chapter can be applied. Any intersections that are found can be used as approximations to line-curve intersections if the application is willing to accept that the polyline is a suitable approximation to the curve. However, the points of intersection might be used as an attempt to localize the search for actual points of intersection on the curve. For example, if a line-polyline intersection occurred on the segment $\langle X(s_i), X(s_{i+1})\rangle$, the next step could be to search for a root of $q(s) = 0$ in the interval $[s_i, s_{i+1})$.

7.4.3 Hierarchical Bounding

The algebraic method mentioned earlier always incurs the cost of root finding for a polynomial equation. Presumably the worst case is that after spending the computer time to find any real-valued roots of $q(s) = 0$, there are none; the line and polynomial curve do not intersect. An application might want to reduce the cost for determining there is no intersection by providing coarser-level tests in hopes of an "early out" from the intersection testing. Perhaps more important is that if the application will perform a large number of line-curve intersection tests with different lines, but the same curve, the total cost of polynomial root finding can be prohibitive. Some type of curve preprocessing certainly can help to reduce the costs.

One coarse-level test involves maintaining a bounding polygon for the curve. In particular, if the curve is built from control points and the curve lies in the convex hull of the control points, an intersection test is first applied to the line and the convex hull (a convex polygon). If they do not intersect, then the line and curve do not intersect. If the line and polygon do intersect, then the application proceeds to the more expensive line-curve test.

An alternative is to use an axis-aligned bounding rectangle for the curve. The line-rectangle intersection test is quite inexpensive and is discussed in Section 7.7 on separating axes. If the application is willing to allow a few more cycles in hopes of an early-out no-intersection test, a variation of the algorithm is to construct a hierarchy of bounding boxes, each level providing a better fit (in some sense) than the previous level. Moreover, if the line does not intersect a bounding box at some level, then there is no point in processing further levels below the node of that box since the line cannot intersect the curve in that localized region. Figure 7.3 illustrates the idea. A curve is shown with a two-level hierarchy. The line intersects the top-level box, so the next level of the hierarchy must be analyzed. The line intersects the left child box at the next level, so further intersection tests are needed using either line-box or line-curve tests. The line does not intersect the right child box at the next level, so the line cannot intersect the curve contained in that box. No further processing of that subtree of the hierarchy is needed.

The main question, of course, is how do you construct an axis-aligned bounding box for a curve? For special classes of curves, specifically Bézier curves, this is not difficult. An axis-aligned bounding box for the curve that is not usually of smallest area is constructed for the control points of the curve. A hierarchy of boxes can be built by subdividing the curve and fitting boxes to the control points that correspond to each

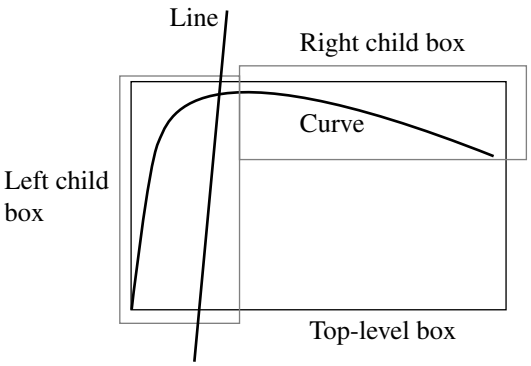

Figure 7.3 Line-curve intersection testing using a hierarchy of bounding boxes.

subcurve. For polynomial curves in general, finding the smallest-area axis-aligned bounding box appears to be as complicated as the algorithm for finding intersections of the line and the curve. The extent of the box in the x-direction is determined by the x-extreme points on the curve. The x-extreme points are characterized by having vertical tangents to the curve. Mathematically, $(x(t), y(t))$ has a vertical tangent if $x'(t) = 0$. Similarly, the y-extreme points are characterized by having horizontal tangents to the curve where $y'(t) = 0$. Each derivative equation is a polynomial equation that can be solved by numerical methods, but proceeding this way invalidates the goal of using a bounding box to avoid expensive root finding in regions where the line does not intersect the curve. This might not be an issue if the original curve is cubic, in which case the derivative equations can be solved using the quadratic formula. This is also not an issue if the application plans on testing for intersections between multiple lines and a single curve. The preprocessing costs for computing a bounding box are negligible compared to the total costs of root finding for the line-curve intersection tests.

