithm to Riemannian Manifolds

Now we consider what needs to be done in order to generalize the Nelder-Mead simplex algorithm to Riemannian manifolds. Let us notice two things about the algorithm. First, all of the 'movements' of the vertices occur along straight lines in $\mathbb{R}^n$. On Riemannian manifolds, the concept of a straight line is replaced by that of a geodesic, which is the shortest possible path connecting two points on a manifold. Secondly, we need to find a centroid to reflect, expand, and contract the simplex through. The Euclidean space mean will be replaced with the Karcher mean on our manifolds. Aside from some subtleties, these are the two main concepts we need in order to generalize the Nelder-Mead algorithm. We will look at each of these in turn.

## 3.1 Geodesics

In the Euclidean space $\mathbb{R}^n$, the shortest path between the points $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{y} \in \mathbb{R}^n$ is the straight line between these two points. On a curved surface such as the sphere $\mathcal{S}^2$ embedded in $\mathbb{R}^3$, we can no longer ask about a straight line between two points $s_1 \in \mathcal{S}^2$ and $s_2 \in \mathcal{S}^2$. However, we can still ask for the shortest possible path on $\mathcal{S}^2$ that connects $s_1$ to $s_2$. This path is called the geodesic between $s_1$ and $s_2$. Note that in this case, if $s_1$ and $s_2$ are antipodal points on $\mathcal{S}^2$, then the geodesic is not unique, since any great circle through $s_1$ and $s_2$ will define two different paths, both of which are the same length. So the concept of a geodesic is typically a local concept on a manifold.

For the reflection, expansion and outside contraction step of the Nelder-Mead algorithm, we not only need to find a geodesic between the two points $\bar{p}$, the Karcher mean defined in subsection 3.2, and $p_{n+1}$ on our Riemannian manifold $\mathcal{M}$, we also need to be able to extend the geodesic beyond $\bar{p}$. Assume we start with the $n+1$ points $p_1, \ldots, p_{n+1} \in U \subset \mathcal{M}$ and, have found the Karcher mean $\bar{p} \in \mathcal{M}$ of the points $p_1, \ldots, p_n$. In order to do the reflection, expansion and contraction steps, we need to find the unique minimal length geodesic $\gamma_{p_{n+1}\bar{p}} : [-2, 1] \to U \subset \mathcal{M}$ between $p_{n+1}$ and $\bar{p}$ such that $\gamma_{p_{n+1}\bar{p}}(1) = p_{n+1}$ and $\gamma_{p_{n+1}\bar{p}}(0) = \bar{p}$. Now, the mapping from a point $\omega \in T_p\mathcal{M}$, the tangent space to $\mathcal{M}$ at point $p$, to a point on the manifold $\mathcal{M}$ is given by

$\mathrm{Exp}_p(\omega)$. Similarly, the mapping from a neighborhood $U \subset \mathcal{M}$ of $p$ to $T_p\mathcal{M}$ is given by $\mathrm{Log}_p(q)$ where $q \in U$ and, we assume the mapping is well-defined. So, we seek an element $\omega \in T_{\bar{p}}\mathcal{M}$ such that the geodesic $\gamma_{p_{n+1}\bar{p}}(\tau) \doteq \mathrm{Exp}_{\bar{p}}(\tau\omega)$ lies within $U$ for $\tau \in [-2, 1]$ and, $\gamma(1) = p_{n+1}$ and $\gamma(0) = \bar{p}$. Then the reflection, expansion, outside contraction and inside contraction points, using the standard choices $\rho = 1$, $\epsilon = 2$, and $\chi = 1/2$, will be given by $\gamma(-1)$, $\gamma(-2)$, $\gamma(-1/2)$, and $\gamma(1/2)$, respectively.

For the shrink step of the Nelder-Mead algorithm, we need to find geodesics $\gamma_{p_i p_1}(\tau_i)$, $i = 2, \ldots, n+1$, that connect the points $p_i$ to the point $p_1$. Further, they need to lie within some subset $V \subset \mathcal{M}$ for $\tau_i \in [0, 1]$ where $\gamma_{p_i p_1}(0) = p_1$ and $\gamma_{p_i p_1}(1) = p_i$, for all $i = 2, \ldots, n+1$. Then, using the standard $\sigma = 1/2$, our new evaluation points are given by $\gamma_{p_i p_1}(1/2)$ for $i = 2, \ldots, n+1$.

Since geodesics are repeatedly calculated, the manifolds one works with should have geodesics that are efficiently calculable. Some typical examples would be Lie groups [20] and, the Stiefel and Grassmann manifolds [11]. Also, as we saw for the case of $\mathcal{S}^2$, geodesics are only uniquely defined when the points are in some restricted neighborhood of the manifold $\mathcal{M}$. For the Nelder-Mead algorithm, we would have $n+1$ points to define our simplex if $\mathcal{M}$ is $n$-dimensional. We require that our simplex not grow so large that we can no longer calculate all of the geodesics required at each iteration of the algorithm. This observation will put restrictions on how large we can allow the simplex to grow in our generalized Nelder-Mead algorithm. We examine this in more detail in subsection 3.3.

## 3.2 The Karcher mean

If we want to generalize the Nelder-Mead algorithm to Riemannian manifolds, we need to replace the $\bar{p}$ in the algorithm with an appropriate centroid defined on the manifold. An example of generalizing the Euclidean centroid concept to Riemannian manifolds is the Karcher mean [16, 17]. The Karcher mean $\bar{p}$ of the points $p_1, \ldots, p_k$ on our manifold $\mathcal{M}$ is defined as

$$\begin{aligned} \bar{p} &\doteq \operatorname*{argmin}_{q \in \mathcal{M}} \frac{1}{2k} \sum_{i=1}^{k} d^2(p_i, q) \\ &\doteq \operatorname*{argmin}_{q \in \mathcal{M}} K(q), \end{aligned} \tag{3.1}$$

where $d(\cdot, \cdot)$ is the distance function on $\mathcal{M}$. For the points $x, y \in \mathcal{M}$, we take $d(x, y)$ to be the length of the geodesic connecting $x$ and $y$.

The Karcher mean is only defined locally on a manifold. This means that the points $p_i \in \mathcal{M}$ in (3.1) must lie within some restricted neighborhood of $\mathcal{M}$ in order for the Karcher mean to exist and be unique. For instance, if $s_1$ is the north pole and $s_2$ is the south pole on $\mathcal{S}^2$, then the entire equator qualifies as a Karcher mean. Similar to the geodesic case, this will place restrictions on how large our simplex can grow.

An algorithm for calculating the Karcher mean is presented in [20, 32]. Let us

illustrate the algorithm when our manifold is the Lie group $\mathcal{SO}(3)$, where a point $p \in \mathcal{SO}(3)$ is a 3-by-3 matrix satisfying the criteria $p^T p = I$ and $\det(p) = 1$. Let $T_p\mathcal{SO}(3)$ be the tangent space to $\mathcal{SO}(3)$ at the point $p$. Then, $\bar{p}$ is the (local) Karcher mean if [16]

$$\mathbf{0} \quad = \quad -\frac{1}{k}\sum_{i=1}^{k} \mathrm{Log}_{\bar{p}}(p_i), \tag{3.2}$$

where $\mathbf{0}$ is the origin of $T_{\bar{p}}\mathcal{SO}(3)$. In fact, the right-hand side of (3.2) is the gradient of $K(q)$ in (3.1) when $\mathrm{Log}_{\bar{p}}$ is replaced by $\mathrm{Log}_q$. This suggests a gradient descent algorithm, which is the method used in [20, 32].

On $\mathcal{SO}(3)$, let us use the Frobenius norm on $T_p\mathcal{SO}(3)$ to induce the Riemmanian metric on $\mathcal{SO}(3)$. Then $\mathrm{Exp}_p(\omega) = p\exp(p^T\omega)$ and, $\mathrm{Log}_p(q) = p\log(p^T q)$, where $q, p \in \mathcal{SO}(3)$, $\omega \in T_p\mathcal{SO}(3)$ and, exp and log have their usual matrix interpretation. Then, the method for finding the Karcher mean is given by Algorithm 3.1. This algorithm will also work for more general connected, compact Lie groups.

**Algorithm 3.1** ***The Karcher Mean Algorithm for*** $\mathcal{SO}(3)$ *[20, 32]*
*Assume we have the points* $p_1, \ldots, p_n \in \mathcal{SO}(3)$. *Fix a tolerance* $\delta > 0$ *and set* $q = p_1$. *Iterate the following:*

1. *Find*

$$\omega \quad = \quad \frac{1}{n}\sum_{i=1}^{n} \log\left(q^T p_i\right).$$

*If* $\|\omega\| < \delta$, *return* $\bar{p} = q$ *and stop, else, go to Step 2.*

2. *Let*

$$q \quad \rightarrow \quad q\exp(\omega)$$

*and go to Step 1.*

In the above example we made heavy use of the Exp and Log mappings. So, in order to efficiently calculate the Karcher mean, we would need manifolds that have tractable Exp and Log mappings. Examples of such manifolds are given in [27]. These include the general linear group $\mathcal{GL}(n)$ of invertible $n$-by-$n$ matrices and it's subgroups, e.g., $\mathcal{SO}(n)$, as well as quotients of $\mathcal{GL}(n)$, e.g., the Grassmann manifold $\mathcal{G}(n,k)$ of $k$-planes in $\mathbb{R}^n$. We also note that a different method for calculating a Karcher-like mean for Grassmann manifolds is presented in [2].

### 3.3 Restricting the simplex growth

As alluded to in the previous subsections, if we let our simplex grow without bound, it may no longer be possible to compute the required geodesics or the Karcher mean. For the example of $\mathcal{SO}(3)$ in subsection 3.2, we have four restrictions on the neighborhood for the Karcher mean algorithm [20]. For simplicity, assume that we are trying to calculate the Karcher mean in some neighborhood $U \subset \mathcal{SO}(3)$ around the identity element $e \in \mathcal{SO}(3)$. We will take $U$ to be a open ball of radius $r$ around $e$, i.e., $U = B_r(e)$. Then, the tangent space $T_e\mathcal{SO}(3)$ is given by the Lie algebra $\mathfrak{so}(3)$ of 3-by-3 skew-Hermitian matrices. The four restrictions on $U$ are:

1. The $\log$ function needs to be well-defined. Let $\mathfrak{B}_\rho(0)$ be the ball of radius $\rho$ around the identity element $\mathbf{0} \in \mathfrak{so}(3)$, where $\rho$ is the largest radius such that $\exp$ is a diffeomorphism, i.e., the *injectivity radius*. Then we need $U \subset \exp(\mathfrak{B}_\rho(0))$. Since $\exp(\mathfrak{B}_\rho(\mathbf{0})) = B_\rho(e)$, we can take $U = B_\rho(e)$ for this requirement to be satisfied.

2. We call a set $V \subset \mathcal{SO}(3)$ *strongly convex* [29] if, for any $X, Y \in V$, the geodesic $\gamma_{XY}(\tau) \subset V$ connecting $X$ and $Y$ is the unique geodesic of minimal length between $X$ and $Y$ in $\mathcal{SO}(3)$. We require $U$ to be a strongly convex set.

