CALCULATING ELLIPSE OVERLAP AREAS

Gary B. Hughes

California Polytechnic State University
Statistics Department
San Luis Obispo, CA 93407-0405, USA

Mohcine Chraibi

Jülich Supercomputing Centre
Forschungszentrum Jülich GmbH
D-52425 Jülich, Germany

Abstract. We present a general algorithm for finding the overlap area between two ellipses. The algorithm is based on finding a segment area (the area between an ellipse and a secant line) given two points on the ellipse. The Gauss-Green formula is used to determine the ellipse sector area between two points, and a triangular area is added or subtracted to give the segment area. For two ellipses, overlap area is calculated by adding the areas of appropriate sectors and polygons. Intersection points for two general ellipses are found using Ferrari's quartic formula to solve the polynomial that results from combining the two ellipse equations. All cases for the number of intersection points (0, 1, 2, 3, 4) are handled. The algorithm is implemented in c-code, and has been tested with a range of input ellipses. The code is efficient enough for use in simulations that require many overlap area calculations.

1. **Introduction.** Ellipses are useful in many applied scenarios, and in widely disparate fields. In our research, which happens to be in two very different areas, we have encountered a common need for efficiently calculating the overlap area between two ellipses.

In one case, the design for a solar calibrator on-board an orbiting satellite required an efficient algorithm for ellipse overlap area. Imaging systems aboard satellites rely on semi-conductor detectors whose performance changes over time due to many factors. To produce consistent data, some means of calibrating the detectors is required; see, e.g., [1]. Some systems use the sun as a light source for calibration. In a typical solar calibrator, incident sunlight passes through an attenuator grating and impinges on a diffuser plate, which is oriented obliquely to the attenuator grating. The attenuator grating is a pattern of circular openings. When sunlight passes through the circular openings, projections of the circles onto the oblique diffuser plate become small ellipses. The projection of the large circular entrance aperture on the oblique diffuser plate is also an ellipse. The total incident light on the calibrator is proportional to the sum of all the areas of the smaller ellipses that are contained within the larger entrance aperture ellipse. However, as the calibration process proceeds, the satellite is moving through its orbit, and the angle

---

Key words and phrases. Ellipse Area, Ellipse Sector, Ellipse Segment, Ellipse Overlap, Algorithm, Quartic Formula.

from the sun into the calibrator changes (~7° in 2 minutes). The attenuator grating ellipses thus move across the entrance aperture, and some of the smaller ellipses pass in and out of the entrance aperture ellipse during calibration. Movement of the small ellipses across the aperture creates fluctuations in the total amount of incident sunlight reaching the calibrator in the range of 0.3 to 0.5%. This jitter creates errors in the calibration algorithms. In order to model the jitter, an algorithm is required for determining the overlap area of two ellipses. Monte Carlo integration had been used; however, the method is numerically intensive because it converges very slowly, so it was not an attractive approach for modeling the calibrator due to the large number of ellipses that must be modeled.

In a more down-to-earth setting, populated places such as city streets or building corridors can become quite congested while crowds of people are moving about. Understanding the dynamics of pedestrian movement in these scenarios can be beneficial in many ways. Pedestrian dynamics can provide critical input to the design of buildings or city infrastructure, for example by predicting the effects of specific crowd management strategies, or the behavior of crowds utilizing emergency escape routes. Current research in pedestrian dynamics is making steady progress toward realistic modeling of local movement; see, e.g., [2]. The model presented in [2] is based on the concept of elliptical volume exclusion for individual pedestrians. Each model pedestrian is surrounded by an elliptical footprint area that the model uses to anticipate obstacles and other pedestrians in or near the intended path. The footprint area is influenced by an individuals' velocity; for example, the exclusion area in front of a fast-moving pedestrian is elongated when compared to a slower-moving individual, since a pedestrian is generally thinking a few steps ahead. As pedestrians travel through a confined space, their collective exclusion areas become denser, and the areas will eventually begin to overlap. A force-based model will produce a repulsive force between overlapping exclusion areas, causing the pedestrians to slow down or change course when the exclusion force becomes large. Implementing the force-based model with elliptical exclusion areas in a simulation requires calculating the overlap area between many different ellipses in the most general orientations. The ellipse area overlap algorithm must also be efficient, so as not to bog down the simulation.

Simulations for both the satellite solar calibrator and force-based pedestrian dynamic model require efficient calculation of the overlap area between two ellipses. In this paper, we provide an algorithm that has served well for both applications. The core component of the overlap area algorithm is based on determining the area of an *ellipse segment*, which is the area between a secant line and the ellipse boundary. The segment algorithm forms the basis of an application for calculating the overlap area between two general ellipses.

2. Ellipse area, sector area and segment area.

2.1. **Ellipse Area.** Consider an ellipse that is centered at the origin, with its axes aligned to the coordinate axes. If the semi-axis length along the x-axis is A, and the semi-axis length along the y-axis is B, then the ellipse is defined by a locus of points that satisfy the implicit polynomial equation

$$\frac{x^2}{A^2} + \frac{y^2}{B^2} = 1 \tag{1}$$

The same ellipse can be defined parametrically by:

$$\left.\begin{array}{l} x = A \cdot \cos(t) \\ y = B \cdot \sin(t) \end{array}\right\} \quad 0 \leq t \leq 2\pi \tag{2}$$

The area of such an ellipse can be found using the parameterized form with the Gauss-Green formula:

$$\begin{aligned} \text{Area} &= \frac{1}{2} \int_A^B [x(t) \cdot y'(t) - y(t) \cdot x'(t)]dt \\ &= \frac{1}{2} \int_0^{2\pi} A \cdot \cos(t) \cdot B \cdot \cos(t) - B \cdot \sin(t) \cdot (-A) \cdot \sin(t)]dt \\ &= \frac{A \cdot B}{2} \int_0^{2\pi} \cos^2(t) + \sin^2(t)]dt = \frac{A \cdot B}{2} \int_0^{2\pi} dt \\ &= \pi \cdot A \cdot B \end{aligned} \tag{3}$$