7.4.4 Monotone Decomposition

Now suppose that $x'(t) \neq 0$ for any $t \in [t_{\min}, t_{\max}]$. The curve is monotonic in x, either strictly increasing or strictly decreasing. In this special case, the x-extents for the axis-aligned bounding box correspond to $x(t_{\min})$ and $x(t_{\max})$. A similar argument applies if $y'(t) \neq 0$ on the curve domain. Generally, if $x'(t) \neq 0$ and $y'(t) \neq 0$ for $t \in [a, b] \subseteq [t_{\min}, t_{\max}]$, then the curve segment is monotonic and the axis-aligned bounding box is determined by the points $(x(a), y(a))$ and $(x(b), y(b))$. Determining that a derivative equation does not have roots is an application of Sturm sequences, as discussed in Section A.5.

The idea now is to subdivide the curve using a simple bisection on the parameter interval with the goal of finding monotonic curve segments. If $[a, b]$ is a subinterval in the bisection for which the curve is monotonic, no further subdivision occurs; the axis-aligned bounding box is known for that segment. Ideally, after a few levels of bisection we obtain a small number of monotonic segments and their corresponding bounding boxes. The line-box intersection tests can be applied between the line and each box in order to cull out monotone segments that do not intersect the line, but if the number of boxes is large, you can always build a hierarchy from the bottom up by treating the original boxes as leaf nodes of a tree, then grouping a few boxes at a time to construct parent nodes. The bounding box of a parent node can be computed to contain the bounding boxes of its children. The parents themselves can be grouped, the process eventually leading to the root node of the tree with a single bounding box. The method of intersection illustrated in Figure 7.3 may be applied to this tree.

A recursive subdivision may be applied to find monotone segments. The recursion requires a stopping condition that should be chosen carefully. If the derivative equations $x'(t) = 0$ and $y'(t) = 0$ both indicate zero root counts on the current interval, then the curve is monotonic on that interval and the recursion terminates. If one of the equations has a single root on the current interval and the other does not, a subdivision is applied. It is possible that the subdivision t-value is the root itself, in which case both subintervals will report a root when there is only a single root. For example, consider $(x(t), y(t)) = (t, t^2)$ for $t \in [-1, 1]$. The derivative equation $x'(t) = 0$ has no solution since $x'(t) = 1$ for all t, but $y'(t) = 2t = 0$ has one root on the interval. The subdivision value is $t = 0$. The equation $y'(t) = 0$ has one root on the subinterval $[-1, 0]$ and one root on the subinterval $[0, 1]$, but these are the same root. The end points of subintervals should be checked to avoid deeper recursions than necessary. The typical case, though, is that a root of a subinterval occurs in the interior of the interval. Once a subinterval that has a single interior root is found, a robust bisection can be applied to find the root and subdivide the subinterval at the root. The recursion terminates for that subinterval.

In an application that will perform intersection queries with multiple lines but only one curve, the monotone segments can be found as a preprocessing step by solving $x'(t) = 0$ and $y'(t) = 0$ using numerical root finders.

