

Cheap contouring of costly functions: The Pilot Approximation Trajectory Algorithm

Janne M.J. Huttunen^{1,3} and Philip B. Stark²

¹ Department of Applied Physics, University of Eastern Finland, P.O. Box 1627, 70211 Kuopio, Finland.

² Department of Statistics, #3860 University of California, Berkeley, CA 94720-3860, USA.

³ Department of Mathematics, University of Auckland, Private Bag 92019, 1142 Auckland, New Zealand.

E-mail: jmhuttun@gmail.com and stark@stat.berkeley.edu

Abstract. The Pilot Approximation Trajectory (PAT) contour algorithm can find a contour of a function accurately when it is not practical to evaluate the function on a grid dense enough to use a standard contour algorithm, for instance, when evaluating the function involves conducting a physical experiment or a computationally intensive simulation. PAT relies on an inexpensive pilot approximation to the function, such as interpolating from a sparse grid of inexact values, or a solving a PDE numerically using a coarse discretization. For each level of interest, the location and “trajectory” of an approximate contour of this pilot function are used to decide where to evaluate the original function to find points on its contour. Those points are joined by line segments to form the PAT approximation of the contour of the original function.

Approximating a contour numerically amounts to estimating a lower level set of the function, the set of points on which the function does not exceed the contour level. The area of the symmetric difference between the true lower level set and the estimated lower level set measures the accuracy of the contour. PAT measures its own accuracy by finding an upper confidence bound for this area.

In examples, PAT requires far fewer function evaluations than standard algorithms to estimate a contour accurately. We illustrate PAT by constructing a confidence set for viscosity and thermal conductivity of a flowing gas from simulated noisy temperature measurements, a problem in which evaluating the function to be contoured requires running finite-element finite-difference code for coupled, nonlinear PDEs.

1. Introduction

A contour, level curve, or isoline of a real-valued function of two variables $g : \Delta \rightarrow \Re$ is the set of points in the domain Δ of the function on which g takes one particular “level” τ :

$$\gamma_{g,\tau} \equiv \{x \in \Delta : g(x) = \tau\}, \quad (1)$$

Contours are useful in many contexts, including visualizing geographical and meteorological data.

Typical contouring algorithms start with the values of g on a dense, regular grid. They approximate values of g between grid points by interpolation, usually linear interpolation (Aramini 1980, Cottafava & Moli 1969, McLain 1974). Work on efficiency in contouring algorithms primarily addresses large data sets; see, for example, Agarwal et al. (2008) or van Kreveld et al. (1997). Contouring algorithms are closely related to algorithms for constructing level sets $\{x \in \Delta : g(x) \geq \tau\}$. Sethian (1999), Scott & Davenport (2007) and Willett & Nowak (2005) discuss efficient algorithms for approximating level sets.

We address a different bottleneck: finding approximate contours when it is very expensive to evaluate g . If g is expensive to evaluate and varies nonlinearly—for example, if evaluating g requires performing a physical experiment, running a computationally intensive simulation (as in climate modeling), or solving coupled nonlinear partial differential equations (PDEs) numerically—it can be impractical to evaluate g on a grid dense enough to approximate contours accurately using standard methods.

Bryan et al. (2005) and Bryan & Schneider (2008) describe an active learning algorithm for identifying contours of a function. Their algorithm uses statistical criteria to decide where to evaluate the function. In this paper we describe a different approach: We use an approximation of the function as a starting point for an iterative search to locate the contour to any desired accuracy. Finally we check the accuracy by random sampling. The check does not require any assumptions about the function or the accuracy of the initial approximation.

In applications, particular contours such as the zero contour can be especially interesting. Examples are given below. In some such cases, an accurate approximate contour can be found using relatively few function evaluations by starting with an “educated guess” about where the contour is—a pilot approximation—then refining that approximation adaptively.

We present an algorithm that exploits this tactic, *Pilot Approximation Trajectory* (PAT). PAT contours at one level at a time. It takes an advantage of the fact that g can be evaluated anywhere in Δ —albeit at a cost. By judiciously selecting where to evaluate g and avoiding evaluating g far from the level of interest, PAT requires fewer evaluations of g than standard contouring methods.

PAT starts by building a cheap pilot approximation \hat{g} of g . The pilot approximation might be constructed by interpolating values of g on sparse grid, for instance using splines, or if g is the numerical solution to a PDE, \hat{g} might be a solution based on a coarser discretization. The pilot approximation is used to predict the location and trajectory (local orientation) of the contour, which are then refined by evaluating g at additional points. PAT treats g as a “black box,” so it can estimate contours of complicated functions, including nondifferentiable functions involving real or simulated experiments.

To compare the performance of PAT to standard contouring algorithms requires quantifying the error of approximate contours. A contour divides the domain Δ into a set where g has values above the level τ and a set where the value of g is less than or equal to τ . The latter is called the *lower level set* of g (at level τ):

$$\Lambda_\tau = \Lambda_\tau(g) \equiv \{x \in \Delta : g(x) \leq \tau\} \quad (2)$$

One can quantify accuracy of approximate contours based on how well they perform that classification—on how much the actual and estimated lower level sets differ: how well the approximate contour classifies points in Δ .

An approximate contour can make two kinds of classification errors: it can include values of g that are above the level, and it can fail to include values of g that are at or below the level. Let $\hat{\Lambda}_\tau$ be the estimated lower level set of g at level τ according to the approximate contour. The first kind of error occurs on the set $\hat{\Lambda}_\tau \cap \Lambda_\tau^c$ and the second kind of error occurs on the set $\hat{\Lambda}_\tau^c \cap \Lambda_\tau$. The union of these two sets, the points in the domain Δ that the approximate contour misclassifies, is the symmetric difference between $\hat{\Lambda}_\tau$ and Λ_τ :

$$\hat{\Lambda}_\tau \ominus \Lambda_\tau \equiv (\hat{\Lambda}_\tau \cap \Lambda_\tau^c) \cup (\hat{\Lambda}_\tau^c \cap \Lambda_\tau). \quad (3)$$

The area of this symmetric difference quantifies the error of the approximate contour. We compare PAT and standard contouring on the basis of their computational burden and on this measure of their accuracy in tests where the true contour is known. In practice, the true contour generally is not known. But PAT can use random sampling to find an upper confidence bound on the area of $\hat{\Lambda}_\tau \ominus \Lambda_\tau$, assessing its own accuracy through additional evaluations of g .

The symmetric difference is not the only reasonable definition of the error of an approximate contour, even among measures based on the lower level set. For instance, if misclassifying some points in the domain has more serious consequences than misclassifying others, the fraction of the area of the domain on which the approximate contour errs might not be the most relevant measure of accuracy. One could use weighted areas to account for differing importance, for instance, integrals with respect to a measure other than Lebesgue measure.

Measuring accuracy by the area of the symmetric difference between the true and approximate lower level sets is meaningful in applications. For instance, one approach to uncertainty quantification, the *ensemble of models* method, (Higdon et al. 2010) relies on dividing the set of possible parameter values (*models*) into those that agree “adequately” with the data and those that do not. Agreement is measured using a *misfit function*. The set of models that agree adequately with the data is a lower level set of the misfit function. In some problems, a lower level set of the misfit is a *confidence set* for the model that generated the data. Our motivating application is to find such a confidence set for a two-dimensional parameter from indirect, noisy measurements—an inverse problem. This is related to the example that motivates the work of Bryan et al. (2005).

The paper is organized as follows. The next subsection describes a common approach to constructing a confidence set from noisy data, the problem that motivates this work. Section 3 presents a basic version of PAT and a refinement to treat delicate cases, such as saddle points. It also discusses constructing confidence bounds on the fraction of the domain that the approximate contour correctly classifies as above or below the contour level. Section 4 applies PAT to two examples: contouring a function with a saddle point and finding a confidence set for viscosity and thermal conductivity from simulated data in an airflow inverse problem. Section 5 summarizes our findings.

2. Confidence sets based on misfit

Consider estimating the parameter $x_0 \in \mathcal{X}$ from observations $Y \in \mathbb{R}^n$ that are related through

$$Y = f(x_0) + \epsilon,$$

where f is a known function, ϵ is stochastic observational error, and x_0 is unknown.

The function f maps $x \in \mathcal{X}$ into an n -vector $(f_1(x), \dots, f_n(x))^T$. Computing f might involve Monte Carlo simulation or the numerical solution of a PDE by finite-element methods, but we ignore uncertainty in calculating $f(x_0)$ for known x_0 . See Stark (1992), Stark (2008) or Kaipio & Somersalo (2005) for a treatment of systematic errors in inverse problems, such as errors that result from approximating f .

The additive error ϵ is a random n -vector $(\epsilon_1, \dots, \epsilon_n)^T$. The error ϵ has expected value zero and known, finite, positive-definite covariance matrix $\Sigma = \mathbb{E}\epsilon\epsilon^T$. Let $\sigma_k = \sqrt{\mathbb{E}\epsilon_k^2} = \Sigma_{kk}^{1/2}$ be the standard deviation of the k th component of ϵ . The parameter x_0 is fixed; the only random element in the problem is the error ϵ .