2.2. **Ellipse Sector Areas.** We define the *ellipse sector* between two points (x_1, y_1) and (x_2, y_2) on the ellipse as the area that is swept out by a vector from the origin to the ellipse, beginning at (x_1, y_1), as the vector travels along the ellipse in a counter-clockwise direction from (x_1, y_1) to (x_2, y_2). An example is shown in Fig. 1. The Gauss-Green formula can also be used to determine the area of such an ellipse sector.

$$\begin{aligned} \text{Sector Area} &= \frac{A \cdot B}{2} \int_{\theta_1}^{\theta_2} dt \\ &= \frac{(\theta_2 - \theta_1) \cdot A \cdot B}{2} \end{aligned} \tag{4}$$
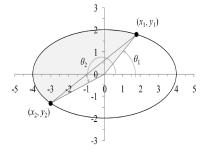

FIGURE 1. The area of an *ellipse sector* between two points on the ellipse is the area swept out by a vector from the origin to the first point as the vector travels along the ellipse in a counter-clockwise direction to the second point. The area of an ellipse sector can be determined with the Gauss-Green formula, using the parametric angles θ_1 and θ_2.

The parametric angle θ that is formed between the x-axis and a point (x, y) on the ellipse is found from the ellipse parameterizations:

$$x = A \cdot \cos(\theta) \implies \theta = \cos^{-1}(x/A)$$
$$y = B \cdot \sin(\theta) \implies \theta = \sin^{-1}(y/B)$$

For a circle ($A = B$ in the ellipse implicit polynomial form), the parametric angle corresponds to the geometric (visual) angle that a line from the origin to the point (x, y) makes with the x-axis. However, the same cannot be said for an ellipse; that is, the geometric (visual) angle is *not* the same as the parametric angle used in the area calculation. For example, consider the ellipse in Fig. 1; the implicit polynomial form is

$$\frac{x^2}{4^2} + \frac{y^2}{2^2} = 1 \tag{5}$$

Suppose the point (x_1, y_1) is at $\left(4/\sqrt{5}, 4/\sqrt{5}\right)$. The point is on the ellipse, since

$$\frac{\left(4/\sqrt{5}\right)^2}{4^2} + \frac{\left(4/\sqrt{5}\right)^2}{2^2} = \frac{4^2/5}{4^2} + \frac{4^2/5}{2^2} = \frac{1}{5} + \frac{4}{5} = 1$$

A line segment from the origin to $\left(4/\sqrt{5}, 4/\sqrt{5}\right)$ forms an angle with the x-axis of $\pi/4$ (≈ 0.7485398). However, the ellipse parametric angle to the same point is:

$$\theta = \cos^{-1}\left(\frac{4/\sqrt{5}}{4}\right) = \cos^{-1}\left(\frac{1}{\sqrt{5}}\right) \approx 1.10715$$

The same angle can also be found from the parametric equation for y:

$$\theta = \sin^{-1}\left(\frac{4/\sqrt{5}}{2}\right) = \sin^{-1}\left(\frac{2}{\sqrt{5}}\right) \approx 1.10715$$

The angle found by using the parametric equations does not match the geometric angle to the point that defines the angle.

When determining the parametric angle for a given point (x, y) on the ellipse, the angle must be chosen in the proper quadrant, based on the signs of x and y. For the ellipse in Fig. 1, suppose the point (x_2, y_2) is at $\left(-3, -\sqrt{7}/2\right)$. The parametric angle that is determined from the equation for x is:

$$\theta = \cos^{-1}\left(\frac{-3}{4}\right) \approx 2.41886$$

The parametric angle that is determined from the equation for y is:

$$\theta = \sin^{-1}\left(\frac{-\sqrt{7}/2}{2}\right) = \sin^{-1}\left(\frac{-\sqrt{7}}{4}\right) \approx -.722734$$

The apparent discrepancy is resolved by recalling that inverse trigonometric functions are usually implemented to return a 'principal value' that is within a conventional range. The typical (principal-valued) $\theta = \arccos(x)$ function returns angles in the range $0 = \theta = \pi$, and the typical (principal-valued) $\theta = \arcsin(x)$ function returns angles in the range $-\pi/2 = \theta = \pi/2$. When the principal-valued inverse trigonometric functions return angles in the typical ranges, the ellipse parametric angles, defined to be from the x-axis, with positive angles in the counter-clockwise direction, can be found with the relations in Table 2.2.

Quadrant II ($x < 0$ and $y \geq 0$) $\theta = \arccos(x/A)$ $= \pi - \arcsin(y/B)$	**Quadrant I** ($x \geq 0$ and $y \geq 0$) $\theta = \arccos(x/A)$ $= \arcsin(y/B)$
Quadrant III ($x < 0$ and $y < 0$) $\theta = 2\pi - \arccos(x/A)$ $= \pi - \arcsin(y/B)$	**Quadrant IV** ($x \geq 0$ $y < 0$) $\theta = 2\pi - \arccos(x/A)$ $= 2\pi + \arcsin(y/B)$

TABLE 1. Relations for finding the parametric angle that corresponds to a given point (x, y) on the ellipse $x^2/A^2 + y^2/B^2 = 1$. The parametric angle is formed between the positive x-axis and a line drawn from the origin to the given point, with counterclockwise being positive. For the standard (principal-valued) inverse trigonometric functions, the resulting angle will be in the range $0 \leq \theta < 2\pi$ for any point on the ellipse.

