

An improved bisection method in two dimensions

Christopher Martin^{a,1,*}, Victoria Rayskin^{b,1}

The Pennsylvania State University, Penn State Altoona

^a*Division of Business and Engineering*

^b*Division of Mathematics and Natural Sciences*

Abstract

An algorithm and supporting analysis are presented here for finding roots of systems of continuous equations in two dimensions by bisection iteration. In each iteration, an initial domain in \mathbb{R}^2 is split into equally sized sub-domains. Investigating a candidate domain's bounding path for encirclements of the origin provides the test for containment of a solution, and the domains not guaranteed to contain a solution are discarded. Attention is paid to the potential for accidental convergence to a false solution, and sampling criteria for resolving the boundary are provided with particular emphasis on robust convergence.

Keywords: nonlinear solvers, bisection, roots, dimension

1. Introduction

For the formidable problem of estimating the roots of systems of nonlinear equations, approaches must balance between the competing interests of computational efficiency and robust convergence. At one extreme, brute-force exploration of a domain is extremely stable but horribly inefficient. At the other extreme, classical favorites (like Newton-Rhaphson) strive to be efficient, but often require substantial effort to obtain reliable convergence in “badly behaved” functions. Especially given the broad commercial availability of CPUs with parallel processing, there is renewed purpose for algorithms that make some compromises for the sake of convergence if they can be readily parallelized.

*crm28@psu.edu

Harvey and Stenger proposed an extension of bisection iteration to two dimensions in 1976 (see [8]), and extensions to higher dimensions have since been proposed [11, 5]. Higher dimensional bisection implementations are usually inhibited by the difficulty of establishing a reliable test for a domain’s containment of a solution. Stenger et. al. addressed the problem by applying the Kronecker Theorem [9] to establish a test based on the topological degree [12] of a domain’s boundary with respect to the origin. If the topological degree is non-zero, the domain is guaranteed to contain a solution.

As we address these ideas in more detail, it should become apparent that implementation has motivated us to consider a boundary’s “angle of encirclement” around a solution, but the result is fundamentally the same as topological degree. Readers may also recognize these as derivative of the Cauchy Residue Theorem [2].

The value in the present approach is found in breaking from the method of counting axis crossings for determining topological degree. Harvey and Stenger utilized the approximation of topological degree with the help of the formula, introduced in [12]:

$$d(f, P, \mathbf{0}) \approx \sum_{i=1}^N \frac{1}{8} \left| \begin{array}{cc} u_i & v_i \\ u_{i+1} & v_{i+1} \end{array} \right|. \quad (1)$$

In this formula, given functions $f_0(\mathbf{x}), f_1(\mathbf{x})$ of $\mathbf{x} \in \mathbb{R}^2$, u_i and v_i are defined by

$$\begin{aligned} u_i &= \text{sign} f_0(\mathbf{x}_i), \\ v_i &= \text{sign} f_1(\mathbf{x}_i). \end{aligned}$$

The series $\{\mathbf{x}_i\}_{i=1}^N$ (with $\mathbf{x}_{N+1} := \mathbf{x}_1$) constitutes the vertices of a closed polygonal path, and the segment $\overline{\mathbf{x}_i, \mathbf{x}_{i+1}}$ is allowed to have at most one sign change. In this way, the formula uses the number of sign changes to count the encirclements of the origin, but it will fail if multiple sign changes occur over a single segment $\overline{\mathbf{x}_i, \mathbf{x}_{i+1}}$.

In practice, unless f_0 and f_1 have a high degree of regularity, it is easy to include multiple sign changes as the algorithm progresses, so the computational complexity of algorithm carefully verifying this condition can be high. Furthermore, the algorithm becomes quite concerned with the path in proximity to axes which may have been arbitrarily chosen.

In the present work, we tolerate the mild computational cost of calculating the topological degree explicitly; by summing up the angles formed by

polygonal segments and the origin. In our approach, resolution is added to the path based solely on its proximity to the origin, and attention is paid to the impact this has as the domain is refined. As a result, the method is insensitive to coordinate rotations and enjoys fewer constraints on the functions' behavior. By example, we demonstrate the algorithm's capability to avoid accidental convergence to "ghost" solutions, which is a problem against which previous authors have warned.

1.1. Bisection in One Dimension

Two-dimensional bisection is probably best developed in the context of its one-dimensional predecessor. For some function, $f(x)$, about which very little may be known, this algorithm seeks a value, x , such that $f(x) = 0$.

To facilitate extension into higher dimensions, we adopt this realization of the algorithm: Given a function, $f : \mathbb{R} \rightarrow \mathbb{R}$, that is continuous over some connected domain, $\Omega \subset \mathbb{R}$, we define the domain $\Omega_f = \{f(x) \mid x \in \Omega\}$. For convenience we adopt the notation that $\Omega_f = f(\Omega)$. If the boundary of Ω is formed by the points a and b , then we also define the set $\Omega'_f = [f(a), f(b)]$. In many cases, Ω_f and Ω'_f will be identical, but as Figure 2 illustrates, in general $\Omega'_f \subseteq \Omega_f$.

