A GJK-Based Algorithm for Constrained Least Squares

Paul Centore

Abstract

A least squares problem is of the form Mx=b, where M is a matrix, and x and b are vectors. M and b are known, and one seeks an x that minimizes the Euclidean norm of Mx-b. Ordinary least squares (OLS) places no restrictions on x, while constrained least squares (CLS) does. Typical approaches to CLS involve linear or quadratic programming. This paper proposes a new approach, incorporating the Gilbert-Johnson-Keerthi (GJK) algorithm, which finds the minimum distance between two convex polytopes. Mx is treated as a set, which depends both on M and on the constrained x. In many cases, Mx is a convex polytope P, for which a set of generators can be found; the generators are linear combinations of columns of M. The GJK algorithm is used to find the point p on P which is nearest to b. GJK is very fast; it needs generators for P, but does not calculate a convex hull. The GJK algorithm also expresses p as a convex combination of the generators. Since the generators are functions of M, one can work backwards to values for x. Example applications are given, involving constrained mixture designs (CMD), bounded variables least squares (BVLS), and non-negative least squares (NNLS).

1 Introduction

Least squares problems arise in many practical situations, particularly when fitting a linear model to a set of measured data. A least squares problem is of the form

$$Mx = b, \tag{1}$$

where M is a known matrix of m rows and n columns, and b is a known vector. Mx and b are assumed to exist in the same vector space, $\mathbf{R}^m$, which is equipped with a Euclidean metric; an orthonormal basis is chosen to simplify computations. One seeks a vector x that minimizes the norm of the vector $Mx - b$. In an ordinary least squares (OLS) problem, there are no restrictions on any of the entries in x. In a constrained least squares (CLS) problem, there are such restrictions.

Consider ordinary least squares first. Typically, there is no $\hat{x}$ such that $M\hat{x} = b$ exactly. A least squares solution $\hat{x}$ that minimizes the norm of $Mx - b$, however, does exist. Let v denote the vector of minimum norm:

$$v = M\hat{x} - b. \tag{2}$$

Geometrically, v is a vector that joins b to a point b_M in Mx, which, since x is unconstrained, is the range of M. The range of M is a subspace of $\mathbf{R}^m$, so it is convex. Since a convex set has a unique point that is nearest to an outside point,[1] both v and b_M must be unique. Furthermore,

$$M\hat{x} = b_M. \tag{3}$$

Equation (3) implies that b_M is in the range of M. While $\hat{x}$ is not an exact solution to $Mx = b$, it is an exact solution to $Mx = b_M$. Furthermore, any other solution to $Mx = b_M$ is also a least squares solution.

This line of reasoning suggests a geometric approach to least squares. Consider Mx as just a subset, rather than a subspace, of $\mathbf{R}^m$, and consider b as a target vector in $\mathbf{R}^m$ that M is trying to reach. Since b is not in Mx, the best we can do is to choose the point b_M that *is* in Mx, and is as close to b as possible. Any solution to the equation $Mx = b_M$ is then a least squares solution to $Mx = b$. Furthermore, at least one $\hat{x}$ satisfies $Mx = b_M$, and the set of all solutions to $Mx = b_M$ is the set $\hat{x} + \ker(M)$.

This geometric approach is fruitful because it applies to both ordinary and constrained least squares. In the preceding OLS derivation, the assumption of an unrestricted x was only needed to insure that Mx was a subspace of $\mathbf{R}^m$. When x is constrained, Mx is no longer a subspace, but is a subset S. If we can find a point b_S on S that is closest to b, the rest of the derivation follows through, and determines a least squares solution.

The general problem of finding the shortest distance from a point to an arbitrary subset is computationally intractable. When the subset is a convex polytope P, however, the Gilbert-Johnson-Keerthi (GJK) algorithm[2,3] can quickly find the nearest point on P to a target point b. The GJK algorithm requires P to be expressed as the convex hull of a known set of generating vectors. We will later see some important CLS examples in which such generator vectors can readily be calculated from M.

The set Mx can be seen as a linear combination of the columns of M:

$$Mx = \sum_{i=1}^{n} x_i M_i, \tag{4}$$

where M_i is the i^{th} column of M. This viewpoint turns constraints on x into restrictions on the coefficients of the M_i's. If there are no constraints, then Mx is just the column space of M, denoted $\text{col}(M)$. Constraints on x make Mx a proper subset of $\text{col}(M)$. Many common constraints, however, lead to tractable geometric forms for Mx. For example, if the constraints require that all entries are between 0 and 1, and also sum to 1, then Mx is the convex hull of the column vectors of M. Since Mx is a convex hull, it is also a convex polytope, and the GJK algorithm can be applied.

This paper's main idea is incorporating the constraints on x into the set Mx. In many practical cases, Mx is a convex polytope P, so the GJK algorithm can find b_P, the closest point to b on the constrained Mx. As a consequence, there exists a solution which not only satisfies $Mx = b_P$ exactly, and $Mx = b$ as closely as possible, but which also satisfies the constraints.

The GJK algorithm that calculates b_P is a fast, iterative method that finds the minimum distance between two convex polytopes. In this paper, the first polytope is the set Mx, and

the second polytope is the point b. The algorithm also expresses b_P as a convex combination of generators. The generators themselves are expressed as linear combinations of the columns of M. Coefficient expressions can be manipulated to produce entries in an x that not only solves $Mx = b_P$, but also satisfies the constraints. Since b_P minimizes the distance from Mx to b, such an x is in fact a constrained least squares solution.

This paper is organized as follows. First, the GJK algorithm is described. Second, the GJK-based algorithm is presented: a convex polytope P is constructed from M and the constraints on x, and the GJK algorithm finds the closest point on P to b; manipulating the GJK output produces entries of x that both satisfy the constraints and minimize the distance from b. Third, three practical examples (constrained mixture design (CMD), bounded variable least squares (BVLS), and non-negative least squares (NNLS)) are worked out in detail, leading to algorithms for those sets of constraints. Finally, the algorithm's possible benefits are discussed.