7.4.5 Rasterization

A raster approach may be used, even though it is potentially quite expensive to execute. An axis-aligned bounding box $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$ is constructed to contain the curve. An $N \times M$ raster is built to represent the box region. The grid points are uniformly chosen as (x_i, y_j) for $0 \le i < N$ and $0 \le j < M$. That is, $x_i = x_{\min} + (x_{\max} - x_{\min})i/(N - 1)$ and $y_j = y_{\min} + (y_{\max} - y_{\min})j/(M - 1)$. Both the line and the curve are drawn into the raster. The step size for the parameter of the curve should be chosen to be small enough so that as the curve is sampled you generate adjacent raster values, potentially with a raster cell drawn multiple times

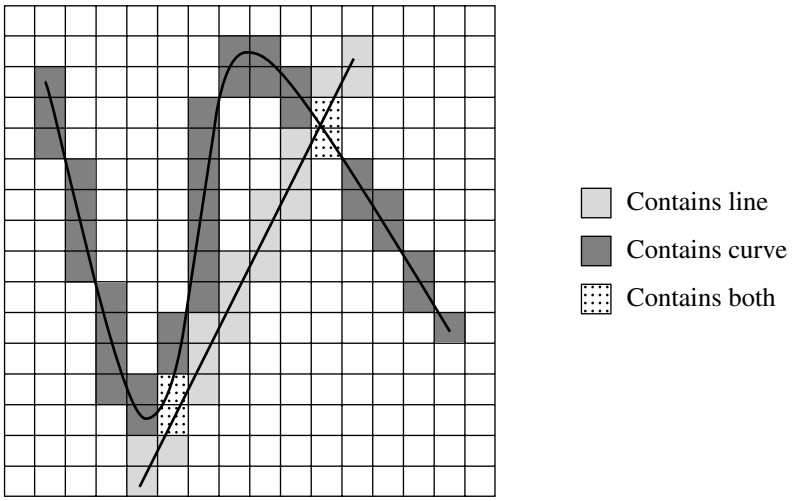

Figure 7.4 A line and a curve rasterized on a grid that is initially zero. The line is rasterized by or-ing the grid with the mask 1 (light gray). The curve is rasterized by or-ing the grid with the mask 2 (dark gray). Grid cells that contain both the line and the curve have a value 3 (dotted).

because multiple curve samples fall inside that cell. The overdraw can be reduced by sampling the curve based on arc length rather than the curve parameter. If the raster is initialized with 0, the line is drawn by or-ing the pixels with 1, and the curve is drawn by or-ing the pixels with 2. The pixels that are associated with line-curve intersections are those with a final value of 3. Figure 7.4 illustrates this.

The effectiveness of this method depends on how well the grid cell size is chosen. If it is chosen to be too large, the line and curve can pass through the same pixel, yet not intersect. The rasterization method reports an intersection when there is none. This situation is shown in Figure 7.4 in the lower-left portion of the grid. If the cell size is chosen to be too small, a lot of time is consumed in rasterizing the curve, especially in pixels that the line does not intersect.

Just as in the polyline approach, the application can choose to accept the pixel values as approximations to the actual line-curve intersections. If more accuracy is desired, the pixels tagged as 3 (and possibly immediate neighbors) can be used as a localization of where the intersections occur. If a contiguous block of pixels is tagged, such as is shown in the upper right of the grid in Figure 7.4, and if the application chooses to believe the block occurs because of a single intersection of curves, a suitable choice for the approximation is the average of pixel locations. If the application chooses not to accept the pixel values as approximations, then it can

store the original line and curve parameters for samples occurring in a pixel with that pixel. Those parameter values can be used to start a search for intersections using a numerical root finder or a numerical distance calculator.

7.5 Quadratic Curves

We present a general algebraic method for computing intersection points between two curves defined implicitly by quadratic curves. The special case of circular components is also presented because it handles intersection between circular arcs. Variations on computing intersection points of two ellipses are also presented here as an illustration of how you might go about handling the more general problem of intersection between two curves, each defined as a level curve for a particular function.