3. Let $f : V \subset \mathcal{SO}(3) \to \mathbb{R}$. We call $f$ *convex* [29] if, for any geodesic $\gamma(\tau) : [0,1] \to V$,

$$(f \circ \gamma)(\tau) \quad \leq \quad (1-t)(f \circ \gamma)(\tau) + t(f \circ \gamma)(\tau), \tag{3.3}$$

   for $t \in [0,1]$. If the inequality is strict in (3.3), we call $f$ *strictly convex*. The function $d(e,p)$ must be convex on $U$, where $p \in U$. The largest $r$ such that $U = B_r(e)$ satisfies 2 and 3 is called the *convexity radius*.

4. The function $K(q)$ in (3.1) is strictly convex on $U$, for $q \in U$.

The above restrictions also hold for a general Lie group $\mathcal{G}$. As shown in [20], as long as there is a $q \in \mathcal{G}$ such that the points $p_1, \ldots, p_n \in \mathcal{G}$ all lie in the open ball $B_{\pi/2}(q)$, then we will satisfy the requirements. Similar computationally tractable restrictions would need to be derived for any Riemannian manifold to guarantee that the Karcher mean can be found.

As stated in subsection 3.1, we need to be able to extend the geodesic from $p_{n+1}$ through $\bar{p}$ in order to do the reflection, expansion and outside contraction steps. It may be possible to extend the geodesics by changing to a different neighborhood that includes the other points of the manifold as well as $p_r$, $p_e$ and $p_{oc}$. This is possible for, e.g., Grassmann manifolds [2]. Here one needs to be careful that the restrictions for computing the Karcher mean are still met in the new neighborhood. Alternately, one could allow the $\rho$, $\epsilon$ and $\chi$ parameters to vary as the algorithm is run, so that we never have to leave the current neighborhood during the current iteration. Another possibility is to have the function we are trying to minimize return an infinite value if we leave

the current neighborhood. This is similar to enforcing box constraints when using the Nelder-Mead algorithm in $\mathbb{R}^n$. Note that the inside contraction and the shrink steps of the algorithm will not cause any problems since the midpoint of the geodesic is already contained in our neighborhood.

# 4 The Nelder-Mead simplex algorithm on Riemannian manifolds

Now we have all of the pieces needed in order to extend the Nelder-Mead algorithm to a Riemannian manifold $\mathcal{M}$. For simplicity, assume we have appropriate methods in place to restrict the growth of the simplex at each iteration and, fix the coefficients for *reflection*, $\rho = 1$, *expansion*, $\epsilon = 2$, *contraction*, $\chi = 1/2$, and *shrinkage*, $\sigma = 1/2$. Choose the initial simplex vertices $p_1, \ldots, p_{n+1} \in \mathcal{M}$. In the following we let $f_i \doteq f(p_i)$ for $i = 1, \ldots, n+1$. If the requirements of subsections 3.1 and 3.2 are met, i.e., the geodesics are well-defined and we can calculate the Karcher mean, each iteration of the generalized Nelder-Mead simplex algorithm on the Riemannian manifold $\mathcal{M}$ is then given by Algorithm 4.1.

The tie breaking rules and termination criterion alluded to in section 2 can be applied to our generalized Nelder-Mead algorithm. Here, the size of the simplex would need to be intrinsically defined on the manifold $\mathcal{M}$. The above algorithm would also need to be modified depending on the method used to restrict the growth of the simplex. For example, if $\gamma_{p_{n+1}\bar{p}}(-2)$ is not well-defined or, we could not calculate the Karcher mean in the next iteration, we could return an infinite value for the function evaluation, which amounts to skipping the expansion step. Alternately, we could adjust the values of $\rho$, $\epsilon$ and $\chi$, in this iteration, so that the geodesic is well-defined and, the Karcher mean can be calculated in the next step.

# 5 Examples of the generalized Nelder-Mead algorithm

Now we will look at two examples of the generalized Nelder-Mead algorithm in some detail. The first will be minimizing a function over the special orthogonal group $\mathcal{SO}(n)$. The second will look at the algorithm when our function to be minimized is defined over the Grassman manifold $\mathcal{G}(n, k)$. We will review both of these manifolds below, paying particular attention to the problem of calculating geodesics and the Karcher mean efficiently.

## 5.1 $\mathcal{SO}(n)$

Let us collect all of the information we will need in order to perform the Nelder-Mead simplex algorithm over the special orthogonal group $\mathcal{SO}(n)$, an $n(n-1)/2$

**Algorithm 4.1** ***The Nelder-Mead Simplex Algorithm for Riemannian Manifolds***

1. *Order the vertices such that* $f_1 \leq \cdots \leq f_{n+1}$.

2. *Find the Karcher mean* $\bar{p}$ *of the points* $p_1, \ldots, p_n$. *Let* $U \subset \mathcal{M}$ *be a neighborhood of* $\bar{p}$ *satisfying all of the convexity conditions of subsection 3.2. Find the geodesic* $\gamma_{p_{n+1}\bar{p}} : [-2, 1] \to U$ *as in subsection 3.1. Try a reflection of* $p_{n+1}$ *through* $\bar{p}$ *by letting the reflection point be given by* $p_r = \gamma_{p_{n+1}\bar{p}}(-1)$. *Let* $f_r = f(p_r)$. *If* $f_1 \leq f_r < f_n$, *replace* $p_{n+1}$ *with* $p_r$ *and go to Step 1, else, go to Step 3 or Step 4.*

3. *If* $f_r < f_1$, *try to expand the simplex. Find the expansion point* $p_e = \gamma_{p_{n+1}\bar{p}}(-2)$. *Let* $f_e = f(p_e)$. *If* $f_e < f_r$, *replace* $p_{n+1}$ *with* $p_e$ *and go to Step 1, else, replace* $p_{n+1}$ *with* $p_r$ *and go to Step 1.*

4. *If* $f_r \geq f_n$, *try to contract the simplex.*

   (a) *If* $f_n \leq f_r < f_{n+1}$, *do an outside contraction. Let* $p_{oc} = \gamma_{p_{n+1}\bar{p}}(-1/2)$. *Find* $f_{oc} = f(p_{oc})$. *If* $f_{oc} \leq f_r$, *replace* $p_{n+1}$ *with* $p_{oc}$ *and go to Step 1, else, go to Step 5.*

   (b) *If* $f_r \geq f_{n+1}$, *do an inside contraction. Let* $p_{ic} = \gamma_{p_{n+1}\bar{p}}(1/2)$. *Find* $f_{ic} = f(p_{ic})$. *If* $f_{ic} < f_{n+1}$, *replace* $p_{n+1}$ *with* $p_{ic}$ *and go to Step 1, else, go to Step 5.*

5. *Shrink the simplex around* $p_1$. *Let* $V \subset \mathcal{M}$ *be a neighborhood of* $p_1$ *satisfying all of the convexity conditions of subsection 3.2. Find the geodesics* $\gamma_{p_i p_1} : [0, 1] \to V$, $i = 2, \ldots, n + 1$, *as in Section 3.1. Let*

$$p_i \quad \to \quad \gamma_{p_i p_1}(1/2),$$

   *for* $i = 2, \ldots, n + 1$. *Go to Step 1.*

diimensional manifold [20, 27]. A point $p \in \mathcal{SO}(n)$ satisfies the conditions $p^T p = I$ and $\det(p) = 1$. An element $\omega \in T_p\mathcal{SO}(n)$ is a skew-Hermitian matrix. Also, the Riemannian metric on $\mathcal{SO}(n)$ is induced by the Frobenius norm on the tangent space. So, for $\omega_1, \omega_2 \in T_p\mathcal{SO}(n)$, the metric $h(\omega_1, \omega_2)$ is given by

$$h(\omega_1, \omega_2) \quad = \quad \mathrm{Tr}\left(\omega_1^T \omega_2\right). \tag{5.1}$$

The Exp and Log maps are

$$\mathrm{Exp}_p(\omega) \quad = \quad p \exp(\omega) \text{ and} \tag{5.2}$$

$$\mathrm{Log}_p(q) \quad = \quad \log(p^T q), \tag{5.3}$$

where $p, q \in \mathcal{SO}(n)$, $\omega \in T_p\mathcal{SO}(n)$, exp and log have their standard matrix interpretation and, we assume $\mathrm{Log}_p(q)$ is well-defined.

We have already seen in subsection 3.3 that we need $p_1, \ldots, p_m \in B_{\pi/2}(q)$ for some $q \in \mathcal{SO}(n)$ in order for the Karcher mean algorithm (Algorithm 3.1 in subsection 3.2) to

be guaranteed to converge. The last remaining pieces of information are the formulas for the geodesic $\gamma_{qp}(\tau)$ from $p \in \mathcal{SO}(n)$ to $q \in \mathcal{SO}(n)$ and, the distance $d^2(p,q)$ between $p$ and $q$. These are given by

$$\gamma_{qp}(\tau) = p \exp\left[\tau \log(p^T q)\right] \text{ and} \tag{5.4}$$

$$d^2(p,q) = \frac{1}{2}\mathrm{Tr}\left(\left[\log(p^T q)\right]^2\right). \tag{5.5}$$

We tested the Nelder-Mead algorithm on an eigenvalue decomposition problem. Given the symmetric, positive definite matrix

$$X = \begin{bmatrix} 5 & 2 & 1 \\ 2 & 7 & 3 \\ 1 & 3 & 10 \end{bmatrix}, \tag{5.6}$$

we wanted to find the point $p \in \mathcal{SO}(3)$ such that the sum of the squares of the off-diagonal elements of $pXp^T$, denoted by $OD(X,p)$, was minimized. We randomly selected a point $g_0 \in \mathcal{SO}(3)$ as an initial guess for the solution. This formed one of the vertices of our initial simplex. The other vertices were randomly chosen such that the simplex was non-degenerate and, satisfied the size restrictions. We did this for $1000$ different initial guesses. If the simplex was not improving the function evaluation after $100$ consecutive iterations, we restarted with a new initial simplex around the same $g_0$. The reason for this is that the convergence of the Nelder-Mead algorithm, in this example, was very dependent on the choice for the initial simplex. We restarted $17\%$ of our runs using this technique. Of course here we had the advantage of knowing the (unique) minimal value of our function, namely $\min_p OD(X,p) = 0$. We'll comment on this more in section 11. Restricting the size of the simplex was accomplished by choosing our neighborhood to be $B_{\pi/4}(\bar{p})$ and allowing the $\rho$, $\epsilon$ and $\chi$ parameters to vary in each iteration of the algorithm. Note that we used $r = \pi/4$, not $r = \pi/2$, for better numerical results. The maximum allowed value of $\epsilon$ at each iteration is found via the formula

$$\epsilon_{max} = \frac{\pi}{4}\left(-\frac{1}{2}\mathrm{Tr}\left[\left(\log(\bar{p}^T p_{n+1})\right)^2\right]\right)^{-1/2}. \tag{5.7}$$

If $\epsilon_{max} < 2$, we will need to scale the $\rho$, $\epsilon$ and $\chi$ parameters during the current iteration. The results are shown in Table 1, where $g_f$ is the solution returned by the Nelder-Mead algorithm and `cputime` is the $MATLAB^{\textcircled{R}}$ command for measuring the CPU time.