2 The GJK Algorithm

2.1 Algorithm Setting

The Gilbert-Johnson-Keerthi (GJK) algorithm[2] was introduced in 1988. The algorithm finds the shortest distance between two convex polytopes in a finite-dimensional Euclidean space. Besides its speed, the GJK algorithm is notable for the form of its inputs: it only requires two sets of points, whose convex hulls are those polytopes. This form occurs naturally in constrained least squares problems.

A convex polytope in $\mathbf{R}^m$ is most easily defined as the convex hull of a finite set of points. Formally, suppose $V = \{v_1, v_2, ..., v_k\}$ is a set of points in $\mathbf{R}^m$. Then $\text{co}(V)$, the convex hull of V, is given by

$$\text{co}(V) = \left\{ \sum_{i=1}^{k} \alpha_i v_i \,\middle|\, \sum_{i=1}^{k} \alpha_i = 1 \text{ and } 0 \leq \alpha_i \leq 1 \forall i \right\}. \tag{5}$$

Intuitively, convex polytopes generalize convex polygons and polyhedra to an arbitrary number of dimensions. Polytopes in three dimensions can be thought of as bodies with corners, sharp edges, and flat faces; in higher dimensions, a polytope's boundary is similarly a union of flat objects of lower dimension. The requirement that a polytope be generated by a finite set of points means that curved surfaces such as spheres cannot be polytopes. Polytopes are not required to be fully dimensional. For example, a line segment in $\mathbf{R}^3$ is a polytope, as is a triangle. Convex polytopes are always closed and bounded. The set of generating points for a convex polytope is not unique. A cube, for example, is a polytope, one of whose generating sets is its eight vertices; the union of the cube's vertices and the cube's center is again a generating set.

Though the algorithm can take two convex polytopes as input, CLS problems need only a special case: one of the polytopes is a target point b. The GJK algorithm then determines the minimum distance from the target point to the polytope, the point b_P on the polytope which is closest to the target point, and an expression for b_P as a convex linear combination of the generators, as in Equation (5).

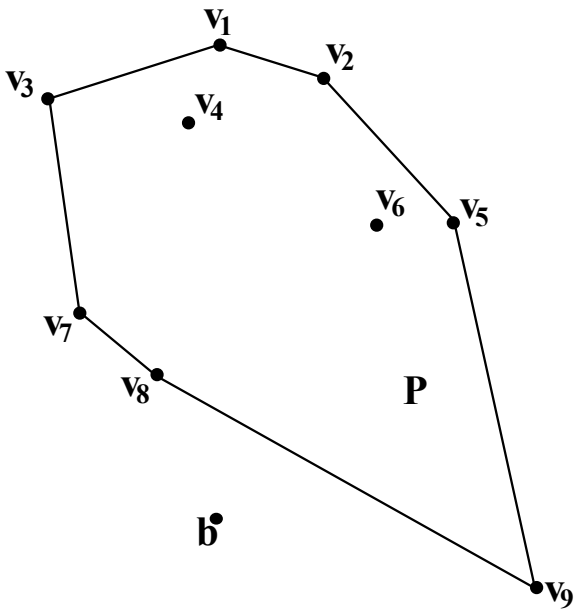

Figure 1: Two-Dimensional Example for GJK Algorithm

2.2 Description of the GJK Algorithm

The GJK algorithm will be explained with a two-dimensional example. Suppose that a convex polytope P in $\mathbf{R}^2$ is generated from a finite set V of points $v_i, i = 1, ..., k$, as shown in Figure 1. While all the extreme points, or corners, of P are points in V, there can also be points of V, such as v_4 and v_6, in the interior of P. The interior points are superfluous but harmless: the algorithm will reach the same result whether or not they are present. A target point b is also shown. The GJK algorithm finds the point b_P on P which is closest to b. The convexity of P guarantees that such a point exists and is unique (see Sect. 19.2 of Ref. 1). While b is often outside P, b can also be inside P, or on its boundary; in such cases the distance to P is 0, and the closest point b_P is just b itself.

An important concept for the GJK algorithm is the support function. The support function $\rho_{d,u}$ uses a unit direction vector d, which originates at a point u, as shown in Figure 2. Given any point v in the space, the support of v, relative to d and u, is defined as

$$\rho_{d,u}(v) = (v - u) \cdot d. \tag{6}$$

Geometrically, the vector d can be extended to a line, with the point u indexed by 0. Other points on this line are indexed by their distance from u, with positive points in the direction of d and negative points in the direction of $-d$. The point v can be projected perpendicularly onto this line. The index of the projection point is then the support of v, relative to d and u. The GJK algorithm uses the support function to move across P in certain directions, approximately towards b. When the algorithm has converged, the generating points that make a non-zero contribution to b_P will all have minimum support, and no further reduction is possible.

The GJK algorithm begins by choosing a subset W of V. W can contain up to $m + 1$ points, where m is the dimension of the space. The points should be affinely independent so

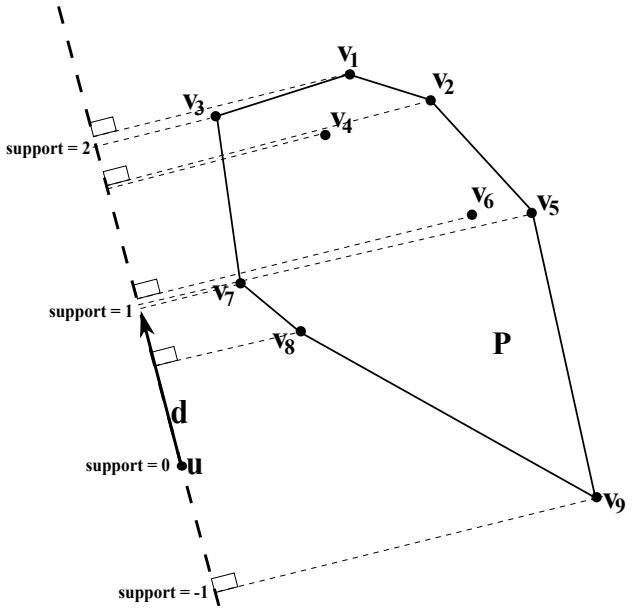

Figure 2: The Support Function

that they generate a simplex. Apart from these restrictions, the choice of W is arbitrary. In two dimensions, W will be either a single point, two points which define a line segment, or three points which define a triangle. For simplicity, W can be chosen to be a single point. For purposes of illustration, we will start with the points v_3, v_6, and v_8, shown in Figure 3, which generate a triangle.