Because of the continuity of f , we may say that Ω definitely contains a solution if Ω'_f contains the origin. Therefore, bisection proceeds as follows:

1. **Define an initial domain:** In a single dimension, this is accomplished by establishing a and b . The true extent of Ω_f will not be known, but we establish Ω'_f by calculating $f(a)$ and $f(b)$.
2. **Test the initial domain for a solution:** If Ω'_f contains the origin, then Ω definitely contains a solution. This test can have three outcomes.
 - (a) If $f(a)$ or $f(b)$ is zero (or numerically small), the corresponding value can be immediately returned.
 - (b) If Ω'_f does not contain the origin, then the algorithm cannot proceed. Halt with an error.
 - (c) If Ω'_f does contain the origin, move on.
3. **Bisect the domain:** Add a point, m , at the midpoint of Ω . Redefine Ω to be either of the new sub-domains, and establish the new Ω'_f by calculating $f(m)$.
4. **Test the new Ω for a solution:** Just as in step 2, this can have three outcomes.

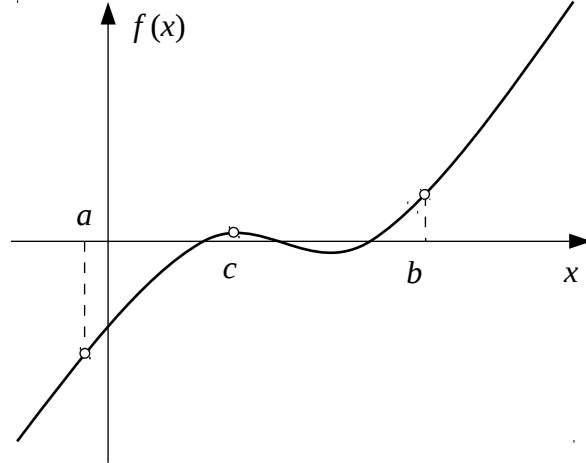


Figure 1: Bisection in one dimension.

- (a) If $f(m)$ is zero or numerically small, return m as the solution.
 - (b) If the new Ω'_f contains the origin, continue to step 5.
 - (c) If Ω'_f does not contain the origin, the domain we disregarded in step 3 *must* contain a solution. Redefine Ω and Ω'_f accordingly and continue.
5. **Repeat:** If the region, Ω , is smaller than the tolerable error in the answer, then return its midpoint as the solution. Otherwise, return to step 3.

In one dimension, the run time of this algorithm (if the origin is contained in Ω'_f) is mainly determined by the number of bisections of the domain. For a given ϵ error, the number bisections, N , is bounded by

$$N \leq \log_2 \left(\frac{\text{size}(\Omega)}{\epsilon} \right). \quad (2)$$

However, in higher dimensions, step 4 is difficult to perform reliably. In this work, we suggest an algorithm which addresses certain critical details for robust convergence.

Figure 1 illustrates a problem suffered by the bisection method. Domains in which the function crosses the x-axis twice (such as $[c, b]$) do not appear

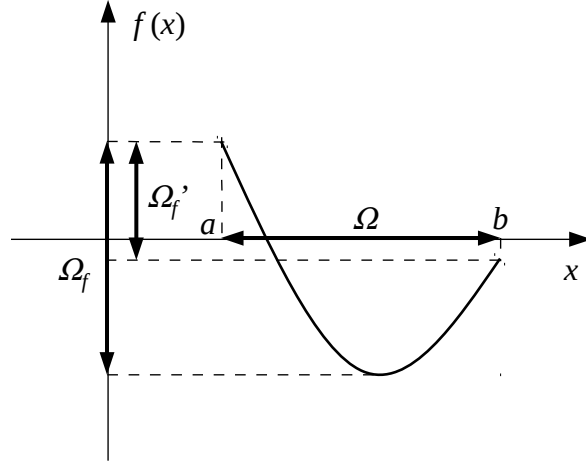


Figure 2: The map between Ω and Ω_f in one dimension.

to contain a solution because the function's sign at the domain's boundary is the same. As a result those solutions will not be identified by the algorithm.

1.2. The problem statement

In two dimensions, we represent a system of nonlinear equations with a two dimensional map, $\mathbf{f} = (f_0(\mathbf{x}), f_1(\mathbf{x}))$ of vectors of two dimensions, $\mathbf{x} = (x_0, x_1)$. For a given map, $\mathbf{f}(\mathbf{x})$, continuous over a simply connected domain $\Omega \subset \mathbb{R}^2$, we seek $\mathbf{x} \in \Omega$ such that

$$\mathbf{f}(\mathbf{x}) = \mathbf{0}. \quad (3)$$

For an extension of the bisection method to two dimensions to be successful, we must have means for implementing the steps identified in section 1.1.

2. Defining a domain

In higher dimensions, there is a rich variety of methods to define a simply connected domain. We adopt the 2-simplex, comprised of three vertices,

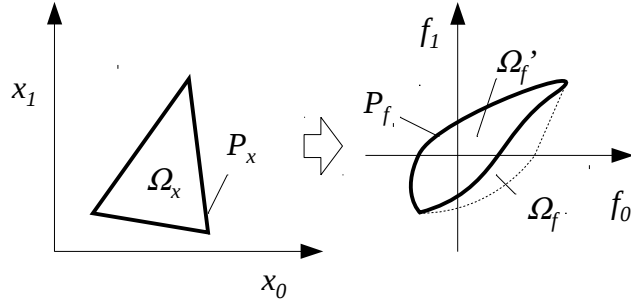


Figure 3: The mapping between Ω and Ω_f .