7.5.1 General Quadratic Curves

Given two curves implicitly defined by the quadratic equations $F(x, y) = \alpha_{00} + \alpha_{10}x + \alpha_{01}y + \alpha_{20}x^2 + \alpha_{11}xy + \alpha_{02}y^2 = 0$ and $G(x, y) = \beta_{00} + \beta_{10}x + \beta_{01}y + \beta_{20}x^2 + \beta_{11}xy + \beta_{02}y^2 = 0$, the points of intersection can be computed by eliminating y to obtain a fourth-degree polynomial equation $H(x) = 0$. During the elimination process, y is related to x via a rational polynomial equation, $y = R(x)$. Each root $\bar{x}$ of $H(x) = 0$ is used to compute $\bar{y} = R(\bar{x})$. The pair $(\bar{x}, \bar{y})$ is an intersection point of the two original curves.

EXAMPLE The equation $x^2 + 6y^2 - 1 = 0$ defines an ellipse whose center is at the origin. The equation $2(x - 2y)^2 - (x + y) = 0$ determines a parabola whose vertex is the origin. Figure 7.5 shows the plots of the two curves. The ellipse equation is rewritten as $y^2 = (1 - x^2)/6$. Substituting this in the parabola equation produces

$$\begin{aligned} 0 &= 2x^2 - 8xy + 8y^2 - x - y = 2x^2 - 8xy + 8(1 - x^2)/6 - x - y \\ &= -(8x + 1)y + (2x^2 - 3x + 4)/3 \end{aligned}$$

This may be solved for

$$y = \frac{2x^2 - 3x + 4}{3(8x + 1)} =: R(x)$$

Replacing this in the ellipse equation produces

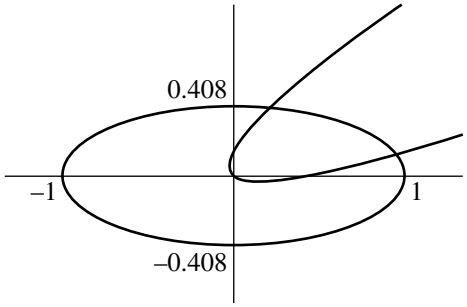

Figure 7.5 Intersections of an ellipse and a parabola.

$$\begin{aligned}
0 &= x^2 + 6y^2 - 1 \\
&= x^2 + 6\left(\frac{2x^2 - 3x + 4}{3(8x+1)}\right)^2 - 1 \\
&= \frac{9(8x+1)^2(x^2-1) + 6(2x^2-3x+4)^2}{9(8x+1)^2} \\
&= \frac{200x^4 + 24x^3 - 139x^2 - 96x + 29}{3(8x+1)^2}
\end{aligned}$$

Therefore, it is necessary that

$$0 = 200x^4 + 24x^3 - 139x^2 - 96x + 29 =: H(x) \quad \blacksquare$$

The polynomial equation $H(x) = 0$ has two real-valued roots, $x_0 \doteq 0.232856$ and $x_1 \doteq 0.960387$. Replacing these in the rational polynomial for y produces $y_0 = R(x_0) \doteq 0.397026$ and $y_1 = R(x_1) \doteq 0.113766$. The points (x_0, y_0) and (x_1, y_1) are the intersection points for the ellipse and parabola.

The general method of solution of two polynomial equations is discussed in detail in Section A.2.

As with any root finder, numerical problems can arise when a root has even multiplicity or the derivative of the function near the root is small in magnitude. These problems tend to arise geometrically when the two curves have an intersection point for which the angle between tangent lines to the curves at the point is nearly zero. If you need extreme accuracy and do not want to miss intersection points, you will need your root finder to be quite robust at the expense of some extra computational time.

If the application only needs to know if the curves intersect, but not where, then the method of Sturm sequences for root counting can be applied to $H(x) = 0$. The method is discussed in Section A.5.