Once W has been chosen, the convex hull $\text{co}(W)$ of its elements forms a simplex, such as the triangle in Figure 3. At this point, Johnson's sub-distance algorithm[3] can be used to determine the point b_W on $\text{co}(W)$ that is closest to b, as well as the minimal subsimplex of $\text{co}(W)$ that contains b_W. In Figure 3, for example, the point b_W is on the line segment generated by v_8 and v_6. That line segment is the minimal subsimplex. Since v_3 is not needed to generate this subsimplex, discard v_3 from W. Johnson's algorithm returns not just the minimal subsimplex, but also an expression for b_W as a convex combination of the polytope's generators. In the figure, for example, $b_W = 0.85v_8 + 0.15v_6$.

For the next step, Figure 4 shows how the support function expands the minimal W into a new simplex that contains points that are closer to b. Construct the unit vector d that originates at b and points toward b_w. Calculate the support for each point in V, relative to d and b. Choose a point v_i of minimal support, and add v_i to the set W. It is possible for the support to be negative, and a point with negative support should be chosen over a point with positive support. In Figure 4, the point v_9 has minimal support, so it was added to the new W that is shown there.

This step has some subtleties. The union of W and the new point might not be affinely independent, so no simplex would be formed. In that case, one must delete further points from W, try a different new point, or adjust one of the points very slightly. This problem is expected in some realistic cases. Suppose that the polytope is a cube in $\mathbf{R}^3$, and that its closest point is on a face of the cube. Then the vertices of that face all have the same

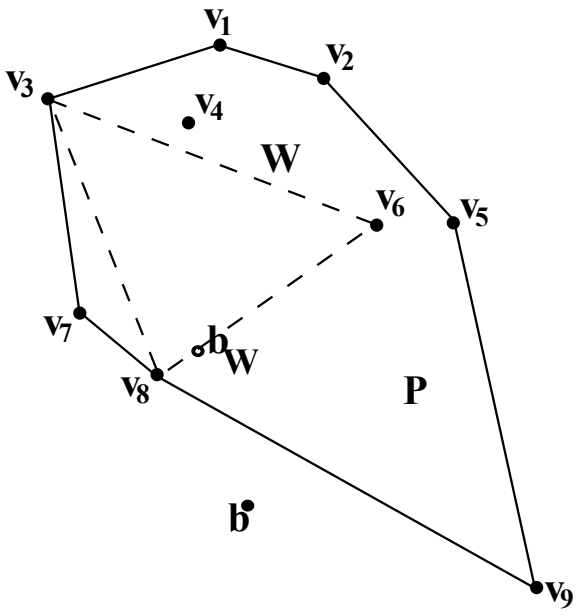

Figure 3: Choosing a Simplex in the Polytope P

support with respect to the minimum line segment between the target and the cube. As the algorithm converges, it is easy to see how it could try to construct a simplex from those four vertices—the attempt will fail because a four-point simplex should have dimension 3 rather than dimension 2. If one of those vertices is adjusted very slightly, then the face will go from a square to two triangles. The adjustments should have negligible effect on the closest point, but will permit the algorithm to form simplices.

Now iterate over the previous two steps:

1. Find the nearest point b_W on the simplex generated by W, and reduce W to the minimal containing subsimplex.
2. Find a point whose support, with respect to the vector from b towards b_W, is as small as possible, and that still makes a simplex when added to W. Add that point to W.

These two steps will produce a series of points b_W, that terminate in the desired b_P. The algorithm terminates when all points of minimal support are already contained in the current W. Figure 5 shows the termination for the example, which occurs after the second set W has been constructed. The point b_P is contained in the minimal subsimplex generated by v_8 and v_9. The support function of $b_P - b$ takes on a minimum only at v_8 and v_9, which are already in the latest W. Geometrically, the vector from b to b_P is perpendicular to an edge of P, so both vertices for that edge will have the same support. In three dimensions, the vector from b to b_P could be perpendicular to a bounding face, all of whose vertices have the same minimal support. Analogues hold for higher dimensions.

The GJK algorithm is very fast. The reason for its speed is that generally only a small number of simplices are needed, and the simplices themselves require only a few vertices, often less than half a dozen. (The slowest part of the algorithm is evaluating the support function.) This speed, however, requires that the polytope of interest be expressed by a set of generating vectors. Although the polytope is implicitly a convex hull of its generators,