The point at $\left(-3, -\sqrt{7}/2\right)$ on the ellipse of Fig. 1 is in Quadrant III. Using the relations in Table 2.2, the parametric angle that is determined from the equation for x is:

$$\theta = 2\pi - \arccos(\frac{-3}{4}) \approx 3.86433$$

The parametric angle that is determined from the equation for y is:

$$\theta = \pi - \arcsin(\frac{-\sqrt{7}/2}{2}) \approx 3.86433$$

With the proper angles, the Gauss-Green formula can be used to determine the area of the sector from the point at $\left(4/\sqrt{5}, 4/\sqrt{5}\right)$ to the point $\left(-3, -\sqrt{7}/2\right)$ in the ellipse of Fig. 1.

$$\begin{aligned} \text{Sector Area} &= \frac{(\theta_2 - \theta_1) \cdot A \cdot B}{2} \\ &= \frac{\left[\left(2\pi - \arccos\left(\frac{-3}{4}\right)\right) - \arccos\left(\frac{4/\sqrt{5}}{4}\right)\right] \cdot 4 \cdot 2}{2} \\ &\approx 11.0287 \end{aligned} \tag{6}$$

The Gauss-Green formula is sensitive to the direction of integration. For the larger goal of determining ellipse overlap areas, we define the ellipse sector area to be calculated from the first point (x_1, y_1) to the second point (x_2, y_2) in a *counterclockwise* direction along the ellipse. For example, if the points (x_1, x_1) and (x_2, y_2) of Fig. 1 were to have their labels switched, then the ellipse sector defined by the new points will have an area that is complementary to that of the sector in Fig. 1, as shown in Fig. 2.

Switching the point labels, as shown in Fig. 2, also causes the angle labels to be switched, resulting in the condition that $\theta_1 > \theta_2$. Since using the definitions in Table 2.2 will always produce an angle in the range $0 = \theta < 2\pi$ for any point on the ellipse, the first angle can be transformed by subtracting 2π to restore the condition that $\theta_1 < \theta_2$. The sector area formula given above can then be used, with the integration angle from $(\theta_1 - 2\pi)$ through θ_2. With the angle labels shown
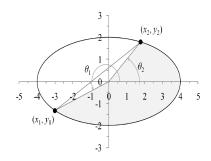
FIGURE 2. We define the ellipse sector area to be calculated from the first point (x_1, y_1) to the second point (x_2, y_2) in a counter-clockwise direction along the ellipse.

in Fig. 2, the area of the sector from the point at $\left(-3, -\sqrt{7}/2\right)$ to the point at $\left(4/\sqrt{5}, 4/\sqrt{5}\right)$ in a counter-clockwise direction is:

$$
\begin{aligned}
\text{Sector Area} &= \frac{(\theta_2 - (\theta_1 - 2\pi)) \cdot A \cdot B}{2} \\
&= \frac{\left[\left(2\pi - \arccos\left(\frac{-3}{4}\right)\right) - \left(\arccos\left(\frac{4/\sqrt{5}}{4}\right) - 2\pi\ \right)\right] \cdot 4 \cdot 2}{2} \quad (7) \\
&\approx 14.1040
\end{aligned}
$$

The two sector areas shown in Fig. 1 and Fig. 2 are complementary, in that they add to the total ellipse area. Using the angle labels as shown in Fig. 1 for both sector areas:

$$
\begin{aligned}
\text{Total Area} &= \frac{(\theta_2 - \theta_1) \cdot A \cdot B}{2} + \frac{(\theta_1 - (\theta_2 - 2\pi)) \cdot A \cdot B}{2} \\
&= \frac{(2\pi) \cdot A \cdot B}{2} = \pi \cdot A \cdot B \quad (8) \\
&= \pi \cdot 4 \cdot 2 \\
&\approx 25.1327
\end{aligned}
$$

2.3. **Ellipse Segment Areas.** For the overall goal of determining overlap areas between ellipses and other curves, a useful measure is the area of what we will call an *ellipse segment*. A secant line drawn between two points on an ellipse partitions the ellipse area into two fractions, as shown in Fig. 1 and Fig. 2. We define the ellipse segment as the area confined by the secant line and the portion of the ellipse from the first point (x_1, y_1) to the second point (x_2, y_2) *traversed in a counter-clockwise direction*. The segment's complement is the second of the two areas that are demarcated by the secant line. For the ellipse of Fig. 1, the area of the segment defined by the secant line through the points (x_1, y_1) and (x_2, y_2) is the area of the sector *minus* the area of the triangle defined by the two points and the ellipse center. To find the area of the triangle, suppose that the coordinates for the vertices of are known, e.g., as (x_1, y_1), (x_2, y_2) and (x_3, y_3). Then the triangle area can be

found by:

$$\text{Triangle Area} = \frac{1}{2} \cdot \left| det \begin{pmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ 1 & 1 & 1 \end{pmatrix} \right| \tag{9}$$

$$= \frac{1}{2} \cdot |x_1 \cdot (y_2 - y_3) - x_2 \cdot (y_1 - y_3) + x_3 \cdot (y_1 - y_2)|$$

In the case where one vertex, say (x_3, y_3), is at the origin, then the area formula for the triangle can be simplified to:

$$\text{Triangle Area} = \frac{1}{2} \cdot |x_1 \cdot y_2 - x_2 \cdot y_1| \tag{10}$$

For the case depicted in Fig. 1, subtracting the triangle area from the area of the ellipse sector area gives the area between the secant line and the ellipse, i.e., the area of the ellipse segment counter-clockwise from (x_1, y_1) to (x_2, y_2):

$$\text{Segment Area} = \frac{(\theta_2 - \theta_1) \cdot A \cdot B}{2} - \frac{1}{2} \cdot |x_1 \cdot y_2 - x_2 \cdot y_1| \tag{11}$$