Suppose $C(\cdot)$ is a measurable mapping from the data space \mathbb{R}^n into subsets of the parameter space \mathcal{X} . If C satisfies

$$\mathbb{P}_x\{C(Y) \ni x\} \geq 1 - \alpha, \quad \forall x \in \mathcal{X}, \quad (4)$$

then C is a $1 - \alpha$ confidence procedure. Inequality (4) says that the *coverage probability* of $C(Y)$ is at least $1 - \alpha$, whatever be $x_0 \in \mathcal{X}$. If we observe the datum $Y = y$, then $C(y)$ is a $1 - \alpha$ confidence set for x_0 .

Before the data are observed, $C(Y)$ is not a fixed set—it is random—and it is meaningful to talk about the chance that $C(Y)$ will end up containing x_0 . In repeated experiments, the fraction of 95% confidence sets that include the true parameter value x_0 converges in probability to 95%, by the Law of Large Numbers. Once the data have been observed, we talk about *confidence level* rather than coverage probability: Nothing random is left. For instance, the assertion

$$\mathbb{P}_{x_0}\{C(y) \ni x_0\} \geq 1 - \alpha$$

does not make sense, because x_0 is a fixed parameter and $C(y)$ is a fixed set. Either x_0 is in $C(y)$ or not; there is no probability.

We consider confidence sets derived from misfit functions: The confidence set contains all parameter values for which the predicted data agree adequately with the observed data. Agreement is measured in a norm. “Adequately” is calibrated to make the coverage probability $1 - \alpha$.

Let $\|\cdot\|_{p,\Sigma}$ be a p -norm on \mathbb{R}^n with weights derived from Σ . For instance, for $p = 2$ we might define

$$\|y\|_{2,\Sigma} \equiv \sqrt{y^T \Sigma^{-1} y}$$

or, for $p = \infty$,

$$\|y\|_{\infty,\Sigma} \equiv \max_{k=1,\dots,n} |z_k / \sigma_k|.$$

Let $\chi_{p,\Sigma,\alpha}$ be the smallest value of c for which

$$\mathbb{P}\{\|\epsilon\|_{p,\Sigma} \leq c\} \geq 1 - \alpha.$$

Then

$$\mathbb{P}_x\{\|f(x) - y\|_{p,\Sigma} \leq \chi_{p,\Sigma,\alpha}\} \geq 1 - \alpha, \quad \forall x \in \mathcal{X}.$$

So, if we define

$$C(y) \equiv \{x \in \mathcal{X} : \|f(x) - y\|_{p,\Sigma} \leq \chi_{p,\Sigma,\alpha}\},$$

then $C(Y)$ is a $1 - \alpha$ confidence set for x_0 . See, for example, Stark (1992) for examples and extensions. The case $p = 2$ is sometimes called a χ^2 confidence set; the case $p = \infty$ is sometimes called a maximum modulus confidence set.

As a concrete example, suppose the n components of ϵ are independent, identically distributed (iid) zero-mean Gaussian random variables with common variance σ^2 . Then $\Sigma = \sigma^2 \mathbf{I}$, where \mathbf{I} is the n by n identity matrix; $\chi_{2,\Sigma,\alpha}$ is σ times the square-root of the $1 - \alpha$ percentage point of the chi-squared distribution with n degrees of freedom.

For a fixed choice of p and α and a fixed y , define the real-valued function g on \mathcal{X} :

$$g(x) \equiv \|f(x) - y\|_{p,\Sigma} - \chi_{p,\Sigma,\alpha}. \quad (5)$$

A $1 - \alpha$ confidence set for x_0 based on the observation $Y = y$ is thus

$$C(y) = \{x \in \mathcal{X} : g(x) \leq 0\}.$$

This is the lower level set of g at level $\tau = 0$. Continuity of g follows from continuity of the p -norm and the assumed continuity of f . Hence, the confidence set—which typically is not simply connected—is closed relative to \mathcal{X} , and the contour $\gamma_{g,0} = \{x : g(x) = 0\}$ is the boundary of the confidence set.

3. The PAT Algorithm

PAT is related to common contouring algorithms like those built into high-level languages such as MATLAB and R. Section 3.1 sketches those algorithms. Section 3.2 describes the basic PAT algorithm. Sections 3.3 and 3.4 present refinements.

Common algorithms and PAT both have the same main steps:

- (i) Partition the domain into rectangular cells.
- (ii) Find a cell edge crossed by a piece (a maximally connected subset) of the contour that has not yet been traced.
- (iii) Find a maximal series of edges of adjacent cells crossed by that piece of the contour.
- (iv) Find (or estimate) points at which the piece of the contour intersects each of those edges.
- (v) Connect those points with line segments (*trace* the piece).
- (vi) Check whether there are additional pieces of the contour. If so, return to the second step; else, stop.

The largest differences are in how these steps are performed: how an untraced piece of the contour is found, how the edges crossed by each piece are found, how the (approximate) points where the piece intersects those edges are found, whether points on the contour in the interior of cells are found and used to construct the approximate contour, and how the check for additional pieces is performed. PAT also has extra “quality control” steps to look for pieces of the contour that might have been missed and to estimate the error of the approximation. This step reliably assesses the accuracy of the approximate lower level set even if the assumptions of PAT do not hold.

3.1. A simple contour algorithm

Let g be a continuous function of two real variables. Let $\Delta \equiv [a_1, b_1] \times [a_2, b_2] \subset \mathbb{R}^2$. We seek the contour

$$\gamma_{g,\tau} \equiv \{x \in \Delta : g(x) = \tau\},$$

for a given level τ . Our approximation to $\gamma_{g,\tau}$ is denoted $\hat{\gamma}_{g,\tau}$. Without loss of generality, we assume that $\tau = 0$, since we can always contour $g - \tau$ instead of g . We suppress τ from the notation unless $\tau \neq 0$. The *contour* γ_g consists of a union of maximally connected pieces, indexed by i .

Standard algorithms for approximating γ_g start by dividing Δ into N smaller rectangles (*cells*) $\{\Delta_k\}_{k=1}^N$. They also rely on knowing the value of g on the grid defined by the vertices of the cells. They approximate g at points not on the grid by interpolation, typically linear interpolation. Such algorithms rely on assumptions like the following:

- (i) The function g is continuous.
- (ii) The area of γ_g is zero.
- (iii) Each piece of γ_g is either a closed curve or both of its endpoints are on the boundary of Δ .
- (iv) The function g is equal to the level τ at most once on any edge of any cell.
- (v) No piece of γ_g is entirely confined to the interior of a cell.

Assumptions (ii) and (iii) imply that g does not have a local extrema at which the value of g is 0. Assumptions (iv) and (v) ensure that the cells are small enough to find all the pieces of the contour. Aramini (1980) proposes approaches to obviate the need for Assumption (iv) by adaptively subdividing cells; see also Suffern (1990). The method of Aramini (1980) requires knowing the approximate magnitude of the Laplacian of g , but we are avoiding the assumption that g is differentiable. Instead, we assume that the cells are sufficiently small that γ_g crosses no edge more than once. The PAT confidence bound on its own error—the area of the symmetric difference between the real and approximate lower level sets—does not depend on any of these assumptions.

Figure 1 sketches how a typical contouring algorithm works. First, the algorithm compares values of g at adjacent vertices until it finds an edge that the contour crosses. When it finds such an edge, the algorithm uses linear interpolation to estimate the point at which the contour intersects that edge. Then the algorithm moves to an adjacent cell, finds an edge of that cell crossed by the contour, and uses linear interpolation to estimate where the contour intersects this second edge. This continues until the algorithm either returns to the cell found in the first step, or reaches the boundary of the domain.

If the algorithm returns to the cell found at the first step, this piece of the contour is a closed curve; the points where the contour was estimated to intersect the edges of cells are connected by straight lines (traced) to form a closed polygon that approximates the corresponding piece of γ_g . On the other hand, if the algorithm reaches the boundary of the domain Δ before returning to the first cell, assumption (iii) guarantees that this piece of the contour intersects the boundary of Δ twice. To find the other point where the contour intersects the boundary of Δ , the algorithm returns to the first point found and moves in the opposite direction until it reaches the boundary. Then the points at which this piece of the contour were estimated to

intersect cell edges are joined by straight lines (traced) to form a piecewise linear curve that approximates the corresponding piece of γ_g .

The contour might consist of disjoint pieces, so when the algorithm finishes tracing one piece, it checks “unvisited” cells to see whether the contour has another piece. If so, that piece is traced as described above. This process continues until every cell has been visited. If no edge is crossed by a contour, assumption (v) guarantees that the contour is the empty set.