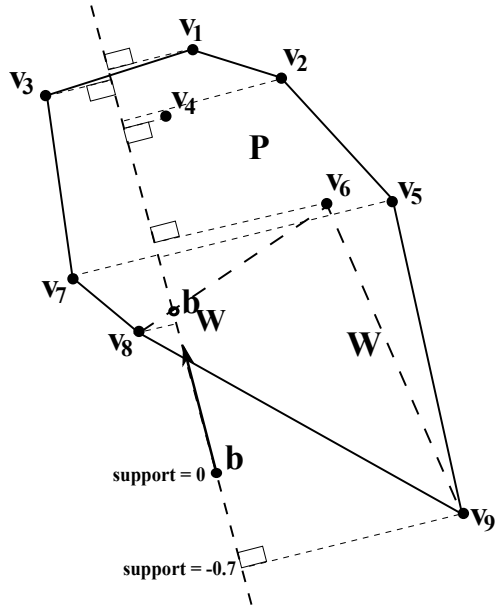

Figure 4: Choosing a New Simplex in the Polytope P

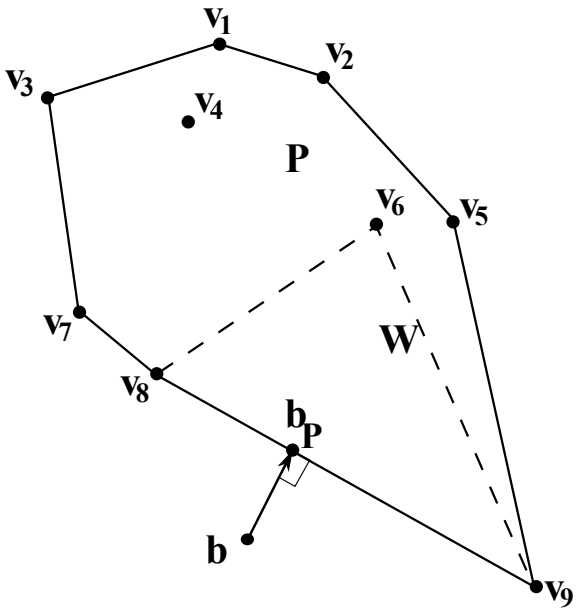

Figure 5: Termination of the GJK Algorithm

a surprising GJK feature is that there is no need to calculate the convex hull directly. This features saves time, too, because finding convex hulls can be slow, especially in high dimensions.

Convex polytopes are also commonly defined as the intersection of half-spaces resulting from linear inequalities; this form appears in linear and quadratic programming (QP). In fact, since the distance from a given point is a quadratic function, the problem of finding the nearest point on a polytope comes up regularly in QP. While the GJK algorithm looks tempting in this setting, it is likely no more efficient than standard QP methods—the GJK algorithm requires converting the half-spaces into a set of generators for the polytope, which can be very time-consuming. Fortuitously, the GJK algorithm seems tailor-made for the constrained least squares problems that are the concern in this paper.

3 A GJK-Based Algorithm for Constrained Least Squares

The GJK algorithm is surprisingly useful in some common CLS problems. In this section, we develop a GJK-based algorithm for such problems.

3.1 Constrained Least Squares Problems

A least squares problem involves a linear system

$$Mx = b, \tag{7}$$

where M is a matrix, and x and b are column vectors. M and b are known, and x is to be solved for. Because of measurement or model error in practical situations, it is likely that no x satisfies $Mx = b$ exactly. Instead, one finds an $\hat{x}$, called a least squares (LS) solution, that satisfies $Mx = b$ as closely as possible. Closeness is measured by the squared norm of the difference vector between $M\hat{x}$ and b:

$$\text{squared norm} = \sum_{i=1}^{m} \left((M\hat{x})_i - b_i\right)^2, \tag{8}$$

where the subscript i denotes the i^{th} coordinate. (We are assuming here a Euclidean metric; for simplicity, vectors are expressed in an orthonormal basis.) When there are no restrictions on x, minimizing Equation (8) is the ordinary, or unconstrained, least squares (OLS) problem.

The constrained least squares problem is to minimize (8) when there are restrictions, or constraints, on the coordinates of x. Two important CLS problems are non-negative least squares (NNLS), in which every coordinate must be positive or zero, and bounded variables least squares (BVLS), in which the i^{th} coordinate must satisfy $\gamma_i \leq x_i \leq \delta_i$. A third example is constrained mixture design (CMD), in which the i^{th} coordinate must satisfy $0 \leq x_i \leq 1$, and all the entries must sum to 1. CMD problems arise naturally when estimating the concentrations of ingredients in a mixture. In general, many other kinds of constraints are possible.

Stable and efficient procedures[4] are well established for ordinary least squares. Constrained least squares problems are considerably more challenging. NNLS and BVLS solutions are often recast as problems in linear or quadratic programming, which can be computationally demanding. The following sections formulate some constrained least squares

problems geometrically, viewing Mx as a convex polytope. In this formulation, the GJK algorithm has the potential of offering a faster and more intuitive procedure.

3.2 Geometric Interpretation of $Mx = b$

Suppose the matrix M has m rows and n columns. Then M can be viewed as a linear transformation from the vector space $\mathbf{R}^n$ to the vector space $\mathbf{R}^m$. The vector b on the right side of Equation (7) is a target vector in $\mathbf{R}^m$. The left side, Mx, can be seen as a set of linear combinations of the columns of M. In fact, each column M_i of M can be seen as a vector in $\mathbf{R}^m$: M_i is the image under M of the vector x which is zero in all coordinates except the i^{th}, where it takes on the value 1. If the variables in x are unconstrained, then Mx is just $\text{col}(M)$. While $\text{col}(M)$ has considerable structure, for now it is best just to view it as a subset S of $\mathbf{R}^m$.

S will change if constraints are placed on x. NNLS, for example, assumes that the entries of x are non-negative, so Mx is the convex cone generated by the columns of M. For BVLS, we will see later that Mx is the zonotope generated by the columns of M. The three subsets in these examples have a strict containment relationship:

$$\text{subspace} \supset \text{convex cone} \supset \text{zonotope}. \tag{9}$$

The variety of forms resulting from different constraints suggests that algorithms might be adapted to certain sets of constraints.

When Mx is seen as a set S in $\mathbf{R}^m$, the equation $Mx = b$, where b is also in $\mathbf{R}^m$, becomes a set containment problem: is the vector b in the set S? Since we are assuming that there is no x exactly satisfying $Mx = b$, it must be that b is not contained in S. Given that failure, the least squares approach to $Mx = b$ finds a point b_S that *is* in S, and is as close to b as possible. Then the equation $Mx = b_S$ will have an exact solution, because b_S is in the range of M. Furthermore, that solution can be expressed so as to satisfy constraints on x, because those constraints were incorporated into the set Mx. In fact, this method is used implicitly in OLS. The set Mx is the subspace $\text{col}(M)$, and b_S is the orthogonal projection of b onto that subspace.

Containment relationships like (9) show why constraints cause difficulty in the first place. A point b_S might be the closest point to b on the subspace $\text{col}(M)$, and thus be helpful in the ordinary least squares problem—but likely b_S is not contained in the BVLS zonotope. Solving the BVLS problem requires the closest point to b on the zonotope.