For the ellipse of Fig. 1, with the points at $(4/\sqrt{5}, 4/\sqrt{5})$ and $(-3, -\sqrt{7}/2)$, the area of the segment defined by the secant line is:

$$\frac{\left[\left(2\pi - \arccos\left(\frac{-3}{4}\right) \right) - \arccos\left(\frac{4/\sqrt{5}}{4}\right) \right] \cdot 4 \cdot 2}{2} - \frac{1}{2} \cdot \left| \frac{4}{\sqrt{5}} \cdot \frac{-\sqrt{7}}{2} - \frac{4}{\sqrt{5}} \cdot -3 \right|$$

$$\approx 9.52865$$

For the ellipse of Fig. 2, the area of the segment shown is the sector area *plus* the area of the triangle.

$$\text{Segment Area} = \frac{(\theta_2 - (\theta_1 - 2\pi)) \cdot A \cdot B}{2} + \frac{1}{2} \cdot |x_1 \cdot y_2 - x_2 \cdot y_1| \tag{12}$$

With the points at $(-3, -\sqrt{7}/2)$ and $(4/\sqrt{5}, 4/\sqrt{5})$ the area of the segment is:

$$\frac{\left[\left(2\pi - \arccos\left(\frac{-3}{4}\right) \right) - \left(\arccos\left(\frac{4/\sqrt{5}}{4}\right) - 2\pi \right) \right] \cdot 4 \cdot 2}{2} + \frac{1}{2} \cdot \left| \frac{4}{\sqrt{5}} \cdot \frac{-\sqrt{7}}{2} - \frac{4}{\sqrt{5}} \cdot -3 \right|$$

$$\approx 15.60409411$$

For the case shown in Fig. 1 and Fig. 2, the sector areas were shown to be complementary. The segment areas are also complementary, since the triangle area is added to the sector of Fig. 1, but subtracted from the sector of Fig. 2. Using the angle labels as shown in Fig. 1 for both sector areas:

$$\text{Total Area} = \left[\frac{(\theta_2 - \theta_1) \cdot A \cdot B}{2} - \frac{1}{2} \cdot |x_1 \cdot y_2 - x_2 \cdot y_1| \right]$$

$$+ \left[\frac{(\theta_1 - (\theta_2 - 2\pi)) \cdot A \cdot B}{2} + \frac{1}{2} \cdot |x_1 \cdot y_2 - x_2 \cdot y_1| \right] \tag{13}$$

$$= \pi \cdot A \cdot B = \pi \cdot 4 \cdot 2 \approx 25.1327$$

The key difference between the cases in Fig. 1 and Fig. 2 that requires the area of the triangle to be either subtracted from, or added to, the sector area is the size of the *integration angle*. If the integration angle is less than π, then the triangle area

ELLIPSE_SEGMENT Area Algorithm:

1.
$$\theta_1 = \begin{cases} \arccos\left(x_1/A\right) & , \quad y_1 \geq 0 \\ 2\pi - \arccos\left(x_1/A\right) & , \quad y_1 < 0 \end{cases}$$
$$\theta_2 = \begin{cases} \arccos\left(x_2/A\right) & , \quad y_2 \geq 0 \\ 2\pi - \arccos\left(x_2/A\right) & , \quad y_2 < 0 \end{cases}$$

2.
$$\widehat{\theta}_1 = \begin{cases} \theta_1 & , \quad \theta_1 < \theta_2 \\ \theta_1 - 2\pi & , \quad \theta_1 > \theta_2 \end{cases}$$

3.
$$\text{Area} = \frac{\left(\theta_2 - \widehat{\theta}_1\right) \cdot A \cdot B}{2} + \frac{sign\left(\theta_2 - \widehat{\theta}_1 - \pi\right)}{2} \cdot \left|x_1 \cdot y_2 - x_2 \cdot y_1\right|$$

where:

the ellipse implicit polynomial equation is

$\frac{x^2}{A^2} + \frac{y^2}{B^2} = 1$

$A > 0$ is the semi-axis length along the x-axis

$B > 0$ is the semi-axis length along the y-axis

(x_1, y_1) is the first given point on the ellipse

(x_2, y_2) is the second given point on the ellipse

θ_1 and θ_2 are the parametric angles corresponding to the points

(x_1, y_1) and (x_2, y_2)

TABLE 2. An outline of the ELLIPSE_SEGMENT area algorithm.

must be subtracted from the sector area to give the segment area. If the integration angle is greater than π, the triangle area must be added to the sector area.

2.4. **A Core Algorithm for Ellipse Segment Area.** A generalization of the cases given in Fig. 1 and Fig. 2 suggests a robust approach for determining the ellipse segment area defined by a secant line drawn between two given points on the ellipse. The ellipse is assumed to be centered at the origin, with its axes parallel to the coordinate axes. We define the segment area to be demarcated by the secant line and the ellipse proceeding counter-clockwise from the first given point (x_1, y_1) to the second given point (x_2, y_2). The ELLIPSE_SEGMENT algorithm is outlined in Table 2, with pseudo-code presented in List. 1. The ellipse is passed to the algorithm by specifying the semi-axes lengths, $A > 0$ and $B > 0$. The points are passed to the algorithm as (x_1, x_1) and (x_2, y_2), which must be on the ellipse.

For robustness, the algorithm should avoid divide-by-zero and inverse-trigonometric errors, so data checks should be included. The ellipse parameters A and B must be greater than zero. A check is provided to determine whether the points are on the ellipse, to within some numerical tolerance, ε. Since the points can only be checked as being on the ellipse to within some numerical tolerance, it may still be possible for the x-values to be slightly larger than A, leading to an error when calling the inverse trigonometric functions with the argument x/A. In this case, the algorithm checks whether the x-value close to A or $-A$, that is within a distance that is less than the numerical tolerance. If the closeness condition is met, then the algorithm assumes that the calling function passed a value that is indeed on the ellipse near the point $(A, 0)$ or $(-A, 0)$, so the value of x is nudged back to A or $-A$ to avoid any error when calling the inverse trigonometric functions. The core algorithm, including all data checks, is shown in List. 1.