Table 1: Eigenvalue Decomposition Problem

| Average $OD(X, g_0)$ | Average $OD(X, g_f)$ | Average `cputime` |
|---|---|---|
| 12.1817 | 2.5729 e(-16) | 17.7475 secs |

## 5.2 $\mathcal{G}(n,k)$

For the Grassmann manifold $\mathcal{G}(n,k)$ of $k$-planes in $\mathbb{R}^n$, we need to find easily calculable formulas for the geodesic between the points $p, q \in \mathcal{G}(n,k)$ and, the Karcher mean of the points $p_1, \ldots, p_m \in \mathcal{G}(n,k)$ [2, 11]. A point $p \in \mathcal{G}(n,k)$, $p^T p = I$, actually represents a whole equivalence class of matrices $[p]$ that span the same $k$-dimensional subspace of $\mathbb{R}^n$. Letting $o_k \in \mathcal{O}(k)$, where $\mathcal{O}(k)$ is the $k$-by-$k$ orthogonal matrix group, we have that

$$[p] \quad = \quad \{p\, o_k \,|o_k \in \mathcal{O}(k)\}\,. \tag{5.8}$$

(Note that we will treat $p$ and $q$ as being points on $\mathcal{G}(n,k)$ as well as being $n$-by-$k$ matrix representatives of the equivalence class of $k$-planes in $\mathbb{R}^n$.) The tangent space to $p \in \mathcal{G}(n,k)$ is given by

$$T_p\mathcal{G}(n,k) \quad = \quad \left\{ \omega \,\middle|\, \omega = p_\perp g \text{ and } g \in \mathbb{R}^{(n-k)\times k} \right\}, \tag{5.9}$$

where $p_\perp$ is the orthogonal complement to $p$, i.e., $p_\perp = \mathrm{null}(p^T)$. The dimension of the tangent space, and, hence, of $\mathcal{G}(n,k)$, is $k(n-k)$. The Riemannian metric on $\mathcal{G}(n,k)$ is induced by the Frobenius norm on the tangent space. That is, for $\omega_1, \omega_2 \in T_p\mathcal{G}(n,k)$, the metric $h(\omega_1, \omega_2)$ is given by

$$\begin{aligned} h(\omega_1, \omega_2) \quad &= \quad \mathrm{Tr}(\omega_1^T \omega_2) \\ &= \quad \mathrm{Tr}(g_1^T g_2). \end{aligned} \tag{5.10}$$

This metric is the unique one (up to a constant multiple) that is invariant under the action of $\mathcal{O}(n)$ on $\mathbb{R}^n$, i.e., rotations and reflections of $\mathbb{R}^n$. The $\mathrm{Exp}_p$ map is given by

$$\mathrm{Exp}_p(\omega) \quad = \quad pV\cos(\Theta) + U\sin(\Theta), \tag{5.11}$$

where $\omega \in T_p\mathcal{G}(n,k)$ has the SVD $\omega = U\Theta V^T$. Also, the $\mathrm{Log}_p$ map is given by

$$\mathrm{Log}_p(q) \quad = \quad U\Theta V^T, \tag{5.12}$$

where $p_\perp p_\perp^T q(p^T q)^{-1} = U\Sigma V^T$ and $\Theta = \arctan(\Sigma)$, when it is well-defined.

Now we can find the geodesic formula between the points $p, q \in \mathcal{G}(n,k)$. We will require that $p$ and $q$ are close enough together that $p^T q$ is invertible. First, we find the SVD of

$$p_\perp p_\perp^T q(p^T q)^{-1} \quad = \quad U\Sigma V^T \tag{5.13}$$

and, let $\Theta = \arctan(\Sigma)$. Then the geodesic from $p$ to $q$ is given by

$$\gamma_{qp}(\tau) \quad = \quad [pV\ U] \begin{bmatrix} \cos(\Theta\tau) \\ \sin(\Theta\tau) \end{bmatrix}, \tag{5.14}$$

where $\gamma_{qp}(0) = p$ and $\gamma_{qp}(1) = q$. The distance between $p$ and $q$ induced by (5.10) is given by

$$d^2(p,q) \quad = \quad \sum_{i=1}^{k} \theta_i^2, \tag{5.15}$$

where the $\theta_i$ are the diagonal elements of $\Theta$. Since the $\theta_i$ are the principle angles between $p$ and $q$ and, $pV$ and $(pV\cos(\Theta) + U\sin(\Theta))$ are the associated principle vectors, we can use Algorithm 12.4.3 of [12] to find the require quantities in (5.14) and (5.15). If we have the SVD

$$p^T q = V\cos(\Theta)Z^T, \tag{5.16}$$

then $pV$ and $qZ$ give us the principle vectors and, $U\sin(\Theta) = qZ - pV\cos(\Theta)$. In practice, we found Algorithm 5.1 for the Log map to be much more stable numerically.

**Algorithm 5.1** ***The*** $\mathrm{Log}_p(q)$ ***Map for Grassmann Manifolds***
*Given points $p, q \in \mathcal{G}(n,k)$, the following returns $\mathrm{Log}_p(q)$:*

1. *Find the CS decomposition $p^T q = VCZ^T$ and $p_\perp^T q = WSZ^T$, where $V$, $W$ and $Z$ are orthogonal matrices and, $C$ and $S$ are diagonal matrices such that $C^TC + S^TS = I$ [12]. Note that $C$ will always be a square, invertible matrix.*
2. *Delete (add) zero rows from (to) $S$ so that it is square. Delete the corresponding columns of $W$ or, add zero columns to $W$, so that it has a compatible size with $S$.*
3. *Let $U = p_\perp W$ and $\Theta = \arctan(SC^{-1})$.*

*Then $U$, $\Theta$ and $V$ are as in (5.12).*

Given the points $p_1, \ldots, p_m \in \mathcal{G}(n,k)$, the Karcher mean is given by

$$\begin{aligned} \bar{p} \quad &= \quad \underset{q \in \mathcal{G}(n,k)}{\operatorname{argmin}} \sum_{j=1}^{m} d^2(q, p_j) \\ &= \quad \underset{q \in \mathcal{G}(n,k)}{\operatorname{argmin}} \sum_{j=1}^{m} \sum_{i=1}^{k} \theta_{i,j}^2. \end{aligned} \tag{5.17}$$

So $\bar{p} \in \mathcal{G}(n,k)$ is the $k$-plane in $\mathbb{R}^n$ that minimizes the sum of the squares of all of the principle angles between itself and the $m$ other $k$-planes. We will use a modified version of Algorithm 4 in [5] in order to calculate the Karcher mean for the Grassmann manifold, given by Algorithm 5.2.

The final item we need to deal with is the restrictions we need to put on our simplex in order to guarantee that we can find the required geodesics and Karcher mean. First, we can always find a unique geodesic between $p \in \mathcal{G}(n,k)$ and $q \in \mathcal{G}(n,k)$ as long

**Algorithm 5.2** ***Karcher Mean Algorithm for Grassmann Manifolds***
*Given the points $p_1, \ldots, p_m \in \mathcal{G}(n,k)$, fix a tolerance $\delta > 0$ and set $q = p_1$. Iterate the following:*

1. *Let*

$$\omega \quad = \quad \frac{1}{m} \sum_{i=1}^{m} \mathrm{Log}_q(p_i).$$

*If $\|\omega\| < \delta$, return $\bar{p} = q$, else, go to Step 2.*

2. *Find the SVD*

$$U \Sigma V^T \quad = \quad \omega$$

*and, let*

$$q \quad \rightarrow \quad qV \cos(\Sigma) + U \sin(\Sigma).$$

*Go to Step 1.*

as every principle angle between $p$ and $q$ is less than $\pi/2$ [30]. Also, if there exists a $q \in \mathcal{G}(n,k)$ such that $p_1, \ldots, p_n \in B_{\pi/4}(q)$, then the Karcher mean exists and is unique [5, 31]. Since $d(p,q) \leq \min(\sqrt{k}, \sqrt{n-k})\pi/2$ for any $p, q \in \mathcal{G}(n,k)$ [30], the simplex can typically grow quite large in practice.

To test the algorithm, let us consider the simple example of minimizing the squared distance to $I_{n,k}$, i.e., the first $k$ columns of the $n$-by-$n$ identity matrix. Here we'll take $n = 5$ and $k = 2$. Then the function we are trying to minimize is given by (5.15). We pick a random element $g_0 \in \mathcal{G}(5,2)$ as an initial guess for the minimizer. This was one of the vertices of our initial simplex. The other vertices were chosen around $g_0$ such that the restrictions on the size of the simplex were met and, the simplex was non-degenerate. We did this for 1000 different $g_0$. In order to restrict the size of the simplex, we chose our neighborhood to be $B_{\pi/4}(\bar{p})$ and, allowed the $\rho$, $\epsilon$ and $\chi$ parameters to vary in each iteration of the algorithm. Given (5.15) for our distance formula, we see that only the expansion step of the simplex algorithm can cause our geodesic to leave the neighborhood $B_{\pi/4}(\bar{p})$. We can find the maximum allowed value of $\epsilon$ from the formula

$$\epsilon_{max} \quad = \quad \frac{\pi}{4} \left( \sum_{i=1}^{k} \theta_i^2 \right)^{-1/2}. \tag{5.18}$$

If $\epsilon_{max} > 2$, we can proceed with the iteration without having to adjust $\rho$, $\epsilon$ and $\chi$, otherwise, we need to scale these parameters. The results are shown in Table 2, where $g_f$ is the solution returned by the Nelder-Mead algorithm and `cputime` is the $MATLAB^{\textcircled{R}}$ command for measuring the CPU time.

Table 2: Minimizing $d^2(I_{5,2}, g)$

| Average $d^2(I_{5,2}, g_0)$ | Average $d^2(I_{5,2}, g_f)$ | Average `cputime` |
|---|---|---|
| 1.9672 | 2.1055 e(-15) | 7.0491 secs |

# 6 The LTMADS algorithm

The mesh adaptive direct search (MADS) algorithms attempt to minimize a function $f(\mathbf{x})$, $\mathbf{x} \in \Omega$, without explicitly or implicitly using any derivative information [3]. So the MADS algorithms are similar in spirit to the Nelder-Mead simplex algorithm, though the details of implementation differ. Also, the MADS algorithms have general convergence results [1, 3], something that is lacking for the Nelder-Mead algorithm.