7.5.2 Circular Components

Let the two circles be represented by $\|X - C_i\|^2 = r_i^2$ for $i = 0, 1$. The points of intersection, if any, are determined by the following construction. Define $\vec{u} = C_1 - C_0 = (u_0, u_1)$. Define $\vec{v} = (u_1, -u_0)$. Note that $\|\vec{u}\|^2 = \|\vec{v}\|^2 = \|C_1 - C_0\|^2$ and $\vec{u} \cdot \vec{v} = 0$. The intersection points can be written in the form

$$X = C_0 + s\vec{u} + t\vec{v} \tag{7.1}$$

or

$$X = C_1 + (s - 1)\vec{u} + t\vec{v} \tag{7.2}$$

where s and t are constructed by the following argument. Substituting Equation 7.1 into $\|X - C_0\|^2 = r_0^2$ yields

$$(s^2 + t^2)\|\vec{u}\|^2 = r_0^2 \tag{7.3}$$

Substituting Equation 7.2 into $\|X - C_1\|^2 = r_1^2$ yields

$$((s - 1)^2 + t^2)\|\vec{u}\|^2 = r_1^2 \tag{7.4}$$

Subtracting Equations 7.3 and 7.4 and solving for s yields

$$s = \frac{1}{2}\left(\frac{r_0^2 - r_1^2}{\|\vec{u}\|^2} + 1\right) \tag{7.5}$$

Replacing Equation 7.5 into Equation 7.3 and solving for t^2 yields

$$\begin{aligned} t^2 &= \frac{r_0^2}{\|\vec{u}\|^2} - s^2 = \frac{r_0^2}{\|\vec{u}\|^2} - \frac{1}{4}\left(\frac{r_0^2 - r_1^2}{\|\vec{u}\|^2} + 1\right)^2 \\ &= -\frac{(\|\vec{u}\|^2 - (r_0 + r_1)^2)(\|\vec{u}\|^2 - (r_0 - r_1)^2)}{4\|\vec{u}\|^4} \end{aligned} \tag{7.6}$$

In order for there to be solutions, the right-hand side of Equation 7.6 must be nonnegative. Therefore, the numerator is negative:

$$(\|\vec{u}\|^2 - (r_0 + r_1)^2)(\|\vec{u}\|^2 - (r_0 - r_1)^2) \le 0 \tag{7.7}$$

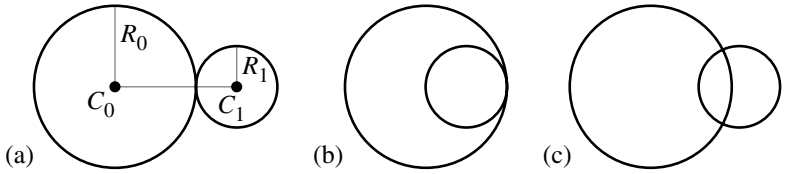

Figure 7.6 Relationship of two circles, $\vec{u} = C_1 - C_0$: (a) $\|\vec{u}\| = |r_0 + r_1|$; (b) $\|\vec{u}\| = |r_0 - r_1|$; (c) $|r_0 - r_1| < \|\vec{u}\| < |r_0 + r_1|$.

If $w = \|\vec{u}\|$, the left-hand side of Inequality 7.7 defines a quadratic function of w, the graph being a parabola that opens upwards. The roots are $w = |r_0 - r_1|$ and $w = |r_0 + r_1|$. For the quadratic function to be negative, only values of w between the two roots are allowed. Inequality 7.7 is therefore equivalent to

$$|r_0 - r_1| \le \|\vec{u}\| \le |r_0 + r_1| \tag{7.8}$$

If $\|\vec{u}\| = |r_0 + r_1|$, each circle is outside the other circle, but just tangent. The point of intersection is $C_0 + (r_0/(r_0 + r_1))\vec{u}$. If $\|\vec{u}\| = |r_0 - r_1|$, the circles are nested and just tangent. The circles are the same if $\|\vec{u}\| = 0$ and $r_0 = r_1$; otherwise the point of intersection is $C_0 + (r_0/(r_0 - r_1))\vec{u}$. If $|r_0 - r_1| < \|\vec{u}\| < |r_0 + r_1|$, then the two circles intersect in two points. The s-value from Equation 7.5 and the t-values from taking the square root in Equation 7.6 can be used to compute the intersection points as $C_0 + s\vec{u} + t\vec{v}$. Figure 7.6 shows the various relationships for the two circles.