LISTING 1. The ELLIPSE_SEGMENT algorithm is shown for calculating the area of a segment defined by the secant line drawn between two given points $(x_1,\ y_1)$ and $(x_2,\ y_2)$ on the ellipse $x_2/A_2 + y_2/B_2 = 1$. We define the segment area for this algorithm to be demarcated by the secant line and the ellipse proceeding counter-clockwise from the first given point $(x_1,\ y_1)$ to the second given point $(x_2,\ y_2)$.

```
1    ELLIPSE_SEGMENT (A, B, X1, Y1, X2, Y2)
2    do if (A   0 or B   0)
3       then return (−1, ERROR_ELLIPSE_PARAMETERS)              :DATA CHECK
4                 2    2      2   2                 2   2     2   2
5    do if (|X1 /A + Y1 /B     1| >   or  |X1 /A + Y1 /B     1| > )
6       then return (−1, ERROR_POINTS_NOT_ON_ELLIPSE)           :DATA CHECK
7    do if (|X1|/A > )
8        do if |X1| − A >
9           then return (−1, ERROR_INVERSE_TRIG)                :DATA CHECK
10          else do if X1 < 0
11                  then X1   −A
12                  else X1   A
13   do if (|X2|/A > )
14       do if |X2| − A >
15          then return (−1, ERROR_INVERSE_TRIG)                :DATA CHECK
16          else do if X2 < 0
17                    then X2   −A
18                    else X2   A
19   do if (Y1 < 0)                       :ANGLE QUADRANT FORMULA (TABLE 1)
20       then 1   2      acos (X1/A)
21       else 1   acos (X1/A)
22   do if (Y2 < 0)                       :ANGLE QUADRANT FORMULA (TABLE 1)
23       then 2   2      acos (X2/A)
24       else 2   acos (X2/A)
25   do if (1 > 2)                               :MUST START WITH 1 < 2
26       then 1   1 − 2
27   do if ((2      1) > )                    :STORE SIGN OF TRIANGLE AREA
28       then trsgn   +1.0
29       else trsgn   +1.0
30   area   0.5*(A*B*(2 − 1)      trsgn *|X1*Y2 − X2*Y1|)
31   return (area , NORMAL_TERMINATION)
```

An implementation of the ELLIPSE_SEGMENT algorithm written in c–code is shown in Appendix 4. The code compiles under Cygwin-1.7.7-1, and returns the following values for the two test cases presented in Fig. 1 and Fig. 2:

LISTING 2. Return values for the test cases in Fig. 1 and Fig. 2

```
32          cc call_es.c ellipse_segment.c −o call_es.exe
33          ./call_es
34          Calling ellipse_segment.c
35          Fig. 1: segment area =              9.52864712, return_value = 0
36          Fig. 2: segment area =             15.60409411, return_value = 0
37          sum of ellipse segments = 25.13274123
38          ellipse area by pi*A*B =          25.13274123
```

3. Extending the Core Segment Algorithm to more General Cases.

3.1. Segment Area for a (Directional) Line through a General Ellipse.
The core segment algorithm is based on an ellipse that is centered at the origin with its axes aligned to the coordinate axes. The algorithm can be extended to more general ellipses, such as rotated and/or translated ellipse forms. Start by considering the case for a standard ellipse with semi-major axis lengths of A and B that is centered at the origin and with its axes aligned with the coordinate axes. Suppose that the ellipse is rotated through a counter-clockwise angle φ, and that the ellipse is then translated so that its center is at the point $(h,\ k)$. The

rotated+translated ellipse could then be defined by the set of parameters (A, B, h, k, φ), with the understanding that the rotation through φ is performed before the translation through (h, k). The approach for extending the core segment area algorithm will be to determine analogs on the standard ellipse corresponding to any points of intersection between a shape of interest and the general rotated and translated ellipse. To identify corresponding points, features of the shape of interest are translated by $(-h, -k)$, and then rotated by $-\varphi$. The translated+rotated features are used to determine any points of intersection with a similar ellipse that is centered at the origin with its axes aligned to the coordinate axes. Then, the core segment algorithm can be called with the translated+rotated intersection points.

Rotation and translation are affine transformations that are also length- and area-preserving. In particular, the semi-axis lengths in the general rotated ellipse are preserved by both transformations, and corresponding points on the two ellipses will demarcate equal partition areas. Fig. 3 illustrates this idea, showing the ellipse of Fig. 1 which has been rotated counter-clockwise through an angle $\varphi = 3\pi/8$, then translated by $(h, k) = (-6, +3)$.
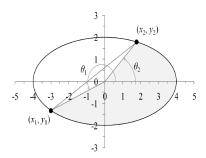

FIGURE 3. Translation and rotation are affine transformations that are also length-and area-preserving. Corresponding points on the two ellipses will demarcate equal partition areas.

Suppose that we desire to find the area of the rotated+translated ellipse sector defined by the line $y = -x$, where the line 'direction' travels from lower-right to upper-left, as shown in Fig. 3. We describe an approach for finding a segment in a rotated+translated ellipse, based on the core ellipse segment algorithm.