GJK can be helpful in least squares by quickly finding closest points, when Mx is a convex polytope. An important observation is that, for many CLS cases in practice, the set Mx, under the constraints imposed on x, *is* a convex polytope P. Apply the GJK algorithm to P to find the point b_P on P, which is closest to b. Then the equation $Mx = b_P$ has an exact solution, because b_P was chosen to belong to the set Mx.

3.3 A Set of Generators for Mx

One limitation of the GJK algorithm—which turns out to be a strength for CLS problems—is that it requires a set of generators for the convex polytope P. In many CLS problems, the

generators are linear combinations of the columns of M, so they can be computed quickly and easily. The choice of linear combinations is adapted to the constraints. A later section will give some concrete examples. This section will assume that the generators are linear combinations of the columns of M, and calculate a potential CLS solution x.

Denote the set of generators for P by V, and assume that every element of V is a linear combination of the the column vectors of M. A convenient notation for V uses an index matrix J, that gives the coefficients of M_i in the generators. If n were 3, for example, and the coefficients could be any combination of 0's and 1's, then we could write

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \\ v_8 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} M_1 \\ M_2 \\ M_3 \end{bmatrix}, \tag{10}$$

where each v_j is an element of V, and J is the matrix of 0's and 1's. Each M_i has a coordinate expression in $\mathbf{R}^m$, so each v_j also has a coordinate expression in $\mathbf{R}^m$. This example has 2^n generators; other examples would have different numbers of generators.

The v's in V generate the convex polytope P, and the GJK algorithm can find the point b_P on P that is nearest to b. The algorithm returns its results in the form

$$\begin{aligned} b_P &= \sum_{j=1}^{2^n} \alpha_j v_j & (11) \\ &= \alpha_i v_i + \alpha_2 v_2 + ... + \alpha_{2^n} v_{2^n}, & (12) \end{aligned}$$

where b_P and the v_i's are vectors in $\mathbf{R}^m$, and the α_i's are scalars. Substituting Equation (10) into Equation (12) gives

$$\begin{aligned} b_P = \quad & \alpha_1 \left(J_{1,1} M_1 + J_{1,2} M_2 + ... + J_{1,n} M_n \right) + ... \\ & + \alpha_2 \left(J_{2,1} M_1 + J_{2,2} M_2 + ... + J_{2,n} M_n \right) + ... \\ & \qquad + ... + ... \\ & + \alpha_{2^n} \left(J_{2^n,1} M_1 + J_{2^n,2} M_2 + ... + J_{2^n,n} M_n \right) & (13) \\ = \quad & \left(\alpha_1 J_{1,1} + \alpha_2 J_{2,1} + ... + \alpha_{2^n} J_{2^n,1} \right) M_1 + ... \\ & + \left(\alpha_1 J_{1,2} + \alpha_2 J_{2,2} + ... + \alpha_{2^n} J_{2^n,2} \right) M_2 + ... \\ & \qquad + ... + ... \\ & + \left(\alpha_1 J_{1,n} + \alpha_2 J_{2,n} + ... + \alpha_{2^n} J_{2^n,n} \right) M_n. & (14) \end{aligned}$$

By construction, b_P is the nearest point to b that is also in Mx. Therefore b_P will replace b in Equation (7), giving

$$b_P = x_1 M_1 + x_2 M_2 + ... + x_n M_n, \tag{15}$$

where b_P and the M_i's are vectors, and the x_i's are scalars. Equations (14) and (15) are two equal linear combinations of the column vectors of M, so we can find a solution (possibly not unique) by equating coefficients. If $\alpha = [\alpha_1, \alpha_2, ...]$ is written as a row vector, then we get

$$x = \alpha J. \tag{16}$$

We have now found an explicit x such that $Mx = b_P$. The next section deals with the implications for CLS solutions.

3.4 Validity and Uniqueness

While it would seem that αJ is a CLS solution, in fact that conclusion requires more work. The difficulty is that there can be multiple solutions to $Mx = b_P$, some of which satisfy the constraints and others of which do not. An example is the following system, when every x_i is constrained to be between 0 and 1:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix} \begin{bmatrix} C_1 \\ C_2 \\ C_3 \end{bmatrix} = \begin{bmatrix} 3 \\ 6 \\ 9 \\ 12 \end{bmatrix}. \tag{17}$$

The vector $y = [0, 0, 1]$ satisfies Equation (17) and also satisfies the constraints. The vector $z = [-1/6, 1/3, 5/6]$, however, also satisfies Equation (17), but does not satisfy the constraints. We need some way of insuring that $x = \alpha J$ really is a valid solution.

Several methods are available. If the equation $Mx = b_P$ has a unique solution $\hat{x}$, then, since there exists one solution that satisfies the constraints, that solution must be $\hat{x}$, so we have a valid CLS solution. Uniqueness can be inferred either from physical considerations, or from linear algebra: if the columns of M are linearly independent, so M has full column rank, then $\hat{x}$, which is αJ in this case, is unique. Even if M does not have full column rank, information can still be inferred about αJ. For example, α is the set of coefficients of a convex combination of vertices. Therefore every α_i is between 0 and 1, and the sum of all the α_i's is 1. The entries of J will be known explicitly, which can place further bounds on αJ. In the cases of constrained mixture design and bounded variables least squares, we will see that such conditions imply that $x = \alpha J$ must satisfy the constraints. In any particular case, of course, αJ can always be tested directly to see if it satisfies the constraints; if it does, then we have a CLS solution, regardless of theoretical considerations.

If all these methods fail, then a more difficult set intersection problem can be solved. The set of exact solutions to $Mx = b_P$ is given by the affine subspace $K = \hat{x} + \ker(M)$. One can also consider the subset C of $\mathbf{R}^n$ that contains all the x's that satisfy the constraints. If every x_i were required to be between 0 and 1, for example, then C would be the unit cube. $K \cap C$ is the set of all least squares solutions that satisfy the constraints. K is an affine subspace and C is usually a convex set, so their intersection can be calculated. In addition, one might require only one CLS solution, i.e. only one point in the intersection, rather than the entire intersection; this requirement would simplify calculations.