We will take the feasible set $\Omega$ to be $\mathbb{R}^n$. The algorithms start at iteration $k = 0$ with an initial guess $\mathbf{p}_0 \in \mathbb{R}^n$ and, an implicit mesh $M_0$. The mesh itself is constructed from some finite set of directions $D \subset \mathbb{R}^n$, where the $n$-by-$n_D$ matrix $D$ satisfies the restrictions:

1. Nonnegative linear combinations of the columns of $D$ span $\mathbb{R}^n$, i.e., $D$ is a positive spanning set, and;
2. Each column of $D$ is of the form $G\mathbf{z}$ for some fixed matrix $G \in \mathcal{GL}(n)$ and integer-valued vector $\mathbf{z} \in \mathbb{Z}^n$.

The columns of $D$ will be dilated by the mesh size parameter $\Delta_k^m > 0$. From our current mesh, we select $0 \leq \kappa < \infty$ points to evaluate $f(\mathbf{x})$ at. All of the evaluation points are put into the set $S_k$. Then, at each iteration k, the current mesh will be defined by

$$M_k \quad = \quad \bigcup_{\mathbf{p} \in S_k} \{\mathbf{p} + \Delta_k^m D\mathbf{z} \,|\mathbf{z} \in \mathbb{N}^{n_D}\}. \tag{6.1}$$

Each iteration of the MADS algorithms have an (optional) search step and a poll step. The search step selects the $\kappa$ points from the mesh $M_k$ in any user defined way. The idea is to attempt to find a point on $M_k$ that will reduce the value of $f(\mathbf{x})$ and, hence, give us a better candidate solution $\mathbf{p}_k$ to our problem. The poll step is run whenever the search step fails to generate an improved candidate solution. Then we do a local exploration of the current mesh $M_k$ near the current candidate solution $\mathbf{p}_k$. In addition to the mesh size parameter $\Delta_k^m$, the MADS algorithms also have a poll size parameter $\Delta_k^p$ that satisfies the conditions

1. $\Delta_k^m \leq \Delta_k^p$ for all $k$, and;
2. $\lim_{k \in K} \Delta_k^m = 0 \Leftrightarrow \lim_{k \in K} \Delta_k^p = 0$ for any infinite set $K \subset \mathbb{N}$ of indices.

If both the search and poll steps fail to find an improving point on $M_k$, we refine the mesh by letting $\Delta_{k+1}^m < \Delta_k^m$.

At iteration $k$, the mesh points chosen from $M_k$ around our current best point $\mathbf{p}_k$ during the poll step are called the frame $P_k$, where

$$P_k \quad = \quad \{\mathbf{p}_k + \Delta_k^m \mathbf{d} \,|\mathbf{d} \in D_k\}, \tag{6.2}$$

where $\mathbf{d} = D\mathbf{u}_k$ for some $\mathbf{u}_k \in \mathbb{N}^{n_D}$. $D_k$ and $\mathbf{d}$ need to meet certain other requirements given in [3]. These requirements will be satisfied by the specific MADS algorithm we examine below, so we will not concern ourselves with them. The same comment holds for the updating procedures for $\Delta_k^m$ and $\Delta_k^p$. So we have the following algorithm:

**Algorithm 6.1** ***A General MADS Algorithm*** *[3]*
*Initialize the parameters* $\mathbf{p}_0$, $\Delta_0^m \leq \Delta_0^p$, *D, G and* $k = 0$.

1. *Perform the (optional) search step and (possibly) the poll step.*
2. *Update* $\Delta_k^m$ *and* $\Delta_k^p$, *set* $k = k + 1$ *and return to Step 1.*

Let us describe the lower-triangular, mesh adaptive direct search (LTMADS) algorithm in [3] in more detail. Specifically, in the language of [3], we will look at the LTMADS algorithm with a minimal positive basis poll and dynamic search. We will be trying to minimize some function $f(\mathbf{x})$ over $\mathbb{R}^n$. The general idea of the algorithm is to have an adaptive mesh around the current candidate $\mathbf{y}$ for our optimal point. At some subset of the points of the current mesh, we will do a poll to see if we can find a point $\mathbf{z}$ such that $f(\mathbf{z}) < f(\mathbf{y})$. If this occurs we will 'expand' the current mesh in order to look at points further away from $\mathbf{z}$ that could potentially reduce the value of $f(\mathbf{x})$. Also, if a poll is successful, we will search further along the direction that helped reduce the value of $f(\mathbf{x})$. Now, if our search and poll steps prove unsuccessful, then we may be around a (local) minimum. In that case, we 'contract' the mesh around our current candidate $\mathbf{y}$ for the point that minimizes $f(\mathbf{x})$. In this way, we can do refined search and poll steps around $\mathbf{y}$. An important point is that the mesh is never actually constructed during the algorithm. All that needs to be done is to define the search and poll step points so that they would be points on the mesh if it was explicitly constructed. This and many more details can be found in [3].

Our version of the LTMADS will proceed as follows. Let $G = I$ and $D = [I \;\; -I]$. Initialize an iteration counter $k = 0$, poll counter $l_c = 0$ and, the mesh ($\Delta_0^m = 1$) and poll ($\Delta_0^p = n$) size parameters. Let $\mathbf{p}_0 \in \mathbb{R}^n$ be an initial guess for the minimizer of $f(\mathbf{x})$ and, $f_0 = f(\mathbf{p}_0)$. We will describe the poll step of the algorithm first since the search step depends on the results of the previous poll step. Let $l = -\log_4(\Delta_k^m)$. The first step is to create an $n$-dimensional vector $b_l$. First we check if $b_l$ was previously created. If $l_c > l$, return the previously stored $b_l$ and exit the $b_l$ construction step. Otherwise, let $l_c = l_c + 1$ and construct $b_l$. To do this construction, randomly select an index $\iota$ from the set $N = \{1, \ldots, n\}$ and, randomly set $b_l(\iota)$ to be $\pm 2^l$. For $i \in N \setminus \{\iota\}$, randomly set $b_l(i)$ to be one of the integers from the set $S = \{-2^l + 1, \ldots, 2^l - 1\}$. Save $\iota$ and $b_l$.

Now that we have $b_l$, we need to construct the points on our current mesh where we will potentially evaluate $f(\mathbf{x})$ at during the poll step. Construct the $(n-1)$-by-$(n-1)$ lower-triangular matrix $L$ as follows. Set the diagonal elements of $L$ randomly to $\pm 2^l$. Set the lower components of $L$ to a randomly chosen element of the set $S$ given above. Finally, randomly permute the rows of $L$. Construct the new matrix $B$ from $b_l$ and $L$ such that

$$B = \begin{bmatrix} L(1:\iota-1,:) & b_l(1:\iota-1) \\ \mathbf{0}^T & b_l(\iota) \\ L(\iota:n-1,:) & b_l(\iota+1:n) \end{bmatrix}. \tag{6.3}$$

Randomly permute the columns of $B$. Finally, construct the matrix

$$D_k = [B \ -B\mathbf{1}]. \tag{6.4}$$

Having constructed the matrix $D_k$, we will now evaluate $f(\mathbf{x})$ at the mesh points $\mathbf{x} = \mathbf{d}_i$, where $\mathbf{d}_i = \mathbf{p}_k + \Delta_k^m D_k(:,i)$, $i = 1, \ldots, n+1$. If we find a $\mathbf{d}_i$ such that $f(\mathbf{d}_i) < f_k$, let $\mathbf{p}_{k+1} = \mathbf{d}_i$ and $f_{k+1} = f(\mathbf{d}_i)$ and, exit the loop. Otherwise, let $\mathbf{p}_{k+1} = \mathbf{p}_k$ and $f_{k+1} = f_k$. After the completion of the loop let $k = k+1$. Notice that we do not necessarily evaluate $f(\mathbf{x})$ at all of the $\mathbf{d}_i$.

Now we can describe the search step at iteration $k+1$, which actually precedes the poll step. If the previous poll step at iteration $k$ found an improved candidate solution $\mathbf{p}_{k+1}$, find the mesh point $\mathbf{s}_{k+1} = \mathbf{p}_k + 4\Delta_k^m D_k(:,i)$, where $D_k(:,i)$ was the direction that improved the function evaluation in the previous poll step during iteration $k$. If $h = f(\mathbf{s}_{k+1}) < f_{k+1}$, let $\mathbf{p}_{k+2} = \mathbf{s}_{k+1}$ and $f_{k+2} = h$, update the iteration count to $k+2$ and, skip the poll step. Otherwise, proceed to the poll step for iteration $k+1$.

After doing the search and poll steps, the size parameters will be updated according to the following rule. If the search and poll steps did find an improved candidate solution $\mathbf{p}_{k+1}$, we may be around a local minimum of $f(\mathbf{x})$. Then we want to refine the mesh by letting $\Delta_{k+1}^m = 1/4\Delta_k^m$. Otherwise, we want to allow ourselves to search in a larger neighborhood around our current candidate solution in order to try and reduce the value of $f(\mathbf{x})$ further. However, we also want to restrict the size of the neighborhood that we look in. So, if we found an improved candidate solution $\mathbf{p}_{k+1}$ in the search or poll step and $\Delta_k^m < 1/4$, let $\Delta_{k+1}^m = 4\Delta_k^m$. Otherwise, let $\Delta_{k+1}^m = \Delta_k^m$. Finally, update the poll size parameter $\Delta_{k+1}^p = n\sqrt{\Delta_{k+1}^m}$.

There is a choice of termination criterion that can be used for the LTMADS algorithm. One can terminate the algorithm when $\Delta_k^p$ drops below some specified tolerance. Alternately, one could choose to end the iterations when a specified number of function evaluations is exceeded. These two termination criterion can be combined, exiting the algorithm whenever one of the criteria are met, as was done in [3]. Finally, convergence results for LTMADS are also examined in [1, 3].

# 7 The LTMADS algorithm on Riemannian manifolds

How can we generalize the LTMADS algorithm to a Riemannian manifold $\mathcal{M}$? The key insight is to realize that the $D_k$ are tangent vectors to our current candidate solution $\mathbf{p}_k \in \mathbb{R}^n$. So, the mesh is actually in the tangent space $T_{\mathbf{p}_k}\mathbb{R}^n$. Let us examine this in more detail when $\mathcal{M}$ is a sub-manifold of $\mathbb{R}^n$. In section 8 we will look at an example when we do not have an $\mathcal{M}$ that is explicitly embedded in $\mathbb{R}^n$. Note that the length we travel along a geodesic in the LTMADS algorithm will be the length of the corresponding tangent vector.

At each iteration $k$, we have a current candidate solution to our problem given by $\mathbf{p}_k$. In the search step, if it is performed during the current iteration, we evaluate $f(\mathbf{x})$ at the point $\mathbf{s}_k = \mathbf{p}_{k-1} + 4\Delta_{k-1}^m D_{k-1}(:, j)$, where $D_{k-1}(:, j)$ was the direction that improved the function evaluation in the previous poll step. Similarly, the poll step potentially evaluates $f(\mathbf{x})$ at the points $\mathbf{d}_i = \mathbf{p}_k + \Delta_k^m D_k(:, i)$, $i = 1, \ldots, n+1$. So, to our current candidate solution $\mathbf{p}_k$, we are adding some vector $\mathbf{v} = \mathrm{Exp}_{\mathbf{p}_k}(\omega) = \omega$, where $\omega \in T_{\mathbf{p}_k}\mathbb{R}^n \simeq \mathbb{R}^n$.