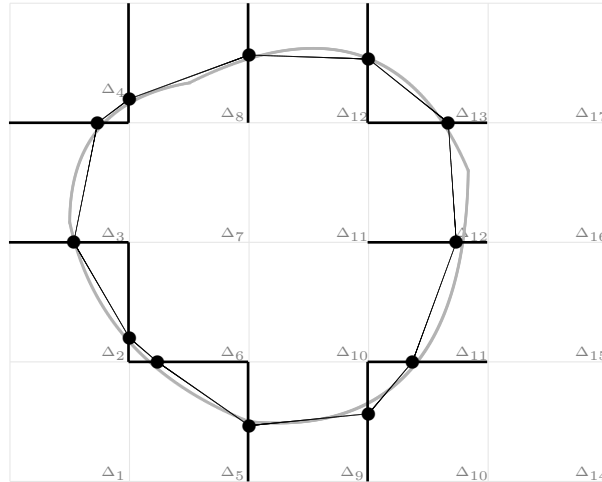


Figure 1. How a typical contour algorithm works. The true contour γ_g is the gray curve. The approximate contour $\hat{\gamma}_g$ is the black polygon. The algorithm first finds a cell (Δ_2 in this case) with an edge that the contour intersects (black edges). The algorithm picks one of those edges and approximates where that piece of the contour intersects the edge (black dots) by linear interpolation. The algorithm then moves from edge to edge of the cell to find an adjacent cell with an edge that intersects the contour (here, Δ_6). This continues until the algorithm returns to the cell it started from or reaches the boundary of the domain (in both directions). The intersection points are connected by line segments to trace that piece of the contour. Then the algorithm checks unvisited cells to see whether the contour has other pieces. If so, the procedure repeats until there are no untraced pieces.

3.2. Contouring costly functions: PAT

As mentioned above, PAT is a modification of contouring algorithms like those just described. PAT assumes that g can be evaluated anywhere in Δ . PAT is designed for situations where evaluating g is expensive, so it seeks to keep the number of evaluations small but still approximate γ_g accurately.

PAT uses a pilot approximation \hat{g} to determine where to evaluate g . For instance, if g involves an expensive simulation or physical experiment, \hat{g} might be a spline function fitted to values of g at a small number of points in Δ . If computing g requires solving a PDE using a finite-element method, \hat{g} might be based on a finite-element model using a coarse discretization.

In general, the contours of g and \hat{g} intersect different cells. To find a cell intersected by γ_g , PAT starts at a point on $\gamma_{\hat{g}}$, the contour of \hat{g} . It searches a line

perpendicular to $\gamma_{\hat{g}}$ for a cell intersected by the contour of g . (It should find such a cell eventually unless γ_g and $\gamma_{\hat{g}}$ are nearly orthogonal—which would imply that \hat{g} is a poor approximation to g . In that case, PAT advances to the next edge intersected by $\gamma_{\hat{g}}$ and tries again.) Once such a cell is found, PAT uses the “trajectory” (local orientation) of the contour of \hat{g} to predict that of the contour of g . For instance, if the contour of \hat{g} intersects the left and top edges of the cell in which the search started, PAT starts by assuming that the contour of g intersects the left and top edges of the cell just found by the search. If that assumption turns out to be false, PAT searches the other edges of the cell crossed by the contour of g to find an edge intersected by γ_g . If the assumption is true, PAT evaluates g fewer times, but if the assumption is false, PAT still works.

Like standard contouring algorithms, PAT divides the domain into cells $\{\Delta_k\}_{k=1}^N$. PAT requires both g and \hat{g} to satisfy assumptions (i)–(v). To simplify the exposition, we assume in this section that the contours of g and \hat{g} have same number of disjoint pieces; we relax this assumption in section 3.4. And we assume in this section that γ_g and $\gamma_{\hat{g}}$ each intersect at most two edges of any cell; we relax this assumption in section 3.3.

Below are the steps in the basic PAT algorithm. The cells intersected by the i th piece of the approximate contour are denoted $\{\hat{\Delta}_{j'}^i\}$, $j' = 0, \dots, J'_i$. Cells intersected by the i th piece of γ_g are denoted $\{\Delta_j^i\}$, $j = 0, 1, \dots, J_i$.

0. Construct the pilot approximation \hat{g} of g . Divide the domain Δ into N cells, $\{\Delta_k\}_{k=1}^N$.[‡] Set $k = i = 1$.
1. Mark cell Δ_k as visited.
2. (a) If $k = N$ and Δ_k has no edge for which $\hat{g}(x_1)\hat{g}(x_2) \leq 0$, where x_1 and x_2 are the vertices of the edge, exit.
 (b) Else if $k < N$ and Δ_k has no edge for which $\hat{g}(x_1)\hat{g}(x_2) \leq 0$, increment k and return to step 1.
 (c) Else Δ_k has an edge with $\hat{g}(x_1)\hat{g}(x_2) \leq 0$, so $\gamma_{\hat{g}}$ intersects Δ_k . The assumptions guarantee that it has at least two such edges. Denote two of them \hat{e}_0^i and \hat{e}_{-1}^i . Set $\hat{\Delta}_0^i = \Delta_k$.
3. Find an edge intersected by γ_g as follows:
 - (a) Use linear interpolation to estimate where $\gamma_{\hat{g}}$ intersects the edges \hat{e}_0^i and \hat{e}_{-1}^i . Denote those points by \hat{x}_0^i and \hat{x}_{-1}^i . If \hat{g} has a zero at a vertex of the cell, we can choose that vertex to be one of the points.
 - (b) Find the line perpendicular to the line segment between the points \hat{x}_0^i and \hat{x}_{-1}^i that goes through $(\hat{x}_0^i + \hat{x}_{-1}^i)/2$; see Figure 2. Search for a root of g on this line that is not on a piece of the contour already traced (our implementation of PAT uses Müller’s method (Müller 1956); other methods might be more efficient). If no new root is found on this line, increment k and return to step 1. If a root is found, denote it x_0^i . The point x_0^i is on the piece of γ_g we will trace next.
 - (c) Find the cell containing x_0^i and denote it Δ_0^i .
4. Follow and trace the piece of γ_g just found: Set $j = j' = 0$. Then:
 - (a) Predict which edge γ_g crosses next. The prediction is denoted e_c . If $\gamma_{\hat{g}}$ is in the same cell (i.e., has also entered the cell Δ_{j+1}^i through the edge e_j^i or, on

[‡] We do not address how to select the appropriate cell size, but the pilot approximation \hat{g} may be helpful. See, e.g., Aramini (1980) for methods of selecting cell size.

the first step, started from the same cell), set e_c to be an edge that $\gamma_{\hat{g}}$ crosses next. Otherwise, let e_c be the edge next crossed by the function $\hat{g}(\cdot) - \hat{g}(x_j^i)$, the pilot approximation \hat{g} shifted by its value at the last contour point of g , x_j^i .

- (b) If edges e_c and e_j^i are adjacent, they share a vertex at which g is already known. In that case, evaluate g at the other vertex of e_c to check whether γ_g crosses e_c . If it does, set $e_{j+1}^i = e_c$.
- (c) If e_c is parallel to e_j^i , evaluate g at one vertex of e_c (e.g., the vertex with the smaller absolute value of \hat{g}). Then g is known at both vertices of one edge perpendicular to e_j^i ; check whether γ_g crosses that edge. If so, that edge is e_{j+1}^i . If not, evaluate g at the remaining vertex of Δ_{j+1}^i to find the edge e_{j+1}^i that γ_g intersects.
- (d) Find the root x_{j+1}^i of g on edge e_{j+1}^i .
- (e) If $\gamma_{\hat{g}}$ also crosses edge e_{j+1}^i , trace the portion of $\gamma_{\hat{g}}$ between the edge $e_{j'}^i$ and the cell Δ_{j+1}^i the same way the standard algorithm does. Update the index j' and mark the cells $\hat{\Delta}_{j'}^i$ intersected by that portion of $\gamma_{\hat{g}}$ as visited.
- (f) Move to the cell that is across edge e_{j+1}^i and denote it by Δ_{j+1}^i .

Set $j \leftarrow j+1$ and repeat these steps until either the initial cell Δ_0^i or the boundary is reached. If a boundary is reached, return to Δ_0^i and trace in the other direction, as the standard contour algorithm does. If the end of $\gamma_{\hat{g}}$ has not been reached, trace $\gamma_{\hat{g}}$ to its end and mark all cells on its path as visited. Increment i .

5. If there are cells not yet visited, increment k until Δ_k is not a visited cell and continue from step 1.
6. For each i , connect the points $\{x_0^i, x_1^i, \dots, x_{j_i}^i\}$ with straight line segments to form the approximation $\hat{\gamma}_g$ of γ_g .

To approximate contours at additional levels, the pilot approximation \hat{g} could be reused, or \hat{g} could be improved by incorporating all the points at which g has been evaluated.

3.3. Saddle points

In the previous section we assumed that the contour intersects at most two edges of any cell. This may not hold if (a) the cell contains a saddle point (the contour crosses itself or two pieces of the contour intersect the same cell) or (b) there is a local minimum or maximum in the cell (so that the same piece of the contour intersects the cell four times, or at three edges and a vertex).

There have been attempts to distinguish these situations automatically, for example by using linear interpolation to decide which edges to connect (Aramini 1980) or by evaluating the function at the center of the cell and comparing that value to the values at vertices (Aramini 1980). Instead, we follow the contour in small steps from one edge to another edge. We believe that this approach errs less frequently in practice, but we have not characterized the conditions under which that is true.