If either or both circular components are arcs, the circle-circle points of intersection must be tested if they are on the arc (or arcs) using the circular-point-on-arc test described earlier in this chapter.

7.5.3 Ellipses

The algebraic method discussed earlier for testing/finding points of intersection applies, of course, to ellipses since they are implicitly defined by quadratic equations. In some applications, more information is needed other than just knowing points of intersection. Specifically, if the ellipses are used as bounding regions, it might be important to know if one ellipse is fully contained in another. This information is not provided by the algebraic method applied to the two quadratic equations defining the ellipses. The more precise queries for ellipses $\mathcal{E}_0$ and $\mathcal{E}_1$ are

- Do $\mathcal{E}_0$ and $\mathcal{E}_1$ intersect?
- Are $\mathcal{E}_0$ and $\mathcal{E}_1$ separated? That is, does there exist a line for which the ellipses are on opposite sides?
- Is $\mathcal{E}_0$ properly contained in $\mathcal{E}_1$, or is $\mathcal{E}_1$ properly contained in $\mathcal{E}_0$?

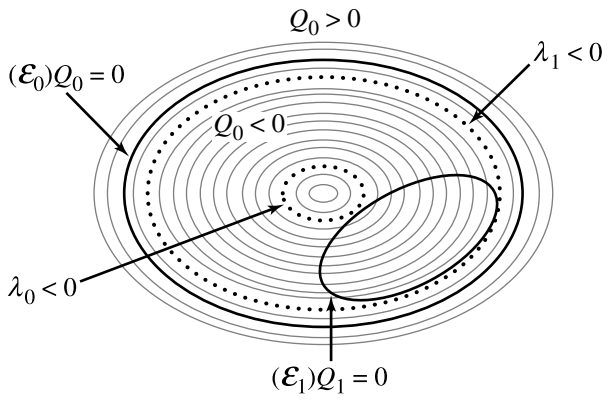

Figure 7.7 $\mathcal{E}_1$ is contained in $\mathcal{E}_0$. The maximum $\mathcal{E}_0$ level curve value λ_1 for $\mathcal{E}_1$ is negative.

Let the ellipse $\mathcal{E}_i$ be defined by the quadratic equation $Q_i(X) = X^T \mathbf{A}_i X + B_i^T X + c_i$ for $i = 0, 1$. It is assumed that the $\mathbf{A}_i$ are positive definite. In this case, $Q_i(X) < 0$ defines the inside of the ellipse, and $Q_i(X) > 0$ defines the outside.

The discussion focuses on level curves of the quadratic functions. Section A.9.1 provides a discussion of level sets of functions. All level curves defined by $Q_0(x, y) = \lambda$ are ellipses, except for the minimum (negative) value λ for which the equation defines a single point, the center of every level curve ellipse. The ellipse defined by $Q_1(x, y) = 0$ is a curve that generally intersects many level curves of Q_0. The problem is to find the minimum level value λ_0 and maximum level value λ_1 attained by any (x, y) on the ellipse $\mathcal{E}_1$. If $\lambda_1 < 0$, then $\mathcal{E}_1$ is properly contained in $\mathcal{E}_0$. If $\lambda_0 > 0$, then $\mathcal{E}_0$ and $\mathcal{E}_1$ are separated or $\mathcal{E}_1$ contains $\mathcal{E}_0$. Otherwise, $0 \in [\lambda_0, \lambda_1]$ and the two ellipses intersect. Figures 7.7, 7.8, and 7.9 illustrate the three possibilities. The figures show the relationship of one ellipse $\mathcal{E}_1$ to the level curves of another ellipse $\mathcal{E}_0$.