Now, the vectors that we add to $\mathbf{p}_k$ are on a mesh. It follows that this mesh actually is in the tangent space $T_{\mathbf{p}_k}\mathbb{R}^n$, with each point on the mesh corresponding to some tangent vector at $\mathbf{p}_k$. Further, aside from 'expanding' or refining the mesh, each point in the mesh, which corresponds to some tangent vector, is parallel transported to the new candidate solution $\mathbf{p}_k$. That is, the mesh is not 'rotated' in $\mathbb{R}^n$ as we move from one candidate solution to a new one. This is how, e.g., we can use the same $b_l$ to initialize the construction of $D_k$ and $D_{k'}$ when $\log_4(\Delta_k^m) = \log_4(\Delta_{k'}^m)$.

A brief word on restricting the size of our tangent vectors is in order. As shown in [3], we have that $\|\mathbf{d}_i - \mathbf{p}_k\|_2 \leq \sqrt{n\Delta_k^m}$. So, we can control how far we need to move along a geodesic by controlling the size of $\Delta_k^m$. However, this is not much of a concern in the LTMADS algorithm because we are given initial conditions $\mathbf{p}_k$ and $\omega \in T_{\mathbf{p}_k}\mathcal{M}$ for our geodesic and, move a time step $\tau = 1$ along the geodesic given by $\gamma(\tau) = \mathrm{Exp}_{\mathbf{p}_k}(\tau\omega)$. Provided our manifold $\mathcal{M}$ is geodesically complete, as it is when our manifold is $\mathbb{R}^n$, we can move as far along the geodesic as we wish. Since we never have the two-endpoint problem of trying to find a geodesic that connects the two points $p, q \in \mathcal{M}$, nor do we need to find a Karcher mean, we do not typically have the need to restrict ourselves to some neighborhood $U \subset \mathcal{M}$, as we did for the Nelder-Mead algorithm. We can remove the requirement that $\mathcal{M}$ be geodesically complete by using, potentially position dependent, restrictions on the size of $\Delta_k^p$. $\Delta_k^p$ is the upper bound on the lengths of the geodesics considered at any poll step. We would also require that the search step geodesic is also calculable.

Let us walk through how the LTMADS algorithm will be modified when we use it on a Riemannian manifold $\mathcal{M}$ of $n$ dimensions, which we will assume is geodesically complete. We are trying to minimize the function $f(q)$, where $q \in \mathcal{M}$. Let $p_0 \in \mathcal{M}$ be our initial candidate solution. As long as we remain at $p_0$, the LTMADS proceeds exactly as above except for one change. Our mapping of the set of tangent vectors $D_k \in T_{p_0}\mathcal{M}$ into $\mathcal{M}$ will now be given by $d_i = \mathrm{Exp}_{p_0}(\Delta_k^m \omega_i)$, where $\omega_i \in D_k$ for

$i = 1, \ldots, n+1$.

Now assume that the previous poll step found a new candidate solution $p_k$, i.e., we found a new potential solution to our minimization problem. Then we will need to do the search step. This, however, is trivial. If the improving geodesic from the previous poll step is given by $\mathrm{Exp}_{p_{k-1}}(\Delta_{k-1}^m \omega_j)$, then the point $s_k$ where we need to evaluate $f(q)$ at is given by $s_k = \mathrm{Exp}_{p_{k-1}}(4\Delta_{k-1}^m \omega_j)$.

The only remaining problem is how to modify the poll step after we find a new candidate solution $p_k$. At $p_{k-1}$ we have the vectors $b_l$ and the columns of $D$. These are parallel transported along the geodesic we previously computed connecting $p_{k-1}$ to our new candidate solution $p_k$. This is exactly what we do in $\mathbb{R}^n$. There is a unique way to do parallel transportation on $\mathcal{M}$ that depends linearly on the tangent vector to be transported, does not change the inner product of two tangent vectors (compatible with the metric) and, does not 'rotate' the tangent vector (torsion-free). Now, after the first parallel transport, $G = I$ will go to some $n$-by-$n$ orthogonal matrix $O$, because parallel transport preserves the inner products of tangent vectors. So now $D = [O \quad -O]$. Remember that we require $d = Du_k$ for some $u_k \in \mathbb{N}^{n_D}$, where $d$ is a column of the matrix $D_k$ in (6.4). When $D = [I \quad -I]$ this was accomplished with our above construction of $D_k$ in (6.4). Now, however, we need to remember that a negative element of the $n$-by-$n$ $B$ matrix in (6.3) corresponds to one of the columns of the $-I$ matrix in $D$. So, after parallel transporting our vectors, we can no longer use $B$ as given in (6.3). Rather, $B$ must be constructed as follows. Let

$$B' \quad = \quad \begin{bmatrix} \widehat{B} \\ 0_{n \times n} \end{bmatrix}, \tag{7.1}$$

where $\widehat{B}$ is the matrix in (6.3). In each column of $B'$, find the negative elements. If one of these elements is $b_{ij}$, let $b_{i(j+n)} = 0$ be replaced by $-b_{ij} > 0$ and set $b_{ij} = 0$. Finally, our new $B$ matrix is given by

$$B \quad = \quad DB'. \tag{7.2}$$

Note, however, that this procedure is equivalent to simply letting $B \rightarrow OB$ in (6.3). Further, this implies that we do not need to explicitly parallel transport the $b_l$ or alter the method of finding the $B$ matrix, since these are easily given once we have parallel transported the $G$ matrix. Everything else will proceed as in the poll step given above for the LTMADS algorithm in $\mathbb{R}^n$.

# 8 The Whitney embedding algorithm

Now we'll examine the problem that led to our consideration of direct search methods over Riemannian manifolds: the Whitney embedding algorithm [7, 8, 10]. This is a particularly useful method for reducing the dimensionality of data. As we will see, the problem is well suited to the generalized LTMADS algorithm since we are required to minimize a non-differentiable function over $\mathcal{G}(n, k)$.

Let us first recall Whitney's theorem.

**Theorem 8.1** ***Whitney's Easy Embedding Theorem*** *[15]*

*Let $\mathcal{M}$ be a compact Hausdorff $C^r$ $n$-dimensional manifold, $2 \leq r \leq \infty$. Then there is a $C^r$ embedding of $\mathcal{M}$ in $\mathbb{R}^{2n+1}$.*

Let $\mathcal{M} \subset \mathbb{R}^m$ be an $n$-dimensional manifold. The method of proof for theorem 8.1 is, roughly speaking, to find a $(2n+1)$-plane in $\mathbb{R}^m$ such that all of the secant and tangent vectors associated with $\mathcal{M}$ are not completely collapsed when $\mathcal{M}$ is projected onto this hyperplane. Then this element $p \in \mathcal{G}(m, 2n+1)$ contains the low-dimensional embedding of our manifold $\mathcal{M}$ via the projection $p^T\mathcal{M} \subset \mathbb{R}^{2n+1}$. An important idea is to make this embedding cause as little distortion as possible. By this we mean, what $p \in \mathcal{G}(m, 2n+1)$ will minimize the maximum collapse of the worst tangent vector? In this way we can keep the low-dimensional embedding from almost self-intersecting as much as possible. It is also possible to achieve a close to isometric embedding by finding an optimal $p \in \mathcal{G}(m, 2n+1)$ [10].

Now, in practice, we will only have some set $\mathcal{P} = \{\mathbf{x} | \mathbf{x} \in \mathcal{M} \subset \mathbb{R}^m\}$ of sample points from our manifold $\mathcal{M}$. We can then form the set of unit length secants $\Sigma$ that we have available to us, where

$$\begin{aligned} \Sigma &= \left\{ \frac{\mathbf{x}-\mathbf{y}}{\|\mathbf{x}-\mathbf{y}\|_2} \middle| \mathbf{x}, \mathbf{y} \in \mathcal{P} \right\} \\ &= \{\sigma_i | i = 1, 2, \dots\}. \end{aligned} \tag{8.1}$$

So now our problem is stated as finding an element $\hat{p} \in \mathcal{G}(m, 2n+1)$ such that

$$\begin{aligned} \hat{p} &= \underset{p \in \mathcal{G}(m,2n+1)}{\operatorname{argmin}} \left[ - \min_{\sigma_i \in \Sigma} \|p^T \sigma_i\|_2 \right] \\ &= \underset{p \in \mathcal{G}(m,2n+1)}{\operatorname{argmin}} S(p). \end{aligned} \tag{8.2}$$

The function $S(p)$ in (8.2) is non-differentiable, so we need to use a direct search method to minimize it over $\mathcal{G}(m, 2n+1)$. A similar method for finding an approximation to our $\hat{p}$ was presented in [8] where a smooth function $\widehat{S}(p)$ was used to approximate our $S(p)$. Then the algorithm in [11] was used to minimize $\widehat{S}(p)$ over $\mathcal{G}(m, 2n+1)$. Also, the dimensionality reduction in (8.2) can be done by reducing the embedding dimension by one and iterating [10]. However, this leads to a concave quadratic program who's solution method is quite complicated [6].

Since the Whitney algorithm is performed over $\mathcal{G}(n, k)$, which is a geodesically complete manifold [2], let us give the required formulas in order to run the LTMADS algorithm [11]. We need to be able to find a geodesic given initial conditions. Also, we need to be able to parallel transport tangent vectors along this geodesic. Let $p \in \mathcal{G}(n, k)$ and $\nu, \omega \in T_p\mathcal{G}(n, k)$, where we have the SVD $\nu = U\Sigma V^T$. Then, the geodesic $\gamma(\tau)$ with $\gamma(0) = p$ and $\dot{\gamma}(0) = \nu$ will be given by

$$\gamma(\tau) = [pV\ U] \begin{bmatrix} \cos(\Sigma\tau) \\ \sin(\Sigma\tau) \end{bmatrix} V^T. \tag{8.3}$$

The parallel translation $\omega(\tau)$ of $\omega$ along $\gamma(\tau)$ in (8.3) is

$$\omega(\tau) \quad = \quad \left( [pV\ U] \left[ \begin{array}{c} -\sin(\Sigma\tau) \\ \cos(\Sigma\tau) \end{array} \right] U^T + U_\perp U_\perp^T \right) \omega. \tag{8.4}$$

Also, the $G$ we start with will be a $k(n-k)$-dimensional identity matrix which will form an orthonormal basis, after reshaping the columns, for the tangent space given by (see (5.9) also)

$$T_p\mathcal{G}(n,k) \quad = \quad \left\{ \omega \,\middle|\, \omega = p_\perp g \text{ and } g \in \mathbb{R}^{(n-k)\times k} \right\}. \tag{8.5}$$

It is this $G$ that we will parallel transport, as the LTMADS algorithm proceeds, as follows. Each column of $G$ corresponds to to a $g$ in (8.5) when it is reshaped into an $(n-k)$-by-$k$ matrix. We then multiply these matrices by $p_\perp$. It is these matrices that we will parallel transport, the matrix $G$ being a convenient method for storing the resulting $g$ matrices.