An ellipse that is centered at the origin, with its axes aligned to the coordinate axes, is defined parametrically by

$$\left.\begin{array}{l} x = A \cdot \cos(t) \\ y = B \cdot \sin(t) \end{array}\right\} \quad 0 \le t \le 2\pi$$

Suppose the ellipse is rotated through an angle φ, with counter-clockwise being positive, and that the ellipse is then to be translated to put its center is at the point (h, k). Any point (x, y) on the standard ellipse can be rotated and translated to end up in a corresponding location on the new ellipse by using the transformation:

$$\left[\begin{array}{c} x_{TR} \\ y_{TR} \end{array}\right] = \left[\begin{array}{cc} \cos(\varphi) & -\sin(\varphi) \\ \sin(\varphi) & \cos(\varphi) \end{array}\right] \cdot \left[\begin{array}{c} x \\ y \end{array}\right] + \left[\begin{array}{c} h \\ k \end{array}\right] \tag{14}$$

Rotation and translation of the original standard ellipse does not change the ellipse area, or the semi-axis lengths. One important feature of the algorithms presented here is that the semi-axis lengths A and B are in the direction of the x- and y-axes,

respectively, in the *un-rotated* (standard) ellipse. In its rotated orientation, the semi-axis length A will rarely be oriented horizontally (in fact, for $\varphi = \pi/4$, the semi-axis length A will be oriented vertically). Regardless of the orientation of the rotated+translated ellipse, the algorithms presented here assume that the values of A and B passed into the algorithm represent the semi-axis lengths along the x- and y-axes, respectively, for the corresponding un-rotated, un-translated ellipse. The angle φ is the amount of counter-clockwise rotation required to put the ellipse into its desired location. Specifying a negative value for φ will rotate the standard ellipse through a clockwise angle. The angle φ can be specified in anywhere in the range $(-8, +8)$; the working angle in the code will be computed from the given angle, modulo 2π, to avoid any potential errors (?) when calculating trigonometric values. The translation (h, k) is the absolute movement along the coordinate axes of the ellipse center to move a standard ellipse into its desired location. Negative values of h move the standard ellipse to the left; negative values of k move the standard ellipse down.

To find the area between the given line and the rotated+translated ellipse, the two curve equations can be solved simultaneously to find any points of intersection. But instead of searching for the points of intersection with the rotated+translated ellipse, it is more efficient to transform the two given points that define the line back through the translation $(-h, -k)$ then rotation through $-\varphi$. The new line determined by the translated+rotated points will pass through the standard ellipse at points that are analogous to where the original line intersects the rotated+translated ellipse.

The transformations required to move the given points (x_1, y_1) and (x_2, y_2) into an orientation with respect to a standard ellipse that is analogous to their orientation to the given ellipse are the inverse of what it took to rotate+translate the ellipse to its desired position. The translation is performed first, then the rotation:

$$\begin{bmatrix} x_{i_0} \\ y_{i_0} \end{bmatrix} = \begin{bmatrix} \cos(-\varphi) & -\sin(-\varphi) \\ \sin(-\varphi) & \cos(-\varphi) \end{bmatrix} \cdot \begin{bmatrix} x_i - h \\ y_i - k \end{bmatrix} \tag{15}$$

Multiplying the vector by the matrix, and simplifying the negative-angle trig functions gives the following expressions for the translated+rotated points:

$$x_{i_0} = \cos(\varphi) \cdot (x_i - h) + \sin(\varphi) \cdot (y_i - k)$$
$$y_{i_0} = -\sin(\varphi) \cdot (x_i - h) + \cos(\varphi) \cdot (y_i - k)$$

The two new points (x_{1_0}, y_{1_0}) and (x_{2_0}, y_{2_0}) can be used to determine a line, *e.g.*, by the point-slope method:

$$y = y_{1_0} + \frac{y_{2_0} - y_{1_0}}{x_{2_0} - x_{1_0}} (x - x_{1_0}) \tag{16}$$

The equation can also be formulated in an alternative way to accommodate cases where the translated+rotated line is vertical, or nearly so:

$$x = x_{1_0} + \frac{x_{2_0} - x_{1_0}}{y_{2_0} - y_{1_0}} (y - y_{1_0}) \tag{17}$$

Points of intersection are found by substituting the line equations into the standard ellipse equation, and solving for the remaining variable. For each case, define the slope as:

$$m_{yx} = \frac{y_{2_0} - y_{1_0}}{x_{2_0} - x_{1_0}}, \quad m_{xy} = \frac{x_{2_0} - x_{1_0}}{y_{2_0} - y_{1_0}} \tag{18}$$

Then the two substitutions proceed as follows:

$$y = y_{1_0} + m_{yx} \cdot (x - x_{1_0}) \quad \text{into} \quad \frac{x^2}{A^2} + \frac{y^2}{B^2} = 1$$

$$\implies \frac{x^2}{A^2} + \frac{(y_{1_0} + m_{yx} \cdot (x - x_{1_0}))^2}{B^2} = 1$$

$$\implies \left[\frac{B^2 + A^2 \cdot (m_{yx})^2}{A^2} \right] \cdot x^2 \tag{19}$$

$$+ \left[2 \cdot \left(y_{1_0} \cdot m_{yx} - (m_{yx})^2 \cdot x_{1_0} \right) \right] \cdot x$$

$$+ \left[(y_{1_0})^2 - 2 \cdot m_{yx} \cdot x_{1_0} \cdot y_{1_0} + (m_{yx} \cdot x_{1_0})^2 - B^2 \right]$$

$$= 0$$

$$x = x_{1_0} + m_{xy} \cdot \left(y - y_{1_0} \right) \quad \text{into} \quad \frac{x^2}{A^2} + \frac{y^2}{B^2} = 1$$

$$\implies \frac{(x_{1_0} + m_{xy} \cdot (y - y_{1_0}))^2}{A^2} + \frac{y^2}{B^2} = 1$$

$$\implies \left[\frac{A^2 + B^2 \cdot (m_{xy})^2}{B^2} \right] \cdot y^2 \tag{20}$$

$$+ \left[2 \cdot \left(x_{1_0} \cdot m_{xy} - (m_{xy})^2 \cdot y_{1_0} \right) \right] \cdot y$$

$$+ \left[(x_{1_0})^2 - 2 \cdot m_{xy} \cdot x_{1_0} \cdot y_{1_0} + (m_{xy} \cdot y_{1_0})^2 - A^2 \right]$$

$$= 0$$

If the translated+rotated line is not vertical, then use the first equation to find the x-values for any points of intersection. If the translated+rotated line is close to vertical, then the second equation can be used to find the y-values for any points of intersection. Since points of intersection between the line and the ellipse are determined by solving a quadratic equation $ax^2 + bx + c$, there are three cases to consider:

1. $\Delta = b^2 - 4ac < 0$: Complex Conjugate Roots (no points of intersection)
2. $\Delta = b^2 - 4ac = 0$: One Double Real Root (1 point of intersection; line tangent to ellipse)
3. $\Delta = b^2 - 4ac > 0$: Two Real Roots (2 points of intersection; line crosses ellipse)

For the first two cases, the segment area will be zero. For the third case, the two points of intersection can be sent to the core segment area algorithm. However, to enforce a consistency in area measures returned by the core algorithm, the integration direction is specified to be from the first point to the second point. As such, the ellipse line overlap algorithm should be sensitive to the order that the points are passed to the core segment algorithm. We suggest giving the line a 'direction' from the first given point on the line to the second. The line 'direction' can then be used to determine which is to be the first point of intersection, i.e., the first intersection point is where the line enters the ellipse based on what 'direction' the line is pointing. The segment area that will be returned from ELLIPSE_SEGMENT by passing the line's entry location as the first intersection point is the area within the ellipse to the right of the line's path.

The approach outlined above for finding the overlap area between a line and a general ellipse is implemented in the ELLIPSE_LINE_OVERLAP algorithm, with pseudo-code shown in List. 3. The ellipse is passed to the algorithm by specifying the counterclockwise rotation angle φ and the translation (h, k) that takes a standard ellipse and moves it to the desired orientation, along with the semi-axes lengths, $A > 0$ and > 0. The line is passed to the algorithm as two points on the line, (x_1, y_1) and (x_2, y_2). The 'direction' of the line is taken to be from (x_1, y_1) toward (x_2, y_2). Then, the segment area returned from ELLIPSE_SEGMENT will be the area within the ellipse to the right of the line's path.

> LISTING 3. The ELLIPSE_LINE_OVERLAP algorithm is shown for calculating the area of a segment in a general ellipse that is defined by a given line. The line is considered to have a 'direction' that runs from the first given point (x_1, y_1) to the second given point (x_2, y_2). The line 'direction' determines the order in which intersection points are passed to the EL-LIPSE_SEGMENT algorithm, which will return the area of the segment that runs along the ellipse from the first point to the second in a counter-clockwise direction. Any routine that calls the algorithm ELLIPSE_LINE_OVERLAP must be sensitive to the order of points that are passed in.

```
39  (Area,Code) ← ELLIPSE\_LINE\_OVERLAP (A,B,H,K,φ,X1,Y1,X2,Y2)
40    do if (A ≤ 0 or B ≤ 0)
41
42      then return (−1, ERROR_ELLIPSE_PARAMETERS)      :DATA_CHECK
43
44    do if ( |φ| > 2π)
45
46      then φ ← (φ modulo 2π) :BRING φ INTO −2π ≤ φ < 2π (?)
47
48    do if (|X1|/A > 2π)
49
50      then X1 ← −A
51
52  X10 ← cos(φ) ∗ (X1 − H) + sin(φ) ∗ (Y1 − K)
53
54  Y10 ← − sin(φ) ∗ (X1 − H) + cos(φ) ∗ (Y1 − K)
55
56  X20 ← cos(φ) ∗ (X2 − H) + sin(φ) ∗ (Y2 − K)
57
58  Y20 ← − sin(φ) ∗ (X2 − H) + cos(φ) ∗ (Y2 − K)
59
60    do if (|X20 − X10| > ε )                   :LINE IS NOT VERTICAL
61
62      then m ← (Y20 − Y10)/(X20 − X10) :STORE QUADRATIC COEFFICIENTS
63
64          a ← (B² + (A∗m)²)/A²
65
66          b ← (2.0∗(Y10∗m −− m²∗X10))
67
68          c ← (Y10² − 2.0∗m∗Y10∗X10 + (m∗X10)² −− B²)
69
70      else if (|Y20 −− Y10|  > ε)                   :LINE IS NOT HORIZONTAL
71
72          then m ← (X20 − X10)/(Y20 − Y10)   :STORE QUADRATIC COEFFS
73
74              a ← (A² + (B∗m)²)/B²
75
76              b ← (2.0∗(X10∗m −− m²∗Y10))
77
78              c ←(X10² − 2.0∗m∗Y10∗X10 + (m∗Y10)² −− A²)
79
```

```
80      else return (−1, ERROR_LINE_POINTS)        :LINE POINTS TOO CLOSE
81
82   discrim ← b² − 4.0*a*c
83
84   do if (discrim < 0.0)                  :LINE DOES NOT CROSS ELLIPSE
85
86      then return (0, NO_INTERSECT)
87
88      else if (discrim > 0.0)                :TWO INTERSECTION POINTS
89
90      then root1 ← (−b − sqrt (discrim))/(2.0*a)
91
92           root2 ← (−b + sqrt (discrim))/(2.0*a)
93
94      else return (0, TANGENT)                :LINE TANGENT TO ELLIPSE
95
96   do if (|X20 − X10| > ε)            :ROOTS ARE X–VALUES
97
98      then do if (X10 < X20)          :ORDER PTS SAME AS LINE DIRECTION
99
100              then x1 ← root1
101
102                   x2 ← root2
103
104              else x1 ← root2
105
106                   x2 ← root1
107
108     else do if (Y10 < Y20)                    :ROOTS ARE Y–VALUES
109
110              then y1 ← root1      :ORDER PTS SAME AS LINE DIRECTION
111
112                   y2 ← root2
113
114              else y1 ← root2
115
116                   y2 ← root1
117
118   (Area, Code) ← ELLIPSE_SEGMENT (A,B,x1,y1,x2,y2)
119
120   do if (Code < NORMAL_TERMINATION)
121
122      then return (−1.0, Code)
123
124      else return (Area, TWO_INTERSECTION_POINTS)
```