Suppose PAT is at step 4(a) and it turns out that γ_g crosses all four edges of the cell. Then the following approach is used to determine the next edge e_{j+1}^i and the point x_{j+1}^i . The idea is to follow the contour from x_j^i to another edge. One way to do this would be to move a short distance along the tangent to γ_g and then move perpendicular to that tangent to get back to γ_g , and repeat these steps until we cross

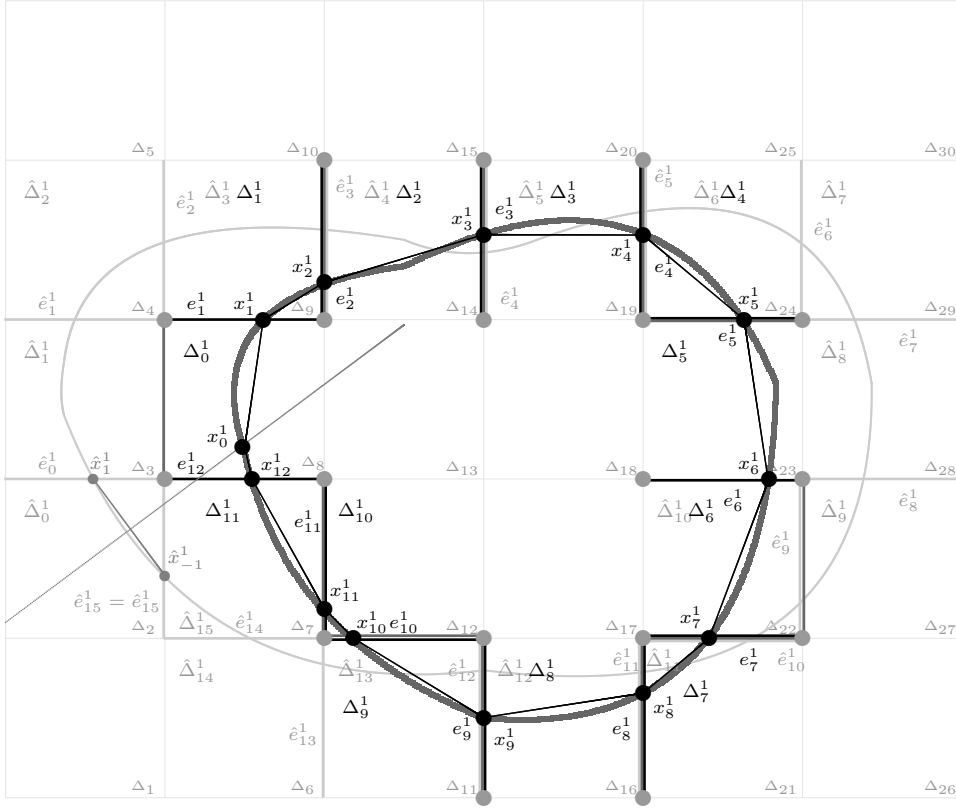


Figure 2. Contouring a costly function g using PAT. The true contour γ_g is the dark gray curve and its PAT approximation $\hat{\gamma}_g$ is the black polygon. The light gray curve is $\gamma_{\hat{g}}$, the approximate contour of the pilot approximation. PAT starts in a cell with an edge intersected by $\gamma_{\hat{g}}$ (here $\Delta_2 = \hat{\Delta}_0^1$) and estimates where $\gamma_{\hat{g}}$ intersects the edges of that cell (dark gray dots) using linear interpolation, just like standard algorithms. Then PAT searches for a root of g (the point x_0^1) on the line perpendicular to the segment between these two intersection points. Tracing of γ_g starts from the cell that contains this zero (here $\Delta_8 = \hat{\Delta}_0^1$). PAT uses the “trajectory” of $\gamma_{\hat{g}}$ to make a guess e_c (dark gray lines on edges) of the next edge e_1^2 of $\hat{\Delta}_0^1$ that γ_g crosses. Then PAT evaluates g at vertices of the cell (gray dots) as necessary to determine whether e_c or a different edge is in fact e_1^2 . PAT determines the point x_1^2 at which γ_g intersects e_1^2 and moves to the next cell γ_g enters. This continues until every piece of the contour has been traced.

an edge. However, since g is not necessarily differentiable, the tangent to γ_g might not exist. Hence, we modify this prediction–correction approach.

We construct a step direction as follows. Pick a small value $\delta > 0$ (for instance, 10% of length of edge e_j^i). Find the unit normal n to edge e_j^i . Set $k \leftarrow 0$, $d_0 \leftarrow \delta n$, and $y_0 \leftarrow x_j^i$. While $y_k \in \Delta_{j+1}^i$:

- (i) Set $\hat{y}_{k+1} \leftarrow y_k + d_k$.
- (ii) On the line through \hat{y}_{k+1} perpendicular to d_k , find the closest root of g . Set y_{k+1} equal to that root.
- (iii) Set $d_{k+1} = \delta(y_{k+1} - y_k)$ and let $k \leftarrow k + 1$.

This approach is illustrated in Figure 3.

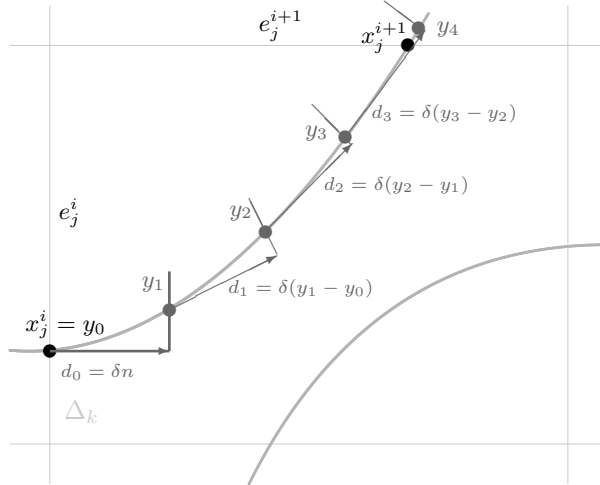


Figure 3. Tracking the contour by prediction and correction to find edge e_j^{i+1} when a cell contains a saddle point.

The edge e_{j+1}^i is chosen to be the edge crossed by the line segment between y_k and y_{k+1} . The point x_{j+1}^i is chosen to be the intersection point on this line segment. The points y_1, \dots, y_k can be included in the series of points connected to form the approximation $\hat{\gamma}_g$ to the i th piece γ_g .

3.4. Missing pieces of the contour

So far, we have assumed that γ_g and $\hat{\gamma}_g$ have the same number of pieces. If $\hat{\gamma}_g$ has more pieces than γ_g , PAT will not succeed in finding a new piece of γ_g when tracing the extra pieces of γ_g , potentially wasting a large number of evaluations of g —but otherwise harmless. On the other hand, if γ_g has more pieces than $\hat{\gamma}_g$, PAT might not find them all. This can matter in applications. For instance, missing a piece of γ_g can cause a misfit-based confidence set to have a confidence level less than its nominal confidence level, by erroneously excluding parameter values that belong in the set. This section discusses how to augment PAT to make it more likely to find all the pieces of γ_g and how to measure statistically the area of the symmetric difference between the lower level set of g and the estimated lower level set of g , as discussed in Section 1.

3.4.1. Heuristic Approaches In general, we cannot be certain that PAT has found every piece of γ_g . However, some heuristic rules can help locate pieces that might otherwise be missed.

Regions where $\hat{\gamma}_g$ is far from γ_g raise suspicion. If $\hat{\gamma}_g$ has disjoint pieces that are close to each other and \hat{g} is very smooth, $\hat{\gamma}_g$ may have fewer pieces than γ_g . We might discover the extra pieces of γ_g by probing such “gaps” between γ_g and $\hat{\gamma}_g$, for example, by evaluating g at the centroid of the gap to see whether g has the sign implied by the PAT contour. If it does not, we can search a line segment between that point and the nearest piece of $\hat{\gamma}_g$ to find a root of g and start tracing a new piece of γ_g there. Our

implementation starts by trying to take one endpoint of the segment to be a point within the gap at which g has already been evaluated, close to the centroid of the gap. If g does not have a different value at the centroid, we also evaluate g at the grid point inside the gap at which the value of \hat{g} is largest or smallest.

If it is inexpensive to find extreme points of \hat{g} , we might check whether \hat{g} has extreme points that are not inside of any piece of $\hat{\gamma}_{\hat{g}}$ or two or more extreme points inside a single piece of $\hat{\gamma}_{\hat{g}}$ and evaluate g at those points. Checking extreme points relies on the heuristic that if \hat{g} is a good approximation to g , its extreme points might be close to those of g . If \hat{g} has a local extremum with a value close to zero, but does not change sign near the extremum, g nonetheless might change sign nearby.

If g has the same sign everywhere it has been evaluated, so PAT has not found any points on the contour, we might use numerical optimization to try to find a point in Δ at which g has the opposite sign. An iterative method could be started from the point with the smallest value of $|g|$ among those observed.