For a numerical example, let us consider the embedding in $\mathbb{R}^{20}$ of the ten-dimensional complex Fourier-Galerkin approximation to the solution $u(x,t)$ of the Kuramoto-Sivashinsky equation [7, 8]

$$u_t + 4u_{xxxx} + 87\left(u_{xx} + \frac{1}{2}(u_x)^2\right) \quad = \quad 0. \tag{8.6}$$

Here we have $951$ points from a two-dimensional manifold in $\mathbb{R}^{20}$ that is known to have an embedding into $\mathbb{R}^3$. So, we want to solve (8.2) where $p \in \mathcal{G}(20,3)$. (The $2n+1$ in Whitney's theorem is an upper bound on the possible embedding dimension.) Let

$$\epsilon \quad = \quad \min_{\sigma_i \in \Sigma} \|\hat{p}^T \sigma_i\|_2, \tag{8.7}$$

where $\hat{p} \in \mathcal{G}(20,3)$ is the solution to (8.2). Then a larger value of $\epsilon$ indicates a better projection of the data into a low-dimensional subspace of the original $\mathbb{R}^{20}$. The method presented in [8] for finding $\hat{p}$ resulted in $\epsilon = .01616$. The method in [10], using $MATLAB^{\textcircled{R}}$'s `quadprog` to solve the quadratic programs rather than the computationally intensive method in [6], gave $\epsilon = .02386$. In contrast, the LTMADS method resulted in $\epsilon = .03887$ after $20$ iterations. The initial $p_0$ in the LTMADS algorithm was taken to be the three left singular vectors that corresponded to the three largest singular values of the $20$-by-$451725$ matrix of unit-length secant vectors, where we used either $\sigma_j$ or $\sigma_i$ when $\sigma_j = -\sigma_i$.

The LTMADS algorithm seems to be a significant improvement over previous methods for finding the Whitney embedding of data sampled from a manifold. As previously mentioned, this improvement is useful for finding an embedding that is as close to isometric as possible. This is itself important because it introduces as little distortion in the projected data as possible. Following the Whitney projection, one can employ the semi-definite programming method in [10] to further improve the embedding. An alternate method would be to run a numerical implementation of Günther's theorem [13, 14].

# 9 LTMADS for constrained optimization problems

Using the LTMADS algorithms for constrained optimization was already considered in [3]. Here, we will show how this can be done as an unconstrained minimization problem using LTMADS over a Riemannian manifold $\mathcal{M}$. The manifold $\mathcal{M}$ will enforce the constraints of our original problem.

For the LTMADS algorithm, we will consider the following constrained optimization problem:

$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{R}^n} \quad & f(\mathbf{x}) && \text{(9.1a)} \\ \text{subject to} \quad & \mathbf{g}(\mathbf{x}) = \mathbf{0}. && \text{(9.1b)} \end{aligned}$$

We can convert any constrained optimization problem to this form by adding in slack variables $\mathbf{z}$ to change any inequality constraints into equality constraints and, defining the extended function

$$f_e(\mathbf{x},\mathbf{z}) \quad = \quad \left\{ \begin{array}{cc} f(\mathbf{x}) & \text{if } \mathbf{z} \geq \mathbf{0} \\ \infty & \text{otherwise} \end{array} \right. .$$

Notice that $f_e(\mathbf{x},\mathbf{z})$ will implicitly enforce the condition $\mathbf{z} \geq \mathbf{0}$. Now, using the Morse-Sard theorem, it is typically the case that the level set $\mathbf{g}(\mathbf{x}) = \mathbf{0}$ of the mapping $\mathbf{g} : \mathbb{R}^n \rightarrow \mathbb{R}^m$ is a regular Riemannian manifold $\mathcal{M}$ of dimension $n-m$ embedded in $\mathbb{R}^n$ [23]. Additionally, if $\mathcal{M}$ is closed, then it is also geodesically complete. By using the LTMADS algorithm over this implicitly defined manifold, we are changing the constrained optimization problem (9.1) into an unconstrained optimization problem on $\mathcal{M}$. The price to be paid for this is the expense of calculating the required geodesics and parallel transports when we run the LTMADS algorithm on $\mathcal{M}$.

Now, let us give the system of differential equations that needs to be solved for calculating the geodesics on $\mathcal{M}$ [11], namely

$$\begin{aligned} \ddot{x}^k &= \dot{\mathbf{x}}^T \left[ \sum_{i=1}^{m} \left[ -\mathbf{g}_{x^k}^T \left[ \nabla\mathbf{g} (\nabla\mathbf{g})^T \right]^{-1} \right]^i \nabla^2 g^i \right] \dot{\mathbf{x}} && \text{(9.2a)} \\ &= \dot{\mathbf{x}}^T \mathbf{L}_{xx}^k \dot{\mathbf{x}}, && \text{(9.2b)} \end{aligned}$$

with the initial conditions given by the position vector $\mathbf{x}_0$, where $\mathbf{g}(\mathbf{x}_0) = \mathbf{0}$, and the tangent vector $\dot{\mathbf{x}}_0$. In (9.2a), $x^k$ is the $k$th component of $\mathbf{x}$, $g^i$ is the $i$th component of $\mathbf{g}(\mathbf{x})$ and, $\mathbf{g}_{x^k}$ is the partial derivative of $\mathbf{g}(\mathbf{x})$ with respect to $x^k$. Setting $\Gamma^k = -\mathbf{L}_{xx}^k$ gives us our Christoffel symbols. So (9.2b) can be rewritten in the standard notation

$$\ddot{x}^k \quad = \quad -\Gamma_{ij}^k \dot{x}^i \dot{x}^j. \tag{9.3}$$

Along the geodesic $\mathbf{x}(\tau)$ calculated in (9.3), we can parallel transport a tangent vector $\omega \in T_{\mathbf{x}_0}\mathcal{M}$ via the equation

$$\dot{\omega}^k \quad = \quad -\Gamma_{ij}^k \dot{x}^i \omega^j. \tag{9.4}$$

From (9.3) and (9.4), we see that many systems of nonlinear ODEs would need to be solved in order to do the operations required by the LTMADS on $\mathcal{M}$, at least for the poll step. The search step could potentially be done by using approximations to the systems of nonlinear ODEs. If a potential new candidate solution is located with the approximations, we could then solve the exact equations to see if it should become our new candidate solution. However, we are still limited by the fact that at least the poll step would require us to solve (9.3) and (9.4). This may not be a fatal limitation, however, as the reduced gradient methods need to do a similar procedure in order to remain on the level set $\mathbf{g}(\mathbf{x}) = \mathbf{0}$ [4, 11].

The Runge-Kutta algorithm can be used to solve (9.3) and (9.4) [29]. These solution methods are given by Algorithm 9.1 and Algorithm 9.2, respectively. At each step of Algorithm 9.1, we need to make sure that $\mathbf{x}_\alpha$ satisfies $\mathbf{g}(\mathbf{x}_\alpha) = \mathbf{0}$ and, that $\mathbf{y}_\alpha$ lies in the tangent plane $T_{\mathbf{x}_\alpha}\mathcal{M}$. Also, since we only need to parallel transport the $G$ matrix from the LTMADS algorithm, in Algorithm 9.2 we need to make sure that the parallel transported $G_\alpha = G(\alpha h)$ matrix satisfies $G_\alpha^T G_\alpha = I$ and, that the elements of $G$ all lie in the tangent plane $T_{\mathbf{x}_\alpha}\mathcal{M}$. Any errors need to be corrected for at each step of the algorithms. This can be done by projecting $\mathbf{x}_\alpha$ back onto $\mathcal{M}$ and, then projecting $\mathbf{y}_\alpha$ onto the resulting tangent plane. Additionally, the elements of $G$ can be projected onto $T_{\mathbf{x}_\alpha}\mathcal{M}$ and, then orthonormalized.

**Algorithm 9.1** ***Geodesic Equation Solver*** *[29]*
*Fix an integer $m$ and, let the step size be given by $h = T/m$. Let $\alpha = 1, \ldots, m$, $\mathbf{x}_\alpha = \mathbf{x}(\alpha h)$,*

$$\begin{aligned} \mathbf{y}_\alpha &= \dot{\mathbf{x}}_\alpha, \\ X_\alpha &= [\mathbf{x}_\alpha \ \ \mathbf{y}_\alpha] \text{ and,} \\ F(X_\alpha) &= \left[\mathbf{y}_\alpha \ \ -\Gamma_{ij}^k(\mathbf{x}_\alpha) y_\alpha^i y_\alpha^j\right]. \end{aligned}$$

*Then the Runge-Kutta algorithm is given by*

$$\begin{aligned} X_0 &= [\mathbf{x}_0 \ \ \dot{\mathbf{x}}_0], & X_{\alpha+1} &= X_\alpha + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4), \\ k_1 &= F(X_\alpha), & k_2 &= F(X_\alpha + k_1/2), \\ k_3 &= F(X_\alpha + k_2/2) \quad \text{and} & k_4 &= F(X_\alpha + k_3). \end{aligned}$$

Now let us consider the linear optimization problem on an $n$-dimensional solid hypersphere from [3]:

$$\min_{\mathbf{x}\in\mathbb{R}^n} \quad \mathbf{1}^T\mathbf{x} \tag{9.5a}$$

$$\text{subject to} \quad \mathbf{x}^T\mathbf{x} \leq 3n. \tag{9.5b}$$

By adding in the slack variable $z$ and defining the extended function as in (9.2), we can convert (9.5) into the form in (9.1). Actually, we could just take (9.5b) as already giving us an $n$-dimensional manifold in $\mathbb{R}^n$, which itself provides the (global) coordinates for

**Algorithm 9.2** ***Parallel Transport Equation Solver***
*Fix an integer $m$ and, let the step size be given by $h = T/m$. Let $\mathbf{x}_\alpha = \mathbf{x}(\alpha h)$ and $\dot{\mathbf{x}}_\alpha = \dot{\mathbf{x}}(\alpha h)$, $\alpha = 1, \ldots, m$, be the solutions from Algorithm 9.1. Finally, let $\omega_\alpha = \omega(\alpha h)$ and $H(\omega_\alpha) = -\Gamma^k_{ij}(\mathbf{x}_\alpha)\dot{x}^i_\alpha \omega^j_\alpha$. Then the Runge-Kutta algorithm is given by*

$$\begin{aligned} \omega_0 &= \omega(0), & \omega_{\alpha+1} &= \omega_\alpha + \frac{h}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right), \\ k_1 &= H(\omega_\alpha), & k_2 &= H(\omega_\alpha + k_1/2), \\ k_3 &= H(\omega_\alpha + k_2/2) \quad \text{and} & k_4 &= H(\omega_\alpha + k_3). \end{aligned}$$

$\mathcal{M}$. Then this would reduce to the standard LTMADS algorithm. Instead of using either of these cases, we will simply replace the inequality in (9.5b) with an equality to have:

$$\min_{\mathbf{x}\in\mathbb{R}^n} \quad \mathbf{1}^T\mathbf{x} \tag{9.6a}$$

$$\text{subject to} \quad \mathbf{x}^T\mathbf{x} = 3n. \tag{9.6b}$$

Then we will be searching over the $n$-dimensional hypersphere. This is equivalent to the original problem since the solution to (9.5) is given by $\mathbf{x} = -\sqrt{3}\ \mathbf{1}$, where the optimal value is $-\sqrt{3}\ n$.