An implementation of the ELLIPSE_LINE_OVERLAP algorithm in c-code is shown in Appendix 5. The code compiles under Cygwin-1.7.7-1, and returns the following values for the test cases presented above in Fig. 3, with both line 'directions':

LISTING 4. Return values for the test cases in Fig. 3.

```
125   cc call_el.c ellipse_line_overlap.c ellipse_segment.c −o call_el.exe
126
127   ./call_el
128
129  Calling ellipse_line_overlap.c
130
131   area =                    4.07186819, return_value = 102
132
133   reverse: area =           21.06087304, return_value = 102
134
135   sum of ellipse segments =       25.13274123
136
137   total ellipse area by pi*A*B =   25.13274123
```

3.2. Ellipse-Ellipse Overlap Area.
The method described above for determining the area between a line and an ellipse can be extended to the task of finding the

overlap area between two general ellipses. Suppose the two ellipses are defined by their semi-axis lengths, center locations and axis rotation angles. Let the two sets of parameters $(A_1, B_1, h_1, k_1, \varphi_1)$ and $(A_2, B_2, h_2, k_2, \varphi_2)$ define the two ellipses for which overlap area is sought. The approach presented here will be to first translate both ellipses by an amount $(-h_1, -k_1)$ that puts the center of the first ellipse at the origin. Then, both translated ellipses are rotated about the origin by an angle $-\varphi_1$ that aligns the axes of the first ellipse with the coordinate axes; see Fig. 4. Intersection points are found for the two translated+rotated ellipses, using Ferrari's quartic formula. Finally, the segment algorithm described above is employed to find all the pieces of the overlap area.
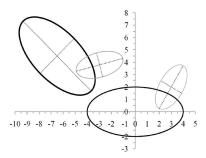

FIGURE 4. Intersection points on each curve are used with the ellipse segment area algorithm to determine overlap area, by calculating the area of appropriate segments, and polygons in certain cases. For the case of two intersection points, as shown above, the overlap area can be found by adding two segments, as shown in Fig. 5.

For example, consider a case of two general ellipses with two (non-tangential) points of intersection, as shown in Fig. 4. The translation+rotation transformations that put the first ellipse at the origin and aligned with the coordinate axes do not alter the overlap area. In the case shown in Fig. 4, the overlap area consists of one segment from the first ellipse and one segment from the second ellipse. The segment algorithm presented above can be used directly for ellipses centered at the origin and aligned with the coordinate axes. As such, the desired segment from the first ellipse can be found immediately with the segment algorithm, based on the points of intersection. To find the desired segment of the second ellipse, the approach presented here further translates and rotates the second ellipse so that the segment algorithm can also be used directly. The overlap area for the case shown in Fig. 4 is equal to the sum of the two segment areas, as shown in Fig. 5. Other cases, e.g. with 3 and 4 points of intersection, can also be handled using the segment algorithm.

The overlap area algorithm presented here finds the area of appropriate sector(s) of each ellipse, which are demarcated by any points of intersection between the two ellipse curves. To find intersection points, the two ellipse equations are solved simultaneously. This step can be accomplished by using the implicit polynomial forms for each ellipse. The first ellipse equation, in its translated+rotated position is written as an implicit polynomial using the appropriate semi-axis lengths:
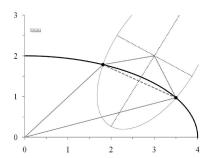
FIGURE 5. The area of overlap between two intersecting ellipses can be found by using the ellipse sector algorithm. In the case of two (non-tangential) intersection points, the overlap area is equal to the sum of two ellipse sectors. The sector in each ellipse is demarcated by the intersection points.

$$\frac{x^2}{A_1^2} + \frac{y^2}{B_1^2} = 1 \tag{21}$$

In a general form of this problem, the translation+rotation that puts the first ellipse centered at the origin and oriented with the coordinate axes will typically leave the second ellipse displaced and rotated. The implicit polynomial form for a more general ellipse that is rotated and/or translated away from the origin is written in the conventional way as:

$$AA \cdot x^2 + BB \cdot x \cdot y + CC \cdot y^2 + DD \cdot x + EE \cdot y + FF = 0 \tag{22}$$

Any points of intersection for the two ellipses will satisfy these two equations simultaneously. An intermediate goal is to find the implicit polynomial coefficients in Ellipse Eq. 22 that describe the second ellipse after the translation+rotation that puts the first ellipse centered at the origin and oriented with the coordinate axes. The parameters that describe the second ellipse after the translation+rotation can be determined from the original parameters for the two ellipses. The first step is to translate the second ellipse center (h_2, k_2) through an amount $(-h_1, -k_1)$, then rotate the center-point through $-\varphi_1$ to give a new center point (h_{2TR}, k_{2TR}):

$$h_{2TR} = \cos\left(-\varphi_1\right) \cdot (h_2 - h_1) - \sin\left(-\varphi_1\right) \cdot (k_2 - k_1)$$
$$k_{2TR} = \sin\left(-\varphi_1\right) \cdot (h_2 - h_1) + \cos\left(-\varphi_1\right) \cdot (k_2 - k_1)$$

The coordinates for a point (x_{TR}, y_{TR}) on the second ellipse in its new translated+rotated position can be found from the following parametric equations, based on an ellipse with semi-axis lengths A_2 and B_2 that is centered at the origin, then rotated and translated to the desired position:

$$\left. \begin{array}{l} x_{TR} = A_2 \cdot \cos\left(t\right) \cdot \cos\left(\varphi_2 - \varphi_1\right) - B_2 \cdot \sin\left(t\right) \cdot \sin\left(\varphi_2 - \varphi_1\right) + h_{2TR} \\ y_{TR} = A_2 \cdot \cos\left(t\right) \cdot \sin\left(\varphi_2 - \varphi_1\right) + B_2 \cdot \sin\left(t\right) \cdot