$\mathbf{x}_a, \mathbf{x}_b, \mathbf{x}_c$, and three corresponding line segments forming a path, P , bounding a triangle, Ω . Due to its simplicity, it is impossible *not* to form a simply connected domain, and the peculiar case where the simplex collapses to a line, as we later discuss, can be avoided.

To the interested reader it will become apparent that this approach can be extended with some difficulty to more complicated regions. However, the simplex may prove sufficiently general, since algorithms employing it lend themselves to any shape that can be constructed from a collection of simplices.

3. Testing for containment

For an initial simply connected region, $\Omega \subset \mathbb{R}^2$, bounded by a closed path, P , there is a corresponding region, $\Omega_f = \mathbf{f}(\Omega)$, and a corresponding path, $P_f = \mathbf{f}(P)$. Since \mathbf{f} is continuous, Ω'_f is a simply connected domain in f -space, bounded by the closed path P_f . Furthermore, if P_f encircles the origin, Ω contains a solution. This mapping is depicted in Figure 3.

Just as was the case in one dimension, it is important to recall that Ω_f

is not necessarily equivalent to Ω'_f (as depicted in Figure 3). Consider

$$\mathbf{f} = \left\{ \begin{array}{c} x_0^2 \\ x_1 \end{array} \right\}. \quad (4)$$

We may select a region, Ω , with the point $x = (0, 0)$ in its interior, so that $(0, 0)$ will clearly be in Ω_f . However, the origin will not belong to Ω'_f because all values of f_0 on P_f will be positive. This is analogous to the problem experienced by the test for containment in one dimension. Therefore, we *cannot* make the stronger assertion that Ω contains a solution if and only if Ω'_f contains the origin.

Equation 4 represents a particularly degenerate case, since it is even quite possible to define a substantial Ω ($\text{int}(\Omega) \neq \{\emptyset\}$) that maps to an Ω'_f with an empty interior. Consider the case when Ω is the unit circle. We tolerate this vulnerability, since one-dimensional bisection suffers from the same problem.

3.1. Calculating encirclement

We test whether Ω'_f contains the origin by calculating the total angle traversed by a ray extending from the origin to P_f . Consider a point \mathbf{f} and a neighboring point, $\mathbf{f} + d\mathbf{f} \in P_f$. The angle formed between rays extending from the origin to each of these two points is given by

$$\begin{aligned} \sin d\theta &= \frac{\mathbf{f} \times (\mathbf{f} + d\mathbf{f})}{\|\mathbf{f}\| \|\mathbf{f} + d\mathbf{f}\|} \\ d\theta &= \frac{\mathbf{f} \times d\mathbf{f}}{\|\mathbf{f}\|^2} \end{aligned} \quad (5)$$

Note that for convenience, we interpret the cross product in two dimensions as a scalar,

$$\mathbf{a} \times \mathbf{b} = a_0 b_1 - a_1 b_0. \quad (6)$$

Therefore, the angle traversed by the ray over one loop around the path is given by

$$\theta = \oint_{P_f} \frac{\mathbf{f} \times d\mathbf{f}}{\|\mathbf{f}\|^2} \quad (7)$$

where, because the path is closed, θ must be some integer multiple of 2π . If θ is 0, then the origin is not contained. If θ is any other value, positive or negative, then the origin is encircled by P_f , and Ω must contain a solution.

In two dimensions, θ is a 2π multiple of the topological degree introduced by Poincaré and Bohl[4],

$$\theta = 2\pi d(\mathbf{f}, P, \mathbf{0}). \quad (8)$$

Furthermore, when $z = f_0 + if_1$, the encirclement is simply related to the residue of the complex function,

$$\begin{aligned} \text{Res}_0 \left(\frac{1}{z} \right) &= \oint_{P_f} \frac{dz}{z} \\ &= \oint_{P_f} \frac{z^* dz}{|z|^2} \\ &= i \oint_{P_f} \frac{\mathbf{f} \times d\mathbf{f}}{\|\mathbf{f}\|^2} + \oint_{P_f} \frac{\mathbf{f} \cdot d\mathbf{f}}{\|\mathbf{f}\|^2} \\ &= i\theta \end{aligned} \quad (9)$$

3.2. Discretization

Equation 7 implies a potentially costly quadrature for each iteration, so we turn our attention to streamlining its evaluation. Any practical algorithm will, at its core, be a sum of small angles, so we consider the angle calculated over a segment of P_f between two points, \mathbf{f}_a and \mathbf{f}_b .

$$\tan(\theta_{ab}) = \frac{\mathbf{f}_a \times \mathbf{f}_b}{\mathbf{f}_a \cdot \mathbf{f}_b} \quad (10)$$

As illustrated in Figure 4, all approximating paths that can be continuously moved to the actual path without passing through the origin will give the same result. There is no need to more finely resolve the quadrature, provided we can be assured that a linear approximation of the path falls on the same side of the origin as does the actual path.

Now the problem is cast in a new light: how sparsely can we discretize the path along P_f , while still being confident that a linear approximation between points passes on the same side of the origin as the actual path?