The equations for the Christoffel symbols associated with (9.6b) are

$$\Gamma^k_{ij} \quad = \quad \frac{x^k}{3n} I. \tag{9.7}$$

In Algorithm 9.1, at each step we renormalized $\mathbf{x}_\alpha$ so that $\mathbf{x}^T_\alpha \mathbf{x}_\alpha = 3n$ and, projected $\mathbf{y}_\alpha$ onto the tangent plane using the projection operator $P_\alpha = (I - 1/(3n)\mathbf{x}_\alpha \mathbf{x}^T_\alpha)$. In Algorithm 9.2, the parallel transported $G$ matrix was projected using $P_\alpha$ and, orthonormalized by setting all of the resulting singular values to unity at each step. A value of $m = 100$ was used in both Algorithms 9.1 and 9.2. A random point on the hypersphere was used as our $\mathbf{x}_0$ and, we let the initial $G$ be given by the SVD

$$P_0 \quad = \quad [G \ \ \mathbf{u}] \begin{bmatrix} S & \mathbf{0} \\ \mathbf{0}^T & 0 \end{bmatrix} V^T. \tag{9.8}$$

For this example we used the maximal positive basis LTMADS algorithm. The only difference from the minimal basis LTMADS algorithm in section 6 is that we let $D_k = [B - B]$ in (6.4) and, take $\Delta^p_k = \sqrt{\Delta^m_k}$. The dynamic search procedure was still used. We terminated the generalized LTMADS algorithm whenever $\Delta^p_k \leq 10^{-12}$ or, when $k > 600n$. The algorithm was run five times each for the dimensions $n = 5, 10, 20$ and $50$. The results are shown in Figure 1. For $n = 5, 10$ and $20$, the generalized LTMADS algorithm converged to the global solution within the allowed number of function evaluations. For $n = 50$, the number of function evaluations was exceeded, although the algorithm nearly converged. If we had used $\Delta^p_k \leq 10^{-9}$, then the algorithm would have converged for all of the values of $n$.

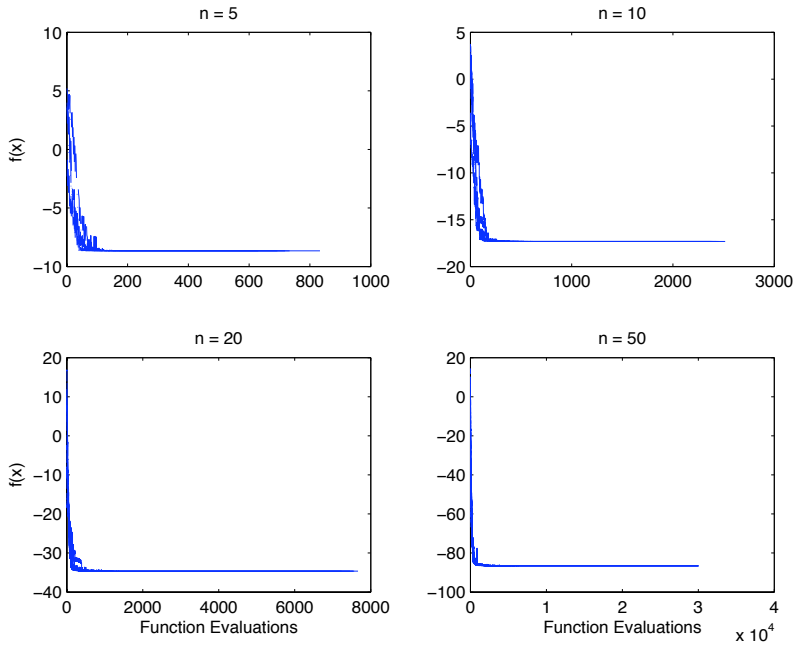


Figure 1: *The objective function value versus number of function evaluations for problem (9.6) using the LTMADS algorithm.*

# 10 A frame based method and convergence analysis

We would not be able to employ the techniques in [3] to prove convergence of the algorithm in section 7 directly because of the parallel transportation of $G$. Indeed, parallel transportation makes the generalized LTMADS algorithm much closer to the frame based method in [9, 25] than to the original LTMADS algorithm in [3]. To overcome this difficulty, we will modify our algorithm so that it becomes a frame based method. The modification only requires a slight change in the update rules for $\Delta_k^m$ and $\Delta_k^p$.

Now let $O_k$ be the parallel transported $G$ matrix at iteration $k$ in the generalized LTMADS algorithm. Then we have that

$$p_k \quad = \quad \mathrm{Exp}_{p_{k-1}}(\Delta_{k-1}^m O_{k-1} D u_{k-1}) \tag{10.1}$$

for some $u_{k-1} \in \mathbb{N}^{n_D}$. The problem is that parallel transportation is generally path dependent, so we cannot say that $p_k = \mathrm{Exp}_{p_0}(\Delta_{k-1}^m D\tilde{u}_{k-1})$ for some $\tilde{u}_{k-1} \in \mathbb{N}^{n_D}$, as was done in [3] where $\mathcal{M} = \mathbb{R}^m$. A consequence of the Hopf-Rinow-de Rham theorem is that any two points on $\mathcal{M}$ can be connected by a geodesic [28], provided $\mathcal{M}$ is geodesically complete. So we can parallel transport $O_k$ from $p_k$ back to $p_0$. Call

this $\widetilde{O}_k$. However, unless $\mathcal{M}$ is flat, as $\mathbb{R}^n$ is, we will generally have $\widetilde{O}_k \neq G$. The fact that $\widetilde{O}_k \neq G$ is what causes the problems of using the techniques in [3] to prove convergence.

Let us review the frame based method in [9]. A positive basis is a positive spanning set with no proper subset that is also a positive spanning set, see 1 before (6.1). In the LTMADS algorithm, the $D_k$ are a positive basis. A frame is defined as

$$\begin{aligned} \Phi_k &= \Phi(\mathbf{x}_k, V_k, h_k) \\ &= \{\mathbf{x}_k + h_k \mathbf{v} \,|\mathbf{v} \in V_k\}, \end{aligned} \tag{10.2}$$

where $\mathbf{x}_k$ is the central point, $V_k$ is a positive basis and, $h_k > 0$ is the frame size. A minimal frame is one for which

$$f(\mathbf{y}) \quad \geq f(\mathbf{x}_k), \text{ for all } \mathbf{y} \in \Phi_k. \tag{10.3}$$

Then $\mathbf{x}_k$ is called a minimal point. A quasi-minimal frame is one for which, given an $\epsilon > 0$,

$$f(\mathbf{y}) + \epsilon \quad \geq f(\mathbf{x}_k), \text{ for all } \mathbf{y} \in \Phi_k. \tag{10.4}$$

We call $\mathbf{x}_k$ a quasi-minimal point. With these definitions, a general frame based method is given in Algorithm 10.1. Note that $N_k$ and $H_k$ in Algorithm 10.1 are fixed until we find a quasi-minimal point $\mathbf{x}_k$ and, perform Step 6.

**Algorithm 10.1** ***Frame Based Method*** *[9]*

1. *Initialize $k = 0$. Let $\mathbf{x}_0 \in \mathbb{R}^m$ be our initial point and, choose $\beta > 0$.*
2. *Choose $N_k > 0$ and $H_k > 0$.*
3. *Construct $\Phi_k$ by any means, where $h_k > H_k$.*
4. *Do any finite search process. Let $\mathbf{x}_t$ be a point from the set $\Omega_k = S_k \cup \{p_k\} \cup \Phi_k$, where $S_k$ is the set of points from the finite search process, such that*
   *(a) $f(\mathbf{x}_t) < f_k - N_k(h_k)^{1+\beta}$ or,*
   *(b) $\mathbf{x}_t$ is the best point in $\Omega_k$.*

   *Let $k = k + 1$.*
5. *If $f(\mathbf{x}_t) < f_k - N_k(h_k)^{1+\beta}$, let $\mathbf{x}_{k+1} = \mathbf{x}_t$ and, go to Step 3.*
6. *Otherwise, perform any finite process. Let $\mathbf{x}_{k+1}$ be the best known point.*
7. *If the stopping criteria are satisfied stop. Otherwise, go to Step 2.*

Because of the difficulties introduced by parallel transport, we will use Algo-

rithm 10.2 to prove our convergence results. Here

$$\begin{aligned} \Phi_k &= \Phi(p_k, D_k, \Delta_k^m) \\ &= \left\{ \mathrm{Exp}_{p_k}(\Delta_k^m d_i) \,|d_i \in D_k \right\}. \end{aligned} \tag{10.5}$$

The modification of the LTMADS algorithm is very simple. In Step 4, we will update $\Delta_k^m$ and $\Delta_k^p$ as we did in section 6, by 'expanding' the mesh, unless $\Delta_k^m = 1$. However, in Step 5, we will always refine the mesh. Other update procedures are possible [9, Theorem 3]. Note that our modification does not prevent the mesh from 'expanding'. Theorem 10.3 below tells us that the modification does, however, build into the algorithm the fact that the mesh will become infinitely fine as the algorithm proceeds, i.e., $\lim_{k\to\infty} \Delta_k^m = 0$. This is the essential difference between the frame based method in Algorithm 10.2 and the LTMADS algorithm in section 7. Also, the assignment of $z_n$ is not required when actually running Algorithm 10.2. We only need the $z_n$ for the statements of Theorems 10.3 and 10.4.

**Algorithm 10.2** ***Frame Based Method for Riemannian Manifolds*** *[3, 9]*

1. *Initialize $n = k = 0$, $\Delta_0^m$ and $\Delta_0^p$. Let $p_0 \in \mathcal{M}$ be our initial point and, choose $\beta, \delta > 0$.*
2. *Construct $D_k$.*
3. *Do the search and/or poll step. Let $p_t$ be a point from the set $\Omega_k = \{s_k\} \cup \{p_k\} \cup \Phi_k$ such that*
   (a) *$f(p_t) < f_k - \delta(\Delta_k^m)^{1+\beta}$ or,*
   (b) *$p_t$ is the best point in $\Omega_k$.*
4. *If $f(p_t) < f_k - \delta(\Delta_k^m)^{1+\beta}$, let $k = k + 1$ and, $p_{k+1} = p_t$. Update $\Delta_k^m$ and $\Delta_k^p$ and, go to Step 2.*
5. *Otherwise, set $z_n = p_k$. Let $n = n + 1$, $k = k + 1$ and*

$$p_{k+1} = \mathrm{argmin}\left\{ f(p_k), f(p_t) \right\}.$$

   *Update $\Delta_k^m$ and $\Delta_k^p$ by refining the mesh.*
6. *If the stopping criteria are satisfied return $p_k$. Otherwise, go to Step 2.*

Now we can give our convergence result for the frame based method in Algorithm 10.2. More general frame based methods can be used without modifying the convergence results. The following assumptions will be made:

1. $\mathcal{M}$ is geodesically complete.
2. The algorithm remains in a compact region $\mathcal{U}$ of $\mathcal{M}$.