3.4.2. A priori knowledge about g In some cases we may know how many pieces γ_g should have. For example, if g is quasiconvex (or quasiconcave), its level set $\{x : g(x) < 0\}$ is convex (or concave), so γ_g consists of at most one piece and the algorithm can stop as soon as one piece of γ_g has been completely traced.

3.4.3. A priori knowledge about $|\hat{g} - g|$ If we have a bound on the error of \hat{g} as an approximation to g , we can use it to check whether PAT has missed pieces of γ_g . For example, if we knew a function $M(x)$ for which

$$|g(x) - \hat{g}(x)| \leq M(x) \quad \text{for all } x \in \mathcal{X},$$

it would suffice to evaluate g at grid points x_j for which $|\hat{g}(x_j)| \leq M(x_j)$ (perhaps starting with points for which $|\hat{g}(x_j)|M(x_j)^{-1}$ is smallest) and check the sign of g at those points.

3.4.4. Random sampling and contour error We can use random sampling both to look for missing pieces and to collect statistical evidence about the accuracy of the PAT contour $\hat{\gamma}_g$ as an approximation to γ_g . Recall from section 1 that the difference between the true lower level set of g at level τ , Λ_τ , and the estimated lower level set of g , $\hat{\Lambda}_\tau$, is the set on which the approximate contour $\hat{\gamma}_g$ errs in classifying points in Δ . As discussed in the introduction, the approximate contour $\hat{\gamma}_g$ makes classification errors on two sets: points where the true value of g is strictly positive but the approximate contour claims that g is nonpositive, and points where the true value of g is nonpositive but the approximate contour claims that g is positive. The area of the union of those sets—the symmetric difference $\Lambda_\tau \ominus \hat{\Lambda}_\tau$ of Λ_τ and $\hat{\Lambda}_\tau$ —is a measure of the accuracy of the approximate contour. Normalizing that area by the total area of Δ gives the fraction of the area of the domain Δ misclassified by the approximate contour.

To keep things simple, we consider how to estimate the fraction p of the area of Δ that is misclassified by the PAT contour $\hat{\gamma}_g$ —the relative area of the symmetric difference. The chance that a point selected at random uniformly from Δ is misclassified by $\hat{\gamma}_g$ is p . If a point near $\hat{\gamma}_g$ is misclassified, it could mean that $\hat{\gamma}_g$ is slightly out of place. If a point far from $\hat{\gamma}_g$ is misclassified, that suggests that $\hat{\gamma}_g$ is a missing piece of γ_g . To check, we can search for a root of g on any line segment between that point and the nearest piece of $\hat{\gamma}_g$. If we find such a root, it becomes a starting point to trace a piece of γ_g that had been missed.

On the other hand, suppose we draw n points at random, independently, uniformly from Δ , and the sign of g at every point in the sample agrees with its sign according to the PAT contour $\hat{\gamma}_g$: No point in the sample is in the symmetric difference $\hat{\Lambda}_\tau \ominus \Lambda_\tau$. Then an event has occurred that has probability $(1-p)^n$. We can use this to find a $1-\alpha$ upper confidence bound on p : For this event to have probability greater than α , p must be less than $1-\alpha^{1/n}$. For instance, for $n=50$, we would have 95% confidence that $p < 5.8\%$; for $n=100$, we would have 95% confidence that $p < 3\%$.

More generally, if n independent draws find m misclassified points, a $1-\alpha$ upper confidence bound for the relative area of $\hat{\Lambda}_\tau \ominus \Lambda_\tau$, the part of the domain Δ misclassified by the PAT contour, is

$$u(n, m, \alpha) \equiv \max \left\{ p : \sum_{i=0}^m p^i (1-p)^{n-i} \geq \alpha \right\}. \quad (6)$$

Evaluating g at a random sample of points in Δ allows us to have confidence in the accuracy of the PAT contour, despite the fact that the assumptions behind PAT might be wrong—at the expense of n extra evaluations of g .

In some applications, one kind of error is more important than the other. For instance, erroneously including extra points in a confidence set makes the set conservative, while erroneously excluding points from a confidence set is misleadingly optimistic. Moreover, the “cost” of misclassifying points might be higher for some points than for others. For instance, erroneously excluding from a confidence set of climate models a model in which global temperature rises 10K in 20 years might be a more serious error than erroneously excluding a model in which global temperature rises 0.5K in 20 years. Such differences can be taken into account by using weighted areas (integration against a nonuniform measure instead of Lebesgue measure) and carrying out the above sampling using corresponding nonuniform distributions.

4. Tests of PAT

In this section we apply PAT to two test cases: Finding a contour of a 2-dimensional function with a saddle point, at the level of the saddle point, and constructing a confidence set in an inverse problem involving air flow.

4.1. Contouring a function with a saddle point

Let g be a sum of two 2-dimensional Gaussian bumps:

$$g(x) = e^{-(Rx-\mu_-)^T C (Rx-\mu_-)} + e^{-(Rx-\mu_+)^T C (Rx-\mu_+)}, \quad x \in \mathbb{R}^2,$$

where the matrices R and C and the vectors μ_\pm are given by

$$R = \begin{pmatrix} \cos(\frac{\pi}{4}) & \sin(\frac{\pi}{4}) \\ -\sin(\frac{\pi}{4}) & \cos(\frac{\pi}{4}) \end{pmatrix}, \quad C = \begin{pmatrix} 8 & 0 \\ 0 & 10 \end{pmatrix}, \quad \mu_\pm = \begin{pmatrix} \pm 0.6 \\ 0 \end{pmatrix}.$$

The matrix R rotates the coordinate system 45° clockwise. The domain for the contour plot is $\Delta \equiv [-1, 1]^2$. The function g is plotted in Figure 4.

This function has a saddle point at the origin. To test the ability of PAT to contour saddle points, we contour at level $\tau = g(0,0)$, the value of g at the saddle point.

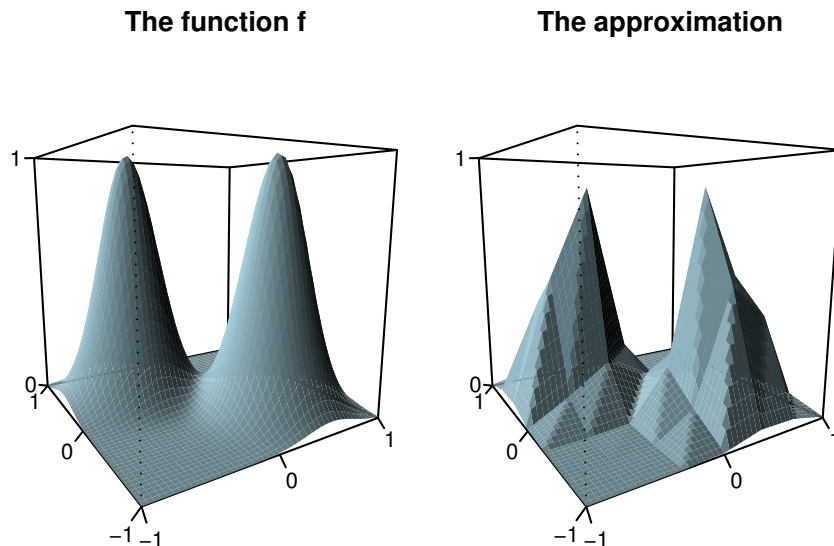


Figure 4. A function g with a saddle point used to test PAC, and its pilot approximation \hat{g} .

To construct the pilot approximation \hat{g} , g was evaluated on an equispaced 9×9 grid in Δ . Every cell in the grid was divided into two triangles, forming a triangular tessellation of Δ . The pilot approximation \hat{g} is the piecewise linear function equal to g at the grid points and linear within each triangle; it is plotted in Figure 4.

The PAT contour $\hat{\gamma}_g$ was found on a 32×32 grid, plotted in Figure 5. For comparison, Figure 5 shows a contour computed using the built-in contouring algorithm in R with data on a 1024×1024 grid and on the 32×32 grid. The 1024×1024 grid is dense enough that the approximate contour can be considered to be the true contour of g . Figure 6 is a close up of the contour near the saddle point.

The PAT contour is almost indistinguishable from a standard contour on a 1024×1024 grid. It is comparable to the standard contour on a 32×32 grid except near the saddle point, where the PAT contour is more accurate. PAT correctly finds that the contour crosses itself, even though the contour of the pilot approximation does not. Moreover, PAT is more accurate than the standard contouring algorithm on the same grid, in part because PAT finds the zeros of g rather than approximating them by linear interpolation. Indeed, if we treat the contour on the 1024×1024 grid as ground truth, the standard contouring on a 32×32 grid misclassifies 0.38% of the domain, while PAT misclassifies only 0.12% of the domain (the area of the symmetric difference was computed using the `gpc1ib` library in R). Repeated random samples of 100 points generally found none that were misclassified by the PAT contour (no sample point was in $\hat{\Lambda}_\tau \ominus \Lambda_\tau$). The corresponding 95% upper confidence bound on the fraction of Δ misclassified by the PAT contour is about 3%, rather larger than the true percentage, 0.12%, because the sample size is so small. Occasionally, a random sample of 100 points revealed one point misclassified by the PAT contour; the corresponding upper 95% confidence bound is 4.7%. By increasing the sample size, the upper confidence bound could be reduced.