Consider the deviation of a path from the approximating linear segment. Let δ be a dimensionless scalar between 0 and 1, indicating the location along the path between points a and b . The actual path can be approximated by

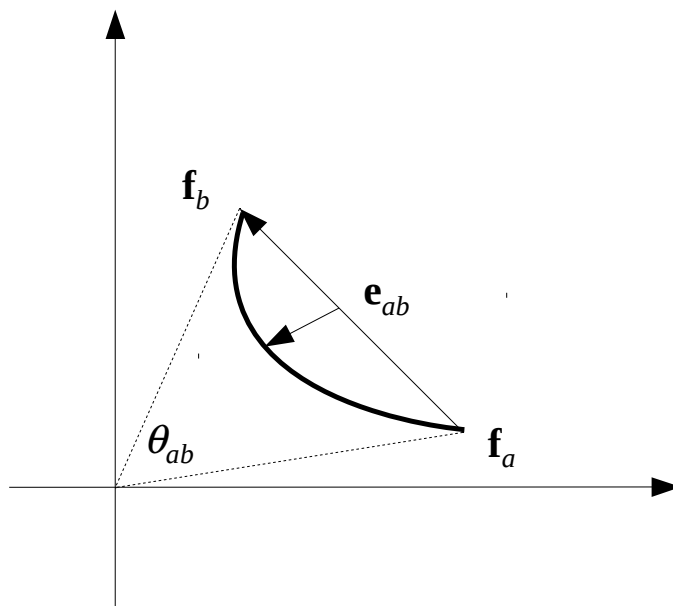


Figure 4: Evaluation of the encirclement between two points \mathbf{f}_a and \mathbf{f}_b . The heavy solid line represents the actual path, while the light solid line represents the linear approximation.

the Taylor expansion,

$$\begin{aligned}
\mathbf{y}(\delta) = & \mathbf{f}(\mathbf{x}_a) \\
& + \delta(\mathbf{x}_b - \mathbf{x}_a) \cdot \nabla \mathbf{f}(\mathbf{x}_a) \\
& + \frac{1}{2} \delta^2 (\mathbf{x}_b - \mathbf{x}_a) \cdot (\nabla \nabla \mathbf{f}(\mathbf{x}_a)) \cdot (\mathbf{x}_b - \mathbf{x}_a) \\
& + \dots
\end{aligned} \tag{11}$$

while the linear approximation is given by

$$\hat{\mathbf{y}}(\delta) = \mathbf{f}(\mathbf{x}_b)\delta + \mathbf{f}(\mathbf{x}_a)(1 - \delta). \tag{12}$$

After several substitutions, the error over the path between \mathbf{f}_a and \mathbf{f}_b (represented by the vector in Figure 4) can be approximated by

$$\begin{aligned}
\mathbf{e}_{ab}(\delta) = & \mathbf{y} - \hat{\mathbf{y}} \\
= & \delta(\delta - 1) \frac{1}{2} (\mathbf{x}_b - \mathbf{x}_a) \cdot (\nabla \nabla \mathbf{f}(\mathbf{x}_a)) \cdot (\mathbf{x}_b - \mathbf{x}_a) \\
& + \dots
\end{aligned} \tag{13}$$

As we discuss in the next section, dominant higher order terms can invalidate this analysis, but assuming that second order terms dominate, the maximum error will occur at the midpoint, $\delta = 0.5$, and will have a magnitude

$$\begin{aligned}
\Delta_{ab} = & \max \|\mathbf{e}_{ab}\| \\
\approx & \frac{1}{8} \|(\mathbf{x}_b - \mathbf{x}_a) \cdot (\nabla \nabla \mathbf{f}(\mathbf{x}_a)) \cdot (\mathbf{x}_b - \mathbf{x}_a)\|
\end{aligned} \tag{14}$$

This provides us with an important result; that approximation error declines like the square of the distance between points. Higher order terms may alter the location of the maximum error, but those behaviors will vanish even more quickly.

Given a pair of points, $\mathbf{x}_a, \mathbf{x}_b \in P$, and the corresponding values, $\mathbf{f}_a = \mathbf{f}(\mathbf{x}_a)$ and $\mathbf{f}_b = \mathbf{f}(\mathbf{x}_b)$, the segment between them should be populated according to the following recursive algorithm:

1. **Bisect the segment:** Create two new segments by adding a new point

$$\mathbf{x}_c = \frac{\mathbf{x}_a + \mathbf{x}_b}{2} \qquad \mathbf{f}_c = \mathbf{f}(\mathbf{x}_c)$$

2. Calculate midpoint error:

$$\Delta_{ab} \approx \left\| \mathbf{f}_c - \frac{\mathbf{f}_a + \mathbf{f}_b}{2} \right\|. \quad (15)$$

3. **Test segments:** If the distance to the origin from the segment between f_a and f_b is less than some multiple of the midpoint error (we use a factor of 2), recursively submit the two new segments (*a-to-c* and *c-to-b*) to step 1 of this algorithm.

The recursion will repeat until the midpoint error converges to some acceptable fraction of the segment’s distance to the origin. In Section 3.4, we discuss certain safeguards to prevent infinite recursion in degenerate cases.

3.3. Rejecting Higher Order Error

When the domain is large, it is entirely possible that maps with higher order characteristics will fool the discretization algorithm in the previous section. Consider the particularly uncooperative cubic case, where

$$\mathbf{e}_{ab}(\delta) \propto 2\delta^3 - 3\delta^2 + \delta, \quad (16)$$

and our midpoint approximation for Δ_{ab} would actually be zero.