Solution uniqueness is another possible concern: multiple minimizing solutions might satisfy the constraints. One reason is that different α's can produce the same nearest point

b_P on P. Geometrically, b_P results from different convex combinations of the generators. A practical example will be presented when we consider constrained mixture design. In addition, different J's will produce different generating sets for Mx, which could affect the calculation αJ.

In general, rigorous answers to questions of validity and uniqueness will be specific to a particular CLS problem. The practical examples to be presented, however, show that they are not hard to come by. Further investigations might determine how serious the question of validity is in practice.

3.5 Summary of GJK-Based Algorithm

The following steps summarize the GJK-based algorithm for a constrained least squares problem of the form $Mx = b$:

1. Determine whether Mx, given the constraints on x, is a convex polytope P. (If not, the GJK-based algorithm cannot be used, so exit.)
2. Construct P as the convex hull of a set of linear combinations of the columns of M. Express this set in an index matrix J. (If no such set exists, then exit.)
3. Construct V, a generating set for P, from the matrix J.
4. Apply the GJK algorithm to V to find b_P, the point on P that is closest to b. The GJK algorithm also produces a row vector α, that gives b_P as a convex combination of the elements in V.
5. Calculate $x = \alpha J$.
6. Analyze the validity and uniqueness of αJ:

    (a) If possible, prove analytically that αJ must satisfy the desired constraints, or
    (b) Test αJ to see whether it satisfies the constraints.
    (c) If either of Steps 6a and 6b succeeds, then αJ is a least squares solution that satisfies the constraints.
    (d) If Steps 6a and 6b both fail, then find a vector p in the intersection of the set $K = \alpha J + \ker(M)$ with the set C of all x's that satisfy the constraints. The vector p will then minimize the norm of $Mx - b$, while still satisfying the constraints.

4 Three Examples

The following sections apply the GJK-based algorithm to some common CLS cases. In all three cases, the GJK-based algorithm provides a simple solution that can be proven to satisfy the constraints.

4.1 Constrained Mixture Designs

The constrained mixture design (CMD) problem is to find a least squares solution to $Mx = b$, under the constraints that

$$0 \leq x_i \leq 1, \tag{18}$$

$$\sum_{i=1}^{n} x_i = 1. \tag{19}$$

CMD problems arise when multiple ingredients are combined into one mixture. (In fact, this paper was motivated by such a problem, involving the Kubelka-Munk paint-mixing model.) The variable x_i is the concentration of the i^{th} ingredient in the mixture. The concentrations can be expressed as percentages of the entire mixture. For example, a mixture might be 50% ingredient 1, 40% ingredient 2, and 10% ingredient 3. Physically, concentrations must be 0 and 100% and must sum to 100%, leading to the constraints.

The set Mx is simply characterized under these constraints. A point in Mx is of the form

$$\sum_i x_i M_i, \tag{20}$$

where the x_i's are coefficients in a linear combination and satisfy (18) and (19). Geometrically, these constraints define the set of convex combinations of the vectors M_i. The set Mx is therefore a convex polytope, and a generating set V is just the column vectors of M. The index matrix J is then just the identity matrix. Apply the GJK algorithm to the generating set V to find the point b_P on P that is nearest to b, and also a vector α such that

$$b_P = \sum_i \alpha_i v_i. \tag{21}$$

Since J is the identity matrix, $v_i = M_i$, and Equation (21) becomes

$$b_P = \sum_i \alpha_i M_i. \tag{22}$$

Either from Equation (22) or by calculating $x = \alpha J$, we see that setting $x_i = \alpha_i$ guarantees that the variables x_i minimize the norm of $Mx - b$.

We now investigate whether those x_i's also satisfy the constraints. In this case, it is clear that they do. Since α_i are the coefficients of a convex combination produced by the GJK algorithm, α_i must be between 0 and 1, and all the α_i's must sum to 1. These are exactly the conditions in (18) and (19), so the x_i's, which just equal the α_i's, must satisfy the desired constraints. We have therefore found a general, easily calculated, solution for the CMD problem.

Though a valid solution has been found, it might not be unique. While the GJK algorithm has found one vector α that satisfies Equation (21), there could be another vector β that also satisfies that equation. For a practical example, let the x_i's represent the concentrations of a set of paints in a mixture. Suppose we are trying to find a mixture of paints whose

colour most closely matches the colour of a target paint b. Then α might tell us that the closest match is given by a mixture of 80% of Paint 1 and 20% of Paint 2, and β might tell us that the closest match is given by a mixture of 60% of Paint 3 and 40% of Paint 4. The physical interpretation is that these two mixtures produce the same colour, and that that colour most closely matches the target colour.

4.2 Bounded Variables Least Squares

The bounded variables least squares (BVLS) problem is to find a least squares solution to $Mx = b$, under the constraints that $\gamma_i \leq x_i \leq \delta_i$. By making the transformation

$$\bar{x}, = \frac{x_i - \gamma_i}{\delta_i - \gamma_i}, \tag{23}$$

the constraints can be simplified to $0 \leq x_i \leq 1$. An element p of Mx can be written as a linear combination of the columns of M:

$$p = \sum_{i=1}^{n} x_i M_i. \tag{24}$$

From this point of view, M_i's are vectors in $\mathbf{R}^m$, and the variables x_i are coefficients in a linear combination of those vectors. Since x_i is between 0 and 1, the set $x_i M_i$, for a fixed i, is geometrically the line segment that joins the origin to M_i. Write $p_i = x_i M_i$. Denote this line segment by σ_i. Then p_i is a point on σ_i, and

$$p = p_1 + p_2 + ... + p_n. \tag{25}$$

This kind of sum, in which each vector is an element of a different set, is an example of the *Minkowski* or *vector sum*, here denoted $\oplus$. The Minkowski sum of two subsets $\mathbf{A}$ and $\mathbf{B}$ of a vector space is defined by

$$\mathbf{A} \oplus \mathbf{B} = \{a + b \big| a \in \mathbf{A}, b \in \mathbf{B}\}. \tag{26}$$

The Minkowski sum naturally generalizes to any number of sets. Minkowski addition is commutative: $\mathbf{A} \oplus \mathbf{B}$ is the same set as $\mathbf{B} \oplus \mathbf{A}$. It is also associative: $(\mathbf{A} \oplus \mathbf{B}) \oplus \mathbf{C}$ and $\mathbf{A} \oplus (\mathbf{B} \oplus \mathbf{C})$ are the same set, so that we can unambiguously write $\mathbf{A} \oplus \mathbf{B} \oplus \mathbf{C}$. Geometrically, the Minkowski sum of $\mathbf{A}$ and $\mathbf{B}$ is the set covered by all the copies of $\mathbf{A}$ whose centers touch $\mathbf{B}$ at at least one point. The Minkowski sum of a line segment and a triangle (in two different planes), for example, would be a prism.