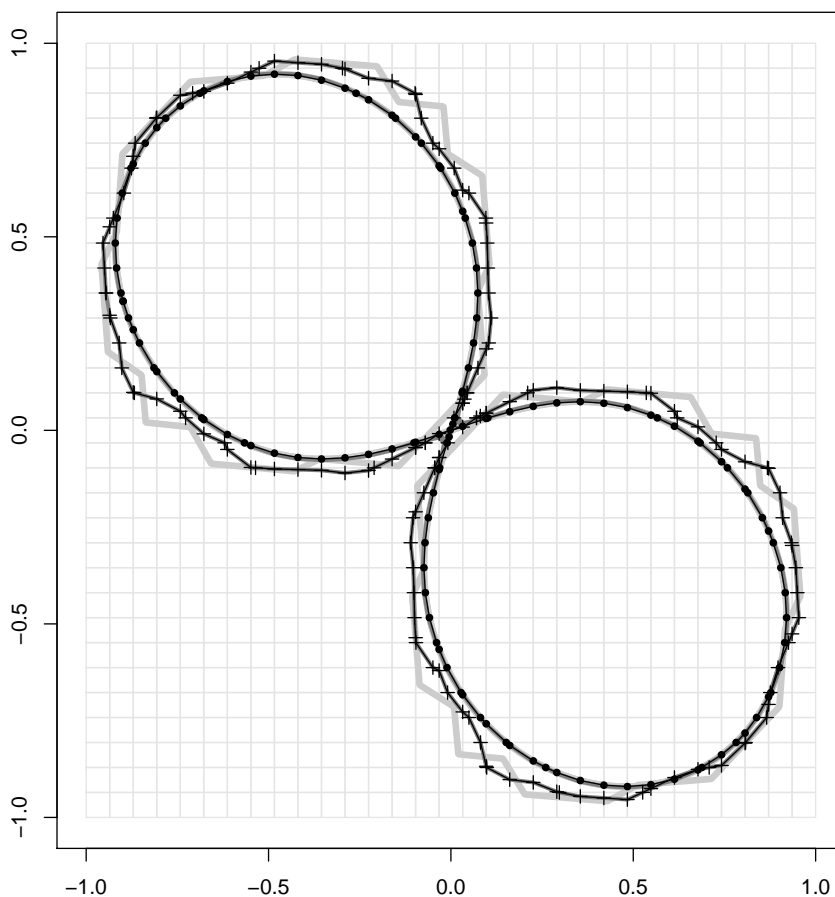


Figure 5. Results for Example 1: PAT contour $\hat{\gamma}_g$ of g (dots connected with black lines) and standard contour of \hat{g} (crosses connected with black lines) on a 32×32 grid. Contours of g and \hat{g} computed using a standard algorithm on a 1024×1024 grid are plotted as solid light gray curves. A standard contour on a 32×32 grid is plotted as dotted dark gray curves.

To compute the PAT contour required evaluating g 370 times (the pilot approximation \hat{g} was evaluated 1109 times). To build the pilot approximation \hat{g} , g was evaluated $9^2 = 81$ times, so g was evaluated 451 times in all. For comparison, computing the contour on a 32×32 grid using a standard contouring algorithm requires $32^2 = 1024$ evaluations of g . In this case our algorithm requires 56% fewer evaluations of g . Counting the additional 100 evaluations of g at randomly selected points, the total number of evaluations is still 46% less than the number required by the standard contouring algorithm—and provides greater accuracy and an internal measure of accuracy that the standard method lacks. The number of evaluations of g that PAT requires depends on the root finding algorithm and its tuning constants; we used the algorithm of Müller (1956).

Missing pieces To test ability of PAT to find a contour accurately even when the topology of the contour of the pilot approximation is wrong, we shifted the level τ

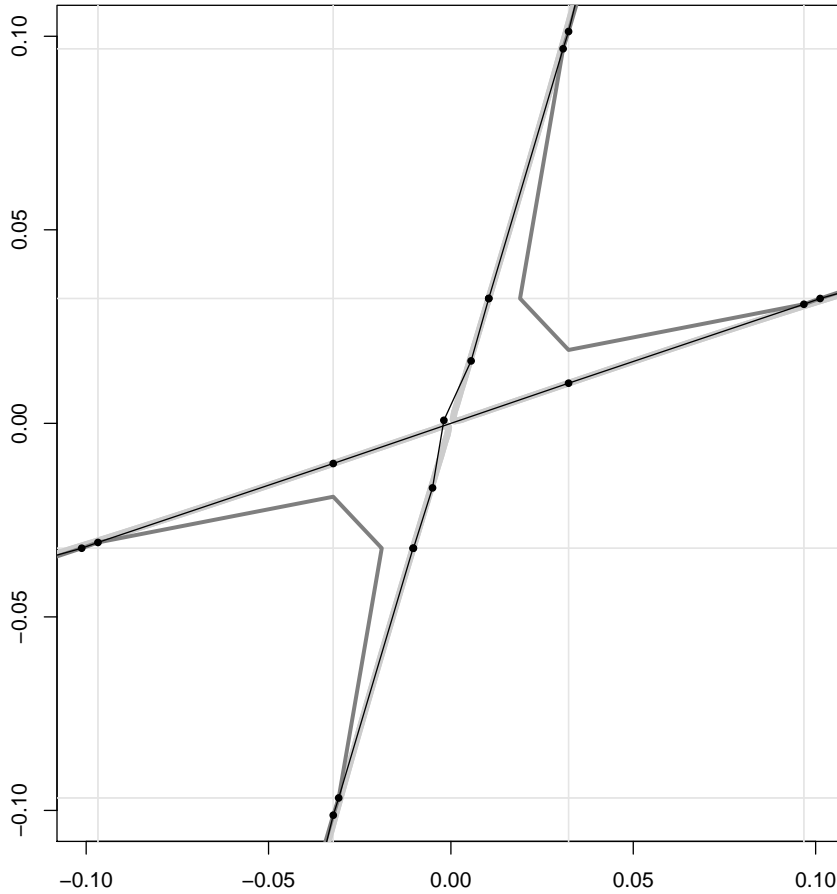


Figure 6. Close-up of the contour near the saddle point (the origin). The approximate PAT contour $\hat{\gamma}_g$ of g is plotted as dots connected with black lines, the standard contour is plotted as a solid light gray line (1024×1024 grid) and a solid light gray line (32×32 grid).

for which the contour is constructed by 0.005 (i.e. $\tau = g(0,0) + 0.005$). This makes the contour $\gamma_{g,\tau}$ consist of two closed curves. We also shifted the approximation \hat{g} by 0.045 so that the contour of the pilot approximation $\gamma_{\hat{g},\tau}$ is a single closed curve surrounding both pieces of the contour $\gamma_{g,\tau}$.

In this test the contour $\gamma_{g,\tau}$ has more pieces than $\gamma_{\hat{g},\tau}$, the contour of the pilot approximation. The method described in Section 3.4 finds the piece: The method checks for whole-cell “gaps” between the PAT contour $\hat{\gamma}_{g,\tau}$ and the approximate contour of the pilot approximation, $\hat{\gamma}_{\hat{g},\tau}$. In this test case, the piece of the PAT contour is inside the approximate contour, so the gaps are outside the PAT contour. Since g is below τ near but outside the PAT contour, the algorithm will detect that a piece of the contour is missing if it finds a point outside the traced piece at which $g > \tau$. To look for such a point, it evaluates g within the gap at the grid point where \hat{g} is largest, since, if \hat{g} is a good approximation to g , we expect g to be large there, too. If $g > \tau$ at that point, the algorithm searches the line segment between that point and the closest point at which g has been already evaluated (a vertex of the cell

intersected by the contour) to find a point where $g = \tau$. It then traces the (new) piece of the contour that passes through that root of $g - \tau$.

The PAT contour on a 32×32 grid is shown in Figure 7. Computing the PAT contour required 423 evaluations of g : $8^2 = 64$ to find the pilot approximation and 359 to trace the contour. If we consider the contour on the 1024×1024 grid to be ground truth, the standard contour on a 32×32 grid misclassifies 0.33% of the domain, while PAT misclassifies only 0.15% of the domain. Again, PAT gives a more accurate contour than the standard method, using far fewer evaluations of g .