When cubic or higher-order terms are important, the algorithm might incorrectly determine that a linear approximation for P_f is quite good. Incorrectly estimating the path can cause the bisection algorithm to wander off into a sub-domain where a solution does not, in fact, exist. The returned value will be a “ghost” solution that may or may not lie anywhere near the actual root. This seems like a substantial problem for our goal to tolerate highly irregular maps.

An effective solution to the problem is to require that the midpoint error test be passed by more than one recursion in a row. Quadratic behavior is detected by the first midpoint test, third- and fourth-order behaviors are detected by an additional recursion, and up to eighth-order behaviors are detected by two additional recursions. Of course, this robustness comes at the cost of additional function evaluations.

3.4. Detecting a Solution on the Boundary

The discretization algorithm we describe in Section 3.2 populates a segment with points by comparing an approximation for path error with the

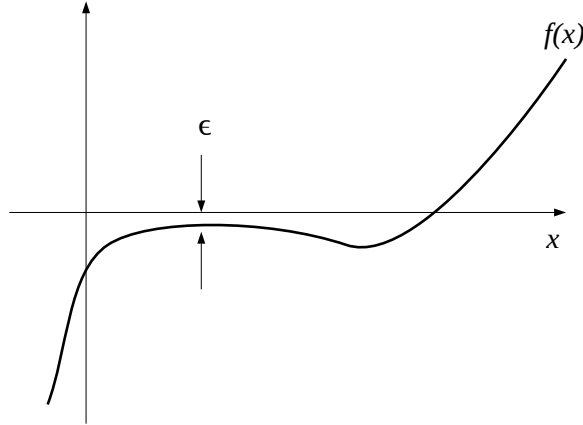


Figure 5: One dimensional problem likely to fool the small error test for convergence.

corresponding segment's distance to the origin. The resulting recursion will converge only when the distance to the origin does not also vanish. This case may seem like a liability, but (when detected) it represents a happy accident; a root lies directly on (or very near to) P_f .

A configurable threshold for a “small” distance from the origin can allow the algorithm to detect this case and return with a solution earlier than expected. One possible test is to define some acceptable error, ϵ , such that if $\|\mathbf{f}(\mathbf{x})\| < \epsilon$, the algorithm will simply return \mathbf{x} . Preferably, to support maps of drastically disparate scales, separate values, ϵ_0 and ϵ_1 should be defined such that when both $|f_0(\mathbf{x})| < \epsilon_0$ and $|f_1(\mathbf{x})| < \epsilon_1$ the algorithm will return \mathbf{x} as a solution.

In general, these cases are rare, and when the authors have observed them in testing, the algorithm still converged to the correct solution unless the solution was deliberately placed precisely on P_f . As a result, we recommend that this option be disabled by default by specifying $\epsilon = 0$. The perils of defining a default error threshold are probably best illustrated by the simple one-dimensional case presented in Figure 5.

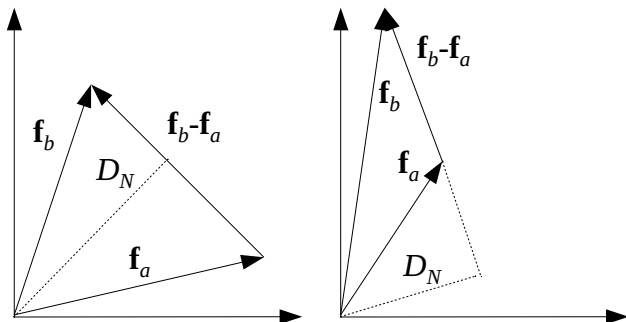


Figure 6: Calculating the distance of a segment from the origin. The projected distance is indicated by a dashed line.

3.5. Calculating Path Distance

The algorithm above relies on calculating a segment's distance from the origin in f -space. For this purpose, we define a segment's distance from the origin as the length of the shortest path between the origin and any point on the segment.

The distance from the origin to an infinite line containing two points, \mathbf{f}_a , and \mathbf{f}_b , is given by

$$\begin{aligned}
 D_N &= \frac{\|\mathbf{f}_a \times (\mathbf{f}_b - \mathbf{f}_a)\|}{\|\mathbf{f}_b - \mathbf{f}_a\|} \\
 &= \frac{\|\mathbf{f}_a \times \mathbf{f}_b\|}{\|\mathbf{f}_b - \mathbf{f}_a\|}
 \end{aligned} \tag{17}$$

However, as shown in Figure 6, Equation 17 underestimates the distance when the closest point does not lie between \mathbf{f}_a and \mathbf{f}_b .

This can be addressed by a piecewise function, dependent on the sign of

the projection of the \mathbf{f}_a and \mathbf{f}_b vectors onto the segment between them.

$$D = \begin{cases} \|\mathbf{f}_a\| & \mathbf{f}_a \cdot (\mathbf{f}_b - \mathbf{f}_a) \geq 0 \\ \|\mathbf{f}_b\| & \mathbf{f}_b \cdot (\mathbf{f}_b - \mathbf{f}_a) \leq 0 \\ \frac{\|\mathbf{f}_a \times \mathbf{f}_b\|}{\|\mathbf{f}_a - \mathbf{f}_b\|} & \text{otherwise.} \end{cases} \quad (18)$$

4. Bisection

Two smaller simplexes of identical area are created when a new edge is extended from one of the points to the midpoint of the edge opposite. Without due caution, this can result in long skinny simplexes. Adler[1] showed that if the longest edge is selected at each iteration, the longest edge of the simplex will asymptotically converge like $(1/\sqrt{2})^k$ after k iterations. Now called longest-edge (LE) bisection, the same method is commonly used in minimization algorithms and to refine finite element grids[3, 10]. The advantage of this approach is made particularly clear in the next section, when we turn our attention to the error of a converged solution.