The definition of Minkowki sum allows us to write

$$Mx = \sigma_1 \oplus \sigma_2 \oplus ... \oplus \sigma_n. \tag{27}$$

This relationship is important because all the summands on the right are convex polytopes, and the Minkowski sum of convex polytopes is again a convex polytope.[6] Furthermore, an extreme point of a Minkowski sum must be the sum of extreme points of the summands. We can therefore create a set V that contains all Mx's extreme points: let V contain all

possible sums of extreme points for individual σ_i's, where each σ_i contributes one extreme point. Some elements of V will not be extreme points of Mx, but every extreme point of Mx will be in V. Since a convex polytope is generated by its extreme points, we can use V as a set of generators for Mx.

Since a line segment is a convex polytope, Mx in Equation (27) is the Minkowski sum of a set of convex polytopes, and is therefore itself a convex polytope. A Minkowski sum of line segments, each of which starts at the origin, is called a *zonotope*.[5] The extreme points, or vertices, of this zonotope are sums of the extreme points of the line segments that generate the zonotope. The i^{th} line segment has two extreme points: 0 and M_i. Formally, let E_i denote these extreme points. Then $E_i = \{0, M_i\}$, and

$$V = \sum_{i=1}^{n} e_i, \text{ such that } e_i \in E_i \ \forall i. \tag{28}$$

Since E_i contains two elements, V contains 2^n elements. In the context of zonotopes, each element of V is called a *node*. While every vertex of the zonotope is a node,[5] not every node is a vertex. Likely, some of the nodes will be in the interior of the zonotope. Nevertheless, since V contains all the vertices of the zonotope P, V is a generating set for P. The interior nodes, like the points v_4 and v_6 in Figure 1, are superfluous but harmless.

The index matrix J assigns a coefficient of 0 or 1 to each M_i, in all possible combinations. Each row of J is therefore a sequence of 0's and 1's, and there are 2^n rows, one for each possible sequence. Equation (10) gives an example of J, when there are three column vectors. J thus defines a set V of 2^n vectors that generate the zonotope. Applying the GJK algorithm to V produces a point on the zonotope that is closest to b, and a vector α that expresses that point as a convex combination of generators.

We calculate

$$x = \alpha J, \tag{29}$$

and investigate whether it satisfies the constraints. Since α gives the coefficients of a convex combination, all its entries are bounded by 0 and 1. From the node construction, all the entries of J are also bounded by 0 and 1. Each entry x_i is the dot product of α and the i^{th} column of J. Since all entries of α and J are non-negative, any such dot product is non-negative. The maximum of any such dot product occurs when every element in that column of J is 1. Since the entries of α sum to 1, any dot product of α and a column of 1's would itself be 1, which is the maximum possible value. Therefore every x_i satisfies the constraint of being between 0 and 1.

4.3 Non-Negative Least Squares

The non-negative least squares (NNLS) problem is to find a least squares solution to $Mx = b$, under the constraint that each x_i is non-negative. As before, begin by investigating the geometric form of Mx, given the constraints. When all the coefficients x_i in the sums $\sum x_i M_i$ are zero or greater, then Mx is the convex cone generated by the column vectors of M. A convex cone consists of rays that start at the origin. Let R_i denote the ray that starts at the origin and goes through M_i. Then Mx is the convex hull of all the rays R_i.

The first step of the GJK-based algorithm is to determine whether Mx is a convex polytope. In the NNLS case, Mx is unbounded, so it cannot be a polytope. We will adapt the algorithm, however, by imposing a sequence of artificial constraints on Mx. We can use the fact that the convex cone generated by $\{M_i, M_2, ..., M_n\}$ is identical to the convex cone generated by $\{\gamma_1 M_i, \gamma_2 M_2, ..., \gamma_n M_n\}$, where each γ_i is positive. We can therefore multiply each M_i by a γ_i such that the norm of $\gamma_i M_i$ is some Γ.

For a particular Γ, construct

$$\Gamma_{\text{CH}} = \text{co}\left(\{0, \gamma_1 M_i, \gamma_2 M_2, ..., \gamma_n M_n\}\right). \tag{30}$$

Γ_{CH} is the convex hull of the origin and the adjusted column vectors. Γ_{CH} is contained in the convex cone Mx. In fact, Γ_{CH} fits perfectly inside Mx at the origin, because Γ_{CH} simply truncates each ray in Mx. As Γ increases, Γ_{CH} occupies more and more of the convex cone.

The boundary of Γ_{CH} can be thought of as consisting of two parts: a conical part and a cap. The conical part consists of the truncated rays that Γ_{CH} shares with Mx, and that are on the boundary of Mx. The cap is all the other boundary points, and makes a sort of faceted base, most likely irregular, for the truncated cone. Since the conical part of the boundary is shared in common with Mx, the behavior in terms of nearest points is identical: if Γ is large enough, and the nearest point b_Γ to b on Γ_{CH} occurs on its conical boundary, then b_Γ is also the nearest point to b on Mx.

While the convex cone Mx is not a polytope, each Γ_{CH} is a polytope, with a generating set V given by the set in Equation (30). We can thus use the GJK algorithm to find b_Γ, the closest point to b on Γ_{CH}. We would like, of course, to find b_M, the closest point on Mx to b. When Γ is large enough that b_Γ occurs on its conical boundary rather than on its cap, then b_Γ will also be the nearest point to b on Mx.

A suggested approach to NNLS, then, is to apply the GJK algorithm to Γ_{CH}, for a sequence of Γ's that increase by a factor of 10. For example, the Γ's could be 1, 10, 100, etc. The result will be a sequence of points b_Γ. At some point, the sequence b_Γ will settle on a common point, no matter how much Γ is increased. Geometrically, b_Γ will have reached the conical boundary of Γ_{CH}, which agrees with the boundary of Mx when Γ is large enough. (There is actually another possibility: that b is in the interior of some Γ_{CH}. This possibility will be easily recognized, however, when the GJK distance is 0.) Since the GJK algorithm returns the coefficients α in a convex combination of the generators in V, and since V is just scaled versions of the columns of M, augmented by the zero vector, it is a straightforward calculation to obtain x from α.

This algorithm is not guaranteed to converge, and is still subject to the same kinds of instability that affect other NNLS algorithms. Suppose, for example, that

$$M_1 = [0, 1], \tag{31}$$
$$M_2 = [1, -\delta], \tag{32}$$
$$b = [1, 1], \tag{33}$$

where δ is positive. The point b is always in the convex cone generated by M_1 and M_2, because

$$(\delta + 1) \cdot [0, 1] + 1 \cdot [1, -\delta] = [1, 1]. \tag{34}$$

The angle between M_1 and M_2 is $\pi/2 + \arctan(\delta)$. If δ becomes very large, then the angle is nearly π. Then the coordinate $x_1 = \delta + 1$ in the solution to $Mx = b$ also become very large, even though b is small. Geometrically, any Γ_{CH} is a very narrow, almost flat, triangle. This situation invites numerical instability, which can degrade many algorithms.

In more stable cases, however, the GJK-based algorithm should be simple and effective. A handful of applications of the GJK algorithm should cover half a dozen orders of magnitude of variability in x, at a modest computational cost.

5 Discussion

The chief innovation of the approachs in this paper are incorporating constraints geometrically in the set Mx, and using the GJK algorithm for finding the closest point on Mx to the target vector b. The form of the problems themselves motivated the GJK algorithm: some practical examples produced convex polytopes with an obvious set of extreme points. Other benefits, and potential benefits, were discovered later, which suggested that the GJK approach was well suited for constrained least squares problems.

One likely benefit is speed. Quadratic programming is used in problems such as BVLS. The distance-squared function from b is a quadratic function on the zonotope P, so quadratic programming can be used to find the nearest point b_P. Quadratic programming methods, however, can be slow, while the GJK algorithm is fast. While speed differences have not yet been tested and quantified, the use of GJK in real-time applications such as robotics, where computing time is at a premium, suggests that GJK performs well. Other factors could make a difference, of course. For example, some algorithms behave differently in high-dimensional spaces, or when polytopes have very large generating sets. The GJK-based algorithm was tested on some examples in 31-dimensional space, though with only a handful of generators. The algorithm had no difficulty, but no speed comparison was made to other methods. While GJK-based algorithms seem to have the potential for faster performance than current methods, controlled comparisons would be needed to make any definite statements.

Another desirable characteristic of the GJK-based algorithm is its natural fit with the formats encountered in least squares problems. As the examples have shown, a generating set for Mx, that incorporates constraints, is usually easily constructed, as linear combinations of the column vectors of M. This small bit of computation provides enough input for the GJK algorithm. By contrast, a convex polytope for programming problems must usually be expressed as an intersection of half-spaces, where each half-space corresponds to a constraint stated as a linear inequality. Any convex polytope can be expressed in either form, but translating from one to the other can be a significant computational burden. Using quadratic programming methods on a zonotope would require finding all its bounding faces, or, equivalently, its convex hull. Even if quadratic programming were faster, any gain could easily be overbalanced by the convex hull calculation. Of course, if the polytope was naturally formulated as an intersection of half-spaces, then the expense of converting to a generating set could overbalance the gain from using the GJK algorithm. In the examples considered in this paper, however, the generating set formulation is clearly more natural, and allows distance-minimizing algorithms to be used immediately, with minimal data preparation.

Lastly, the GJK approach is intuitive and constructive. As the examples have shown,

often a practitioner can, with a bit of analysis, find a simple way to build his own polytope. The understanding gained by this analysis is helpful in its own right, and might suggest further ways of solving particular problems.

This paper presents the GJK-based algorithm as a promising suggestion, which would benefit from further investigation. Any standard methods have undergone long periods of revision and refinement, and undoubtedly the GJK approach could be considerably improved. Also, speed tests and comparisons with other methods would show whether the GJK approach offers any benefits, and if so, how significant they are and what situations they occur in. Along these lines, the author has released some open-source Octave/MATLAB code that implements some of the procedures discussed in this paper. They appear in the folder "GJK," in the author's Munsell And Kubelka-Munk Toolbox, available at

http://www.99main.com/~centore/MunsellAndKubelkaMunkToolbox.

Other researchers are encouraged to use, test, correct, and improve this code, with the understanding that they likewise make any changes publicly available.

References

1. S. R. Lay, *Convex Sets and Their Applications*, Dover Publications, 2007.
2. Elmer G. Gilbert, Daniel W. Johnson, & Sathiya Keerthi, "A Fast Procedure for Computing the Distance Between Complex Objects in Three-Dimensional Space," IEEE Journal of Robotics and Automation, Vol. 4, No. 2, April 1988, pp. 193-203.
3. Rich Rabbitz, "Fast Collision Detection of Moving Convex Polyhedra," in Section I.8 of *Graphics Gems IV (IBM version)*, ed. Paul Heckbert, Academic Press, 1994.
4. William H. Press, Saul A. Teukolsky, William T. Vetterling, & Brian P. Flannery. *Numerical Recipes: The Art of Scientific Computing, 3rd ed.*, Cambridge University Press, 2007.
5. P. McMullen, "On Zonotopes," *Transactions of the American Mathematical Society*, Vol. 159, pp. 91-109, September 1971.
6. R. V. Benson, *Euclidean Geometry and Convexity*, McGraw-Hill Book Company, New York, 1966.