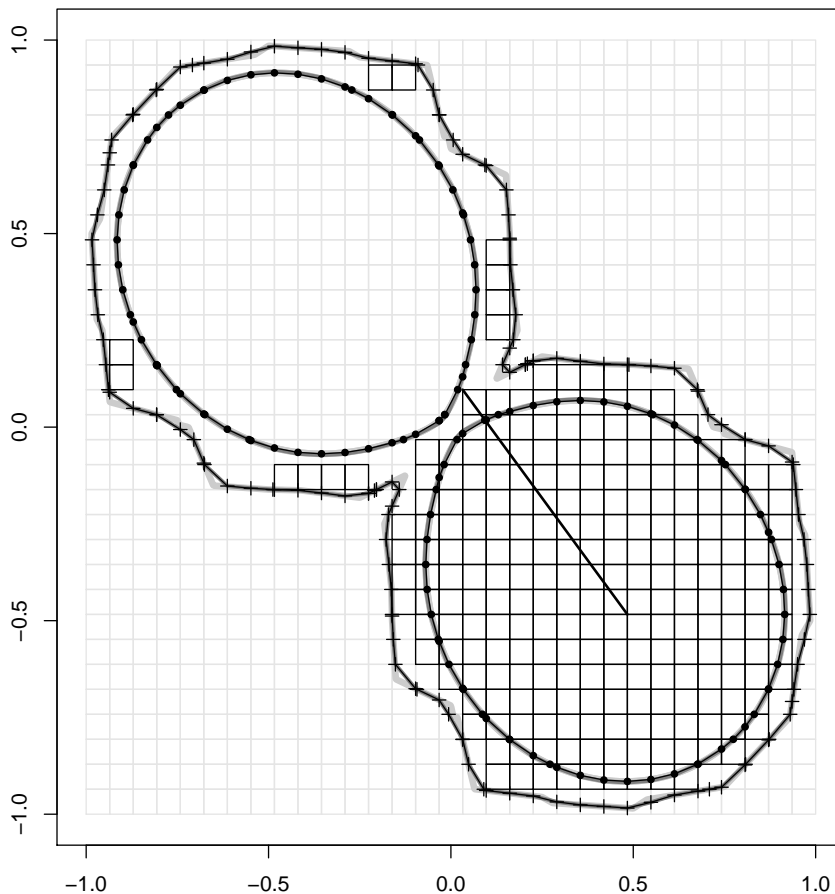


Figure 7. PAT contour for a shifted version of Example 1. In this example, the true contour has two pieces but the contour of the pilot approximation has only one. The algorithm identifies the cells marked in black are identified as a “gap” between $\hat{g}_{\hat{g}}$, the approximate contour of \hat{g} and \hat{g}_g , the PAT contour of g . The black line segment joins the point at which \hat{g} is largest and the closest point in the gap at which g has already been evaluated. The algorithm searches that segment for a root of $g - \tau$. The second piece of the PAT contour is traced starting at the root of $g - \tau$ on the black line segment. The solid and dotted gray curves are as described in Figure 5.

Workload scaling To investigate empirically how the number of function evaluations PAT requires scales, we used PAT to contour the saddle function using grids ranging

from 20×20 to 200×200 . Table 1 gives the results. PAT’s computational advantage over standard methods increases with the dimension of the grid. The number of evaluations of g for standard method scales like the number of points in the grid, n^2 , while the number of evaluations of g for PAT scales approximately as $O(n)$. The accuracy of PAT is substantially higher than that of standard methods on coarse grids, and slightly higher even for fine grids.

Table 1. Evaluations of g required for different grid sizes.

Grid Dimension	Evaluations of g		Saved	Relative Error
	PAT	Standard		
20×20	266 + 81	400	13%	2.43
30×30	355 + 81	900	51%	3.20
50×50	512 + 81	2,500	76%	1.49
100×100	910 + 81	10,000	90%	1.07
200×200	1814 + 81	40,000	95%	1.06

Table 2. Workload scaling for PAT and standard contouring for various grid dimensions, for the test function in figure 4. “Saved” is 1 minus the ratio of evaluations of g for PAT to evaluations of g for the standard method. “Relative error” is the ratio of the area of the symmetric difference of the true and approximate lower level sets for the standard contour to the area of that difference for the PAT contour. Numbers larger than 1.0 indicate that PAT is more accurate than the standard method.

4.2. Test case 2: confidence sets for viscosity and thermal conductivity of air flow

An ideal homogeneous gas flows from left to right in a 2-dimensional pipe, sketched in Figure 8. The gas is heated at the input boundary on the left with a known temperature profile. There is no heat source within the pipe. Once the flow is in steady state, the temperature of the gas is measured at the 101 points shown in Figure 8; the measurements have independent, identically distributed (iid) additive Gaussian noise with expected value zero. From the boundary condition and the temperature measurements, we want to estimate the viscosity and the thermal conductivity of the gas and to find the uncertainties of the estimates using a joint χ^2 confidence set, as described in Section 2.

We model the flow using Elmer finite element software (IT Center for Science (CSC) 2011). The diameter of the pipe has a step, which causes the flow to detach from the wall. If the viscosity is sufficiently small (i.e., if the Reynolds number is large), the flow may become turbulent, a possibility we ignore in this example.

Mathematical models The steady-state temperature distribution $T(\cdot)$ of the gas is modeled by the heat equation. Since there are no internal sources, this equation is:

$$\rho C (v \cdot \nabla T) - \nabla \cdot (\kappa \nabla T) = 0, \quad (7)$$

where C is heat capacity, v is the velocity field, and κ is thermal conductivity. The velocity field is modeled by the Navier-Stokes equation:

$$\rho(v \cdot \nabla)v - \nabla \cdot (2\mu\epsilon) + \nabla p = \rho f, \quad (8)$$

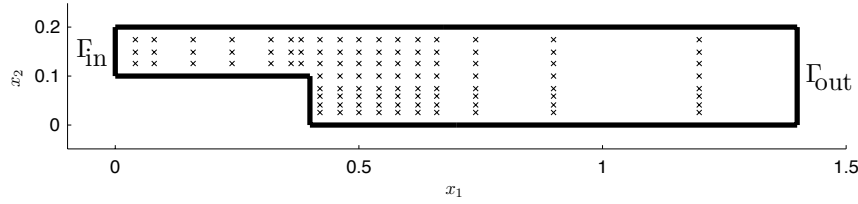


Figure 8. The computational domain for the gas flow example, a two-dimensional pipe whose diameter doubles abruptly. An ideal gas flows in through Γ_{in} on the left and out through Γ_{out} on the right. The inner wall of the pipe is the boundary Γ_{wall} (not labeled). Boundary conditions are imposed on temperature and velocity at Γ_{in} ; there is no flow across the pipe wall; and the velocity through Γ_{out} is purely horizontal. Steady-state temperature is measured with independent, additive, zero-mean Gaussian noise at the 101 points indicated by crosses.

where ρ is density, μ is viscosity, p is pressure, f is gravitational force, and ϵ is the infinitesimal strain tensor. The velocity and temperature boundary conditions at the input boundary vary quadratically with height. The velocities at the walls are zero and purely horizontal at the output boundary. The temperatures at the walls are fixed, and the flow satisfies the Neumann zero boundary condition at Γ_{out} .

The heat equation and the Navier-Stokes equation are solved as a coupled PDE problem using Elmer. The function f and its pilot approximation \hat{f} (see the notation in Section 2) are as follows. The function f maps (x_1, x_2) to HT , where T is the temperature distribution for a fixed viscosity of 10^{-x_1} Ns/m² and a fixed thermal conductivity of $10^{-2} \cdot x_2$ W/(m·K), and H is the matrix that interpolates linearly between points on the computational mesh to calculate T at the measurement points. The domain of f is $\Delta = [3, 4] \times [2, 3]$, which we verified experimentally contains every point in the confidence set. A logarithmic transformation of viscosity improves numerical stability, since the effect of viscosity on velocity and temperature is exponential. We scale the x_2 -axis to make the axes equally long, which improves computational efficiency. An approximation \hat{f} for f is formed in the same way using a very sparse mesh.

Results PAT was used to approximate the contour corresponding to the boundary of the 95% χ^2 confidence set (see Section 2). The function g was defined using (5) with $p = 2$ (χ^2 confidence sets). The pilot approximation \hat{g} is obtained similarly by replacing f with \hat{f} in Equation (5).

The computational mesh used to with evaluations of f contained 330 elements and 1099 nodes. For the pilot approximation \hat{f} , the mesh contained 108 elements and 389 nodes. This is approximately the fewest elements required for Elmer to be stable in this problem. The material parameters used in the computations are shown in Table 3. PAT contours were computed using a grid of 30×30 cells (31×31 grid points).

The simulated data were computed using Elmer with a dense mesh of 563 elements and 1830 nodes. Independent, identically distributed zero-mean Gaussian noise with standard deviation 2.79K (2% of the elevation in the temperature) was added to the simulated values.

To assess accuracy of the PAT confidence sets, we also computed the contour using

Table 3. Material Properties for Gas-Flow Simulations

Parameter	Value
viscosity	$1/5500 \text{ N}\cdot\text{s}/\text{m}^2$
thermal conductivity κ	$2.55 \cdot 10^{-2} \text{ W}/(\text{m}\cdot\text{K})$
heat capacity C	$1010 \text{ J}/(\text{kg}\cdot\text{K})$
specific heat ratio	1.4
reference pressure	10^5 Pa

the standard algorithm in R on a 256×256 grid. The PAT contours and reference contours are shown in Figure 9. The PAT contours are almost indistinguishable from contours computed by standard methods on the dense grid. Differences are visible only in sharp corners where the cell size is too large for a straight line segment to turn as rapidly as the actual contour.