The final iteration will result in a small simplex wherein somewhere, a solution must lie. As we shall see, the shape of that simplex in x -space is important to minimizing the error of our final estimate. At this juncture, we might make one of two convenient presumptions in order to establish a single point with which to approximate a solution; that all points in the region are equally probable to be a solution; or that the simplex is sufficiently small that we have a reasonable assurance that \mathbf{f} will behave linearly over the domain.

4.1. Uniform Probability Distribution

If we know absolutely nothing about the disposition of the map inside the region, we are forced to presume that there is an equal probability that the solution lies at any given point in the region, i.e., the probability distribution of solutions is uniform. If that is the case, the point that minimizes the expected error is, by definition, the simplex's centroid.

$$\bar{\mathbf{x}} = \frac{\iint_{\Omega} \mathbf{x} \, dx_0 dx_1}{\iint_{\Omega} dx_0 dx_1} \quad (19)$$

$$= \frac{\mathbf{x}_a + \mathbf{x}_b + \mathbf{x}_c}{3}. \quad (20)$$

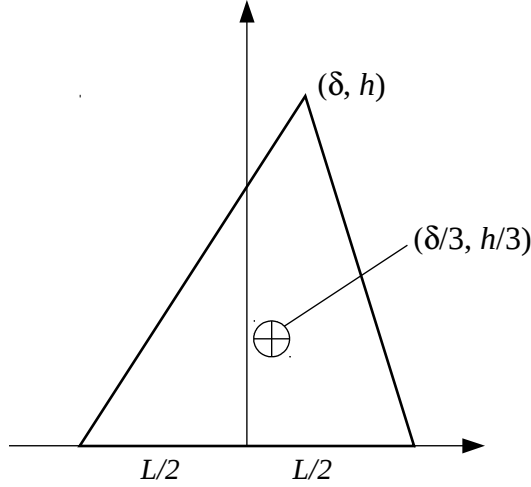


Figure 7: Centroid of a simplex of base, L , height h , and asymmetry δ .

The expected error can be determined by the formula

$$\Delta_{\mathbf{x}}^2 = \frac{\iint_{\Omega} (\mathbf{x} - \bar{\mathbf{x}})^2 dx_0 dx_1}{\iint_{\Omega} dx_0 dx_1}, \quad (21)$$

the numerator of which is simply the area moment of inertia. Figure 7 shows a simplex with an asymmetry indicated by the vertex's distance from the center axis. Since the area moment of inertia is independent of translation and rotation, any simplex can be transformed to this problem. It can be shown that Equation 21 results in

$$\Delta_{\mathbf{x}}^2 = \frac{h^2 + \frac{3}{4}L^2 + \delta^2}{18}. \quad (22)$$

Note that for a given area, the minimum expected error occurs when δ is zero.

Recall that after every iteration, the area of the simplex will be halved. The goal, then, is to minimize expected error for a given amount of computational effort. In general, $\Delta_{\mathbf{x}}^2$ will scale proportionally with area, but Equation 22 indicates that $\Delta_{\mathbf{x}}^2$ is also strongly influenced by the simplex's

shape. Therefore, we want the shape that provides the smallest $\Delta_{\mathbf{x}}$ possible for a given $Lh/2$. It can be shown that this is accomplished by an equilateral triangle, where $h = L\sqrt{3}/2$ and $\delta = 0$.

4.2. Linear Region

When iteration is complete, if the simplex is smaller than the scales of curvature in the map, then they will behave quite linearly over the region. When we allow ourselves to make this assumption about the behavior of \mathbf{f} in our domain, we may interpolate to find the point in the domain most likely to be the solution. The error expected from this approach will depend very much on how well the map complies with our assumption that it is linear over the domain.

Consider a non-orthogonal coordinate system depicted in Figure 8, defined such that

$$\mathbf{x} = \mathbf{x}_a + (\mathbf{x}_b - \mathbf{x}_a)\sigma + (\mathbf{x}_c - \mathbf{x}_a)\gamma \quad (23)$$

$$\mathbf{f} \approx \mathbf{f}_a + (\mathbf{f}_b - \mathbf{f}_a)\sigma + (\mathbf{f}_c - \mathbf{f}_a)\gamma \quad (24)$$

It follows, that the approximate solution will be

$$\mathbf{x} = \mathbf{x}_a - \mathbf{X}\mathbf{F}^{-1}\mathbf{f}_a \quad (25)$$

where \mathbf{F} and \mathbf{X} are 2×2 matrices with columns constructed from the vector differences of Equations 23 and 24,

$$\mathbf{X} = \begin{bmatrix} (\mathbf{x}_b - \mathbf{x}_a) & (\mathbf{x}_c - \mathbf{x}_a) \end{bmatrix} \quad (26)$$

$$\mathbf{F} = \begin{bmatrix} (\mathbf{f}_b - \mathbf{f}_a) & (\mathbf{f}_c - \mathbf{f}_a) \end{bmatrix}. \quad (27)$$