3. The function $f : \mathcal{M} \to \mathbb{R}$ is $C^1$ and, an initial point $p_0 \in \mathcal{M}$ is given. The gradient $\nabla f$ is Lipschitz in $\mathcal{U}$.

Assumption 1 was made in section 7 and, is for simplicity only, while assumptions 2 and 3 are as in [9]. All of the results in [9] carry over without modification to Algorithm 10.2 on Riemannian manifolds. The first result is

**Theorem 10.3** *[9]*

*The sequence of quasi-minimal points* $\{z_n\}$ *is infinite and* $\lim_{k\to\infty} \Delta_k^m = 0$.

Using Theorem 10.3, the convergence result for the frame based method is

**Theorem 10.4** *[9]*

*Every cluster point of the sequence of quasi-minimal points* $\{z_n\}$ *is a stationary point of* $f(q)$.

Proving any convergence results for the LTMADS algorithm in section 7 seems to be a much more challenging task. Again, this is because of the parallel transportation of tangent vectors that needs to be performed.

For an example of the frame based method, we will redo the hypersphere problem given by (9.6). We set $\beta = \delta = 10^{-8}$. Everything else was done exactly as it was for the LTMADS example. The results for the frame based method were the same as in the LTMADS example, see Figure 2.

# 11 Discussion

We've demonstrated that the Nelder-Mead simplex algorithm can be generalized to Riemannian manifolds. The main limitations are that the manifold in question has tractable formulas for the $\mathrm{Exp}$ and $\mathrm{Log}$ mappings. Many manifolds met in practice will have such formulas, two of which we examined in detail: $\mathcal{SO}(n)$ and $\mathcal{G}(n,k)$. The algorithm was shown to successfully converge on our test cases for $\mathcal{SO}(n)$ and $\mathcal{G}(n,k)$. These examples were chosen to demonstrate that the algorithm can be practically implemented and, because the true global solutions to the problems are known. The algorithm would really be of more practical use when the function to be minimized is non-differentiable or, maybe even non-continuous or noisy [21].

Regarding our restart procedure in subsection 5.1. The main difficulty, as previously stated, was that the algorithm could become 'stuck' in the region of the initial simplex simply because the simplex was constructed disadvantageously, not because the function had a local minimum there. A related problem is examined in [21]. Of course, our method for avoiding this problem would not necessarily be practical unless one already

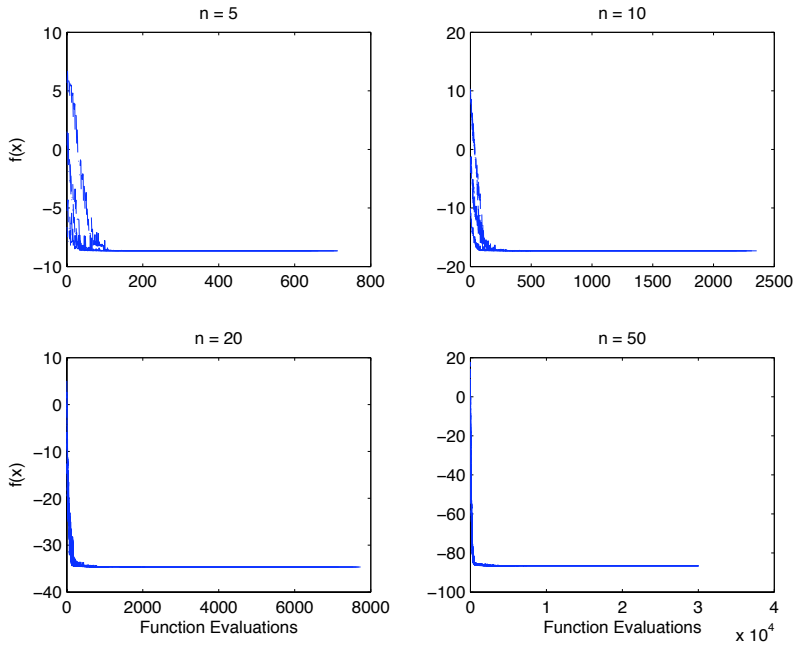


Figure 2: *The objective function value versus number of function evaluations for problem (9.6) using the frame based method.*

knew, or could guess, what the minimal value of the objective function was a priori. Other, more general methods for avoiding this situation can be imagined [24, 26].

The LTMADS algorithm has a more attractive generalization to Riemannian manifolds than the Nelder-Mead algorithm does. The reason for this is that the computations one needs to do in order to run the algorithm (finding geodesics with initial conditions and, parallel transportation) are relatively easy, at least in comparison to the computations needed for the Nelder-Mead algorithm's generalization. Also, LTMADS has convergence results [1, 3], something the Nelder-Mead algorithm generally lacks [18]. Whether the generalized LTMADS algorithm in section 7 also has some convergence properties still needs to be shown. We were able to give a convergence result for a related frame based method presented in section 10.

# 12 Acknowledgments

The author would like to thank Kevin R. Vixie (Los Alamos National Laboratory) and Los Alamos National Laboratory for providing the postdoctoral fellowship that allowed this research. The author would also like to thank Michael Kirby (Colorado

State University) and the Director's of Central Intelligence Postdoctoral Fellowship program for providing prior research funding, out of which this paper grew, and, the code to run the method in [8] for the Whitney algorithm. Additionally, appreciations go to Tom Asaki (Los Alamos National Laboratory) for suggesting the generalization of LTMADS and, John E. Dennis, JR (Rice University) for his helpful comments on earlier versions of this paper.

# References

[1] M. A. Abramson and C. Audet. Convergence of mesh adaptive direct search to second-order stationary points. *SIAM Journal on Optimization*, 17:606–619, 2006.

[2] P.-A. Absil, R. Mahoney, and R. Sepulchre. Riemannian geometry of Grassmann manifolds with a view on algorithmic computation. *Acta Applicandae Mathematicae*, 80:199–220, 2004.

[3] C. Audet and J. E. Dennis, JR. Mesh adaptive direct search algorithms for constrained optimization. *SIAM Journal on Optimization*, 17:188–217, 2006.

[4] M. S. Bazaraa, H. D. Sherali, and C. M. Shetty. *Nonlinear Programming: Theory and Algorithms*. Wiley, 2nd edition, 1993.

[5] E. Begelfor and M. Werman. Affine invariance revisited. *IEEE Computer Science Society Conference on Computer Vision and Pattern Recognition*, 2006.

[6] I. Bomze and G. Danninger. A global optimization algorithm for concave quadratic programming problems. *SIAM Journal on Optimization*, 3:826–842, 1993.

[7] D. Broomhead and M. Kirby. A new approach to dimensionality reduction: theory and applications. *SIAM Journal on Applied Mathematics*, 60:2114–2142, 2000.

[8] D. Broomhead and M. Kirby. Dimensionality reduction using secant-based projection methods: the induced dynamics in projected systems. *Nonlinear Dynamics*, 41:47–67, 2005.

[9] I. D. Coope and C. J. Price. Frame based methods for unconstrained optimization. *Journal of Optimization Theory and Applications*, 107:261–274, 2000.

[10] D. W. Dreisigmeyer and M. Kirby. A numerically implementable close-to-isometric embedding algorithm. *In preparation*.

[11] A. Edelman, T. A. Arias, and S. T. Smith. The geometry of algorithms with orthogonality constraints. *SIAM Journal on Matrix Analysis and Applications*, 20:303–353, 1998.

[12] G. H. Golub and C. F. van Loan. *Matrix Computations*. Johns Hopkins, 3rd edition, 1996.

[13] M. Günther. On the perturbation problem associated to isometric embeddings of Riemannian manifolds. *Annals of Global Analysis and Geometry*, 7:69–77, 1989.

[14] M. Günther. Isometric embeddings of Riemannian manifolds. In *Proceedings of the International Congress of Mathematicians, Kyoto, Japan, 1990*, pages 1137–1143, 1991.

[15] M. W. Hirsch. *Differential Topology*. Springer, 1976.

[16] H. Karcher. Riemannian center of mass and mollifier smoothing. *Communications on Pure and Applied Mathematics*, 30:509–541, 1977.

[17] W. S. Kendall. Probability, convexity and harmonic maps with small image I: Uniqueness and fine existence. *Proceedings of the London Mathematical Society*, 61:371–406, 1990.

[18] J.C. Lagarias, J. A. Reeds, M.H. Wright, and P. E. Wright. Convergence properties of the Nelder-Mead simplex method in low dimensions. *SIAM Journal on Optimization*, 9:112–147, 1998.

[19] J. H. Manton. On the various generalizations of optimization algorithms to manifolds. Invited paper at the *Sixteenth International Symposium on Mathematical Theory of Networks and Systems*, July, Katholieke Universiteit Leuven, Belgium, 2004.

[20] J. H. Manton. A globally convergent numerical algorithm for computing the center of mass on compact Lie groups. In *Eighth International Conference on Control, Automation, Robotics and Vision*, 2004.

[21] K. I. M. McKinnon. Convergence of the Nelder-Mead simplex method to a non-stationary point. *SIAM Journal on Optimization*, 9:148–158, 1998.

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

[23] P. J. Olver. *Applications of Lie Groups to Differential Equations*. Springer, 2nd edition, 1993.

[24] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vettering. *Numerical Recipes in C*. Cambridge University Press, 2nd edition, 1992.

[25] C. J. Price and I. D. Coope. Frames and grids in unconstrained and linearly constrained optimization: a nonsmooth approach. *SIAM Journal on Optimization*, 14:415–438, 2003.

[26] A. Prsa and T. Zwitter. Introducing adapted Nelder and Mead's downhill simplex method to a fully automated analysis of eclipsing binaries. In *Proceedings of the Gaia Symposium: The Three-Dimensional Universe with Gaia (ESA SP-576)*, 2004.

[27] I. U. Rahman, I. Drori, V. C. Stodden, D. L. Donoho, and P. Schroeder. Multiscale representations for manifold-valued data. *Multiscale Modeling and Simulation*, 4:1201–1232, 2005.

[28] M. Spivak. *A Comprehensive Introduction to Differential Geometry*, volume 1. Publish or Perish, 3rd edition, 2005.

[29] C. Udriste. *Convex Functions and Optimization Methods on Riemannian Manifolds*. Kluwer, 1994.

[30] Y.-C. Wong. Differential geometry of Grassmann manifolds. *Proceedings of the National Academy of Sciences (U.S.A.)*, 57:589–594, 1967.

[31] Y.-C. Wong. Sectional curvatures of Grassmann manifolds. *Proceedings of the National Academy of Sciences (U.S.A.)*, 60:75–79, 1968.

[32] R. P. Woods. Characterizing volume and surface deformations in an atlas framework: theory, applications and implementation. *NeuroImage*, 18:769–788, 2003.