Computing the PAT contour for the χ^2 confidence set required evaluating f 243 times and \hat{f} 1023 times. Thus the number of the evaluations of f was about a quarter of $31^2 = 961$, the minimum number of evaluations required to compute the contour on the a 31×31 grid using a standard contouring method. Moreover, the PAT confidence set is more accurate than the confidence set identified by a standard contouring method on a 31×31 grid: The relative area of $\hat{\Lambda}_\tau \ominus \Lambda_\tau$ for the PAT confidence set is 0.06% while the relative area for a standard contour on the same grid is 0.15%.

5. Conclusions

The Pilot Approximation Trajectory (PAT) algorithm is an efficient way to find one or more contours of a function g that is expensive to evaluate. PAT uses an inexpensive pilot approximation of g to predict the location of the contour. This can reduce the number of evaluations of g required to contour g to a given level of accuracy. PAT uses an iterative method to find points on the contour, and takes advantage of the fact that g can be evaluated anywhere, not just the vertices of a regular grid. The computational advantage of PAT increases with the dimension of the grid used in the standard method. In examples, the number of function evaluations that PAT requires scales roughly as the square-root of the number of grid points, while conventional methods scale linearly with the number of grid points. The savings can be enormous.

We give numerical examples illustrate the utility and efficiency of PAT. The first example involves contouring a function that has a saddle point near the contour level. The second example is to find a confidence set for an unknown 2-dimensional parameter in a nonlinear inverse problem. In both examples, PAT contours are more accurate than standard contours and require only a fraction as many function evaluations.

PAT may be useful in a variety of problems where evaluating the function to contour is expensive, for example when evaluating the function requires conducting a physical experiment or running complex coupled PDE solvers. As our second example shows, PAT may be useful for inference in inverse problems with two unknowns.

Since PAT uses values of g at cell vertices to determine whether the contour crosses an edge, PAT cannot tell whether an edge is intersected more than once. Reducing cell size ameliorates this issue, but increases the number of evaluations of g . A more sophisticated approach might split cells adaptively depending on the local smoothness

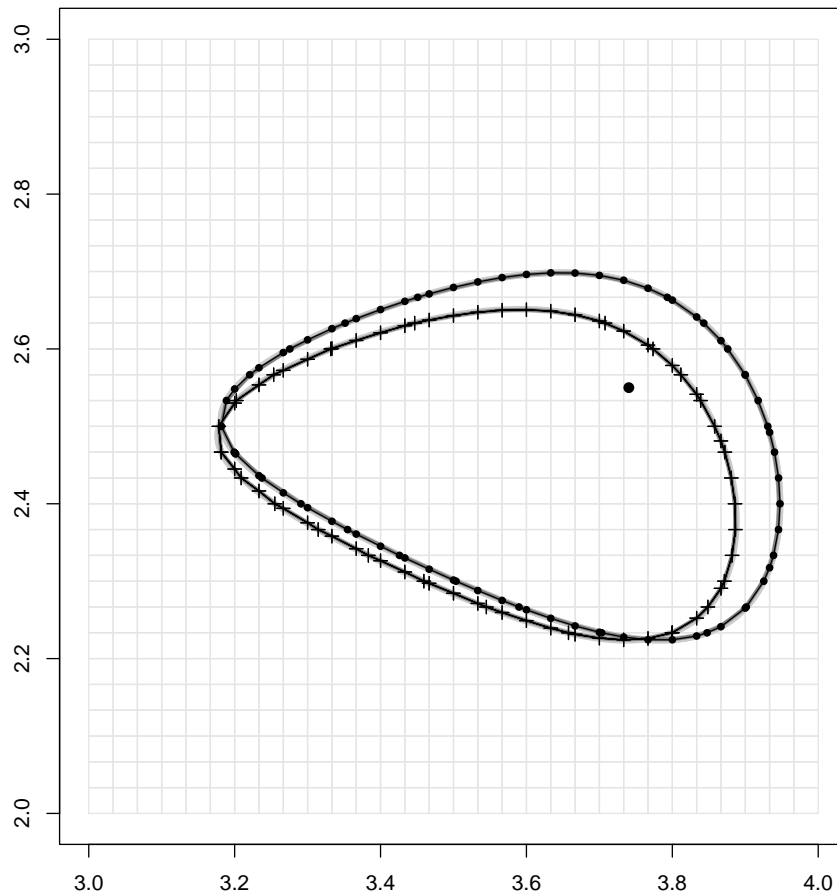


Figure 9. The χ^2 confidence set. The black dots connected with lines are the PAT contour of the actual misfit function. The crosses are the contour of the pilot approximation for PAT. The gray solid lines are contours computed using a standard algorithm on a 256×256 grid.

of g . Aramini (1980) proposes adapting grid size locally using the Laplacian of g or the value of g at the centers of cells. Similar techniques might be used with PAT. For PAT, a criterion for splitting cells might be based on \hat{g} (e.g., tracing the contour of \hat{g} on a dense grid to see whether it crosses any edge more than once).

As presented here, PAT is for functions of two variables. PAT could be extended to functions of three variables using an approach similar to Marching cubes (Lorensen & Cline 1987). How to extend PAT to dimensions higher than three is an open question.

Acknowledgments

JMJH was supported by Osk. Huttunen foundation, Jenny & Artturi Wihuri foundation and Finnish Cultural foundation, and the Academy of Finland (application number 213476, the Finnish Programme for Centres of Excellence in Research 2006-2011). PBS received support from Lawrence Livermore National Laboratory Grant

B585264.

References

- Agarwal, P., Arge, L., Mølhave, T. & Sadri, B. (2008), I/O-efficient algorithms for computing contours on a terrain, *in* ‘Proceedings of the twenty-fourth annual symposium on Computational geometry’, ACM, New York, NY, USA, pp. 129–138.
- Aramini, M. (1980), ‘Implementation of an improved contour plotting algorithm’, Master’s thesis, University of Illinois.
- Bryan, B. & Schneider, J. (2008), Actively learning level-sets of composite functions, *in* ‘ICML 2008: Proceedings of the 25th International Conference on Machine Learning’, ACM.
- Bryan, B., Schneider, J., Nichol, R. C., Miller, C. J., Genovese, C. R. & Wasserman, L. (2005), Active learning for identifying function threshold boundaries, *in* ‘NIPS Conference Proceedings’, NIPS.
- Cottafava, G. & Moli, G. L. (1969), ‘Automatic contour plotting’, *Communications of the ACM* **12**(7), 386–391.
- Higdon, D., Klein, R., Anderson, M., Berliner, M., Covey, C., Ghattas, O., Graziani, C., Habib, S., Seager, M., Sefcik, J., Stark, P. & Stewart, J. (2010), Panel report on uncertainty quantification and error analysis, *in* ‘Scientific Grand Challenges in National Security: The Role of Computing at the Extreme Scale’, U.S. Department of Energy Office of Advanced Scientific Computing Research and National Nuclear Security Administration, http://extremecomputing.labworks.org/nationalsecurity/PNNL_19379_NNSAFinalReport.pdf.
- IT Center for Science (CSC) (2011), ‘Elmer: Open source finite element software for multiphysical problems’, <http://www.csc.fi/english/pages/elmer>.
- Kaipio, J. & Somersalo, E. (2005), *Statistical and Computational Inverse Problems*, Applied Mathematical Sciences 160, Springer, N.Y.
- Lorensen, W. E. & Cline, H. E. (1987), ‘Marching cubes: A high resolution 3d surface construction algorithm’, *Computer Graphics* **21**.
- McLain, D. (1974), ‘Drawing contours from arbitrary data points’, *The Computer Journal* **17**(4), 318–324.
- Müller, D. E. (1956), ‘A method for solving algebraic equations using an automatic computer’, *Mathematical Tables and Other Aids to Computation* **10**, 208–215.
- Scott, C. & Davenport, M. (2007), ‘Regression level set estimation via cost-sensitive classification’, *IEEE Transactions on signal processing* **55**(6), 2752–2757.
- Sethian, J. (1999), *Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science*, Cambridge University Press.
- Stark, P. (1992), ‘Inference in infinite-dimensional inverse problems: Discretization and duality’, *Journal of Geophysical Research* **97**(B10), 14055–14082.
- Stark, P. (2008), ‘Generalizing resolution’, *Inverse Problems* **24**, 034014.
- Suffern, K. (1990), ‘Quadtree algorithms for contouring functions of two variables’, *The Computer Journal* **33**(5), 402–407.
- van Kreveld, M. J., van Oostrum, R., Bajaj, C. L., Pascucci, V. & Schikore, D. (1997), Contour trees and small seed sets for isosurface traversal, *in* ‘SCG ’97 Proceedings of the thirteenth annual symposium on Computational geometry’, ACM, New York, NY, USA, pp. 212–220.
- Willett, R. & Nowak, R. (2005), Level set estimation via trees, *in* ‘Proceedings of International Conference of Acoustics, Speech and Signal Processing’, pp. 1085–1088.