It may appear that this approach is immune to the dependency on shape suffered by the previous section, but the appearance of \mathbf{F}^{-1} creates a vulnerability to singular matrices. This will occur when $(\mathbf{f}_b - \mathbf{f}_a) \propto (\mathbf{f}_c - \mathbf{f}_a)$. While this can occur regardless of the shape, in a map that behaves linearly, it is guaranteed to occur when $(\mathbf{x}_b - \mathbf{x}_a) \propto (\mathbf{x}_c - \mathbf{x}_a)$, or when the simplex collapses (or nearly collapses) onto a line.

The conclusion we draw is that the linear approach has the potential to provide greater accuracy in the final estimate, but it exposes the algorithm to vulnerabilities to division by zero without relief from the problems we see with the uniform-probability-distribution approximation. In general, we

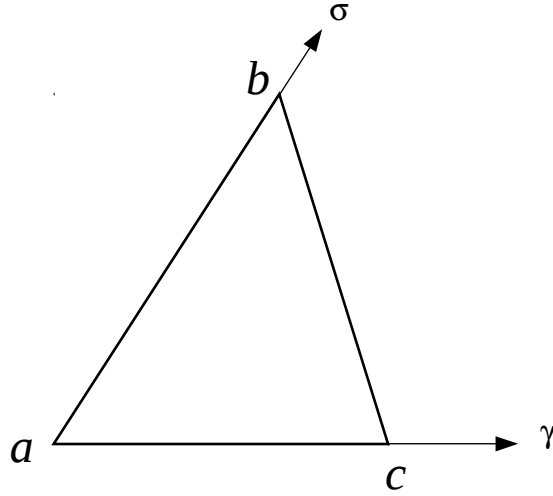


Figure 8: Coordinate system for interpolating in a simplex.

do not recommend using linear interpolation for the solution. If greater accuracy at lower computation cost is required, it may prove more prudent to use bisection to produce an estimate quite close to a solution and use Newton-Raphson iteration to “polish” the root to the desired precision.

5. Application

The performance and limitations of the algorithm are, perhaps, best illustrated through example. In this final section, we illustrate the one of the algorithm’s vulnerabilities and its solution, and we propose future work for its extension.

5.1. Vulnerability to high-order

Here, we demonstrate the inversion of an example mapping exhibiting traits deliberately intended to be problematic.

$$f_0 = x_0 + x_1 \exp(x_0) \tag{28}$$

$$f_1 = x_1^3 \tag{29}$$

Of course, the root is obviously at $\mathbf{x} = (0, 0)$.

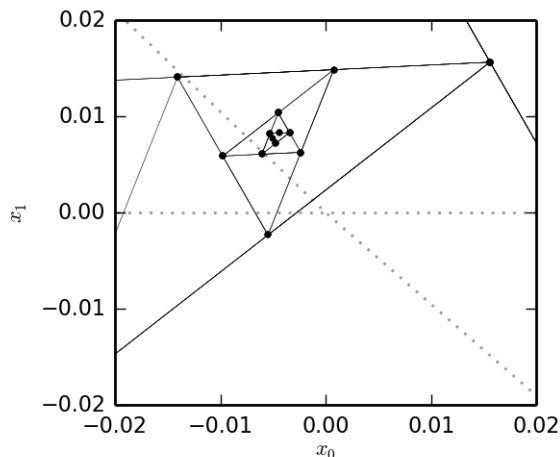


Figure 9: Paths in x -space converging to a ghost solution. The gray curves indicate 0-level paths for f_0 and f_1 , so their intersection is the correct solution.

We execute the algorithm we have described here with initial vertices at $(1,0)$, $(-5,1)$, and $(-9,-1)$. These conditions actually violate assumption II on page 360 of Harvey and Stenger’s original paper[8]. Figure 9 shows a portion of the progression in x -space. It is immediately apparent that the algorithm has incorrectly calculated an encirclement and has converged to a ghost solution. Figure 10 tells us why; the cubic behavior has rendered the single-recursion midpoint error test inadequate.

Figures 11 and 12 show the results of the same problem executed while requiring one additional recursive midpoint test (as described in Section 3.3). Note that the cubic character of f_1 is much more clearly resolved in the vicinity of the origin and that the algorithm now converges correctly.

5.2. Complex Eigenvalues

Bisection is already popular for identifying strictly real eigenvalues[7, 6]. In these cases, of course, the iteration is strictly one-dimensional, and there are already good means for bracketing roots. Consider, for example, the present algorithm were applied to identify eigenvalues of the form

$$x_0 + ix_1 = \lambda \tag{30}$$

$$f_0 + if_1 = \det(A - \lambda B), \tag{31}$$

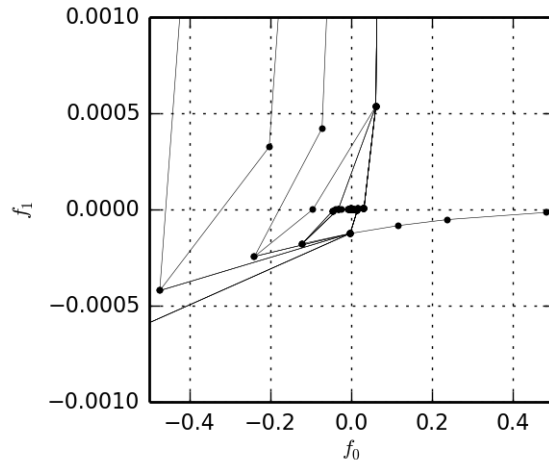


Figure 10: Paths in f-space converging to a ghost solution.

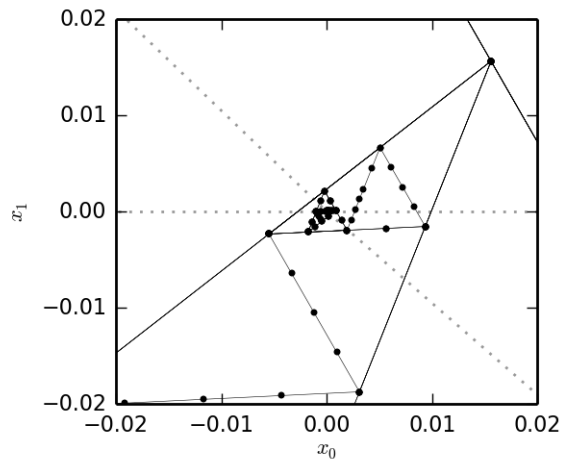


Figure 11: Paths in x-space converging to the correct solution.

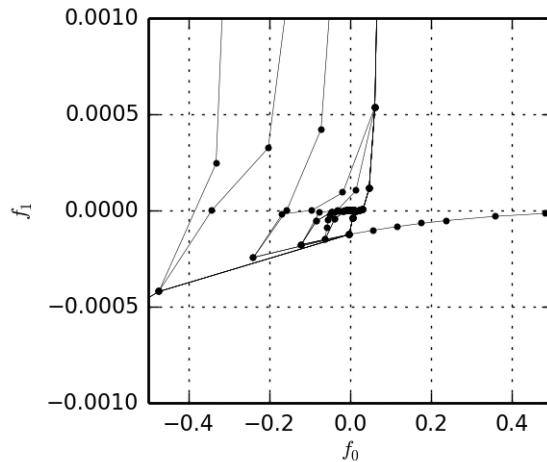


Figure 12: Function evaluations corresponding to the points found in Figure 11.

This represents a complexification of a typical generalization of the sparse-matrix eigenvalue problem common to solid mechanics problems.

5.3. Extension to Higher Dimensions

Extension of this modified algorithm to higher dimensions can be achieved by substituting a solid angle formula for Equation 7. Challenges arise in establishing functional sampling criterion on the domain's boundary. Intuitively, a midpoint sampling method similar to the one we use may prove functional, but there are certain questions of geometric implementation that need to be addressed. For example, in the algorithm we present here, by interrogating the domain's boundary at segments' midpoints, the boundary has already been sub-divided in a manner that lends itself to the domain's eventual bisection. Caution must be used if the same helpful property is to be retained in higher dimensions.

- [1] Andrew Adler. On the bisection method for triangles. *Mathematics of Computation*, 40(162):571–574, 1983.
- [2] Lars Ahlfors. *Complex Analysis: Theory of Analytic Functions of One Complex Variable*. McGraw Hill, New York, 1979.

- [3] Guillermo Aparicio, Leocadio Casado, Eligimus Hendrix, Inmaculada García, and Boglárka Tóth. On computational aspects of a regular n -simplex bisection. In *Eighth International Conference on P2P, Parallel, Grid, Cloud, and Internet Computing*, pages 513–518, 1983.
- [4] P. Bohl. über die bewegung eines mechanischen systems in der nähe einer gleichgewichtslage. *Journal für die reine und angewandte Mathematik*, 127:179–276, 1904.
- [5] A. Eiger, K. Sikorski, and F. Stenger. A bisection method for systems of nonlinear equations. *ACM Transactions on Mathematical Software*, 10(4):367–377, 1984.
- [6] D.J. Evans and J. Shanehchi. Implementation of an improved bisection algorithm in buckling problems. *Numerical methods in engineering*, 19(7):1047–1052, 1983.
- [7] D.J. Evans, J. Shanehchi, and C.C. Rick. A modified bisection algorithm for the determination of the eigenvalues of a symmetric tridiagonal matrix. *Numerische Mathematik*, 38:417–419, 1982.
- [8] CH. Harvey and Frank Stenger. A two dimensional analogue to the methods of bisections for solving nonlinear equations. *Quarterly of Applied Mathematics*, 33:351–368, 1976.
- [9] J. M. Ortega and W. C. Rheinboldt. *Iterative solutions of nonlinear equations in several variables*. Academic Press, New York, 1970.
- [10] Marcia-Cecilia Rivara. Lepp-bisection algorithms, applications and mathematical properties. *Applied Numerical Mathematics*, 59:2218–2235, 2009.
- [11] Krzysztof Sikorski. A three-dimensional analogue to the method of bisections for solving nonlinear equations. *Mathematics of Computation*, 33(146):722–738, 1979.
- [12] Frank Stenger. Computing the topological degree of a mapping in n -space. *Numerical Mathematics*, 25:23–38, 1975.