This can be formulated as a constrained optimization that can be solved by the method of Lagrange multipliers (see Section A.9.3): Optimize $Q_0(X)$ subject to the constraint $Q_1(X) = 0$. Define $F(X, t) = Q_0(X) + t\,Q_1(X)$. Differentiating with respect to the components of X yields $\vec{\nabla} F = \vec{\nabla} Q_0 + t\vec{\nabla} Q_1$. Differentiating with respect to t yields $\partial F/\partial t = Q_1$. Setting the t-derivative equal to zero reproduces the constraint $Q_1 = 0$. Setting the X-derivative equal to zero yields $\vec{\nabla} Q_0 + t\vec{\nabla} Q_1 = \vec{0}$ for some t. Geometrically this means that the gradients are parallel.

Note that $\vec{\nabla} Q_i = 2\mathbf{A}_i X + B_i$, so

$$\vec{0} = \vec{\nabla} Q_0 + t\vec{\nabla} Q_1 = 2(\mathbf{A}_0 + t\mathbf{A}_1)X + (B_0 + tB_1)$$

Formally solving for X yields

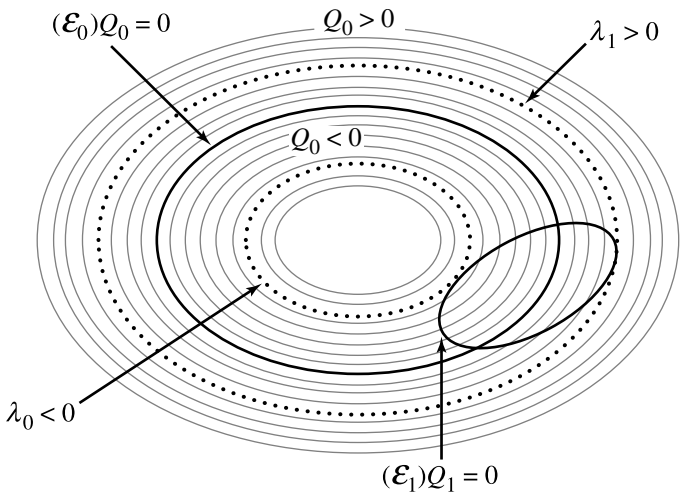

Figure 7.8 $\mathcal{E}_1$ transversely intersects $\mathcal{E}_0$. The minimum $\mathcal{E}_0$ level curve value λ_0 for $\mathcal{E}_1$ is negative; the maximum value λ_1 is positive.

$$X = -\frac{1}{2}(\mathbf{A}_0 + t\mathbf{A}_1)^{-1}(B_0 + tB_1) = \frac{1}{\delta(t)}\vec{Y}(t)$$

where $\mathbf{A}_0 + t\mathbf{A}_1$ is a matrix of linear polynomials in t and $\delta(t)$ is its determinant, a quadratic polynomial. The components of $\vec{Y}(t)$ are quadratic polynomials in t. Replacing this in $Q_1(X) = 0$ yields

$$p(t) \coloneqq \vec{Y}(t)^T \mathbf{A}_1 \vec{Y}(t) + \delta(t) B_1^T \vec{Y}(t) + \delta(t)^2 C_1 = 0 \tag{7.9}$$

a quartic polynomial in t. The roots can be computed, the corresponding values of X computed, and $Q_0(X)$ evaluated. The minimum and maximum values are stored as λ_0 and λ_1, and the earlier comparisons with zero are applied.

This method leads to a quartic polynomial, just as the original algebraic method for finding intersection points did. But the current style of query does answer questions about the relative positions of the ellipses (separated or proper containment) whereas the original method does not.

EXAMPLE Consider $Q_0(x, y) = x^2 + 6y^2 - 1$ and $Q_1(x, y) = 52x^2 - 72xy + 73y^2 - 32x - 74y + 28$. Figure 7.10 shows the plots of the two ellipses. The various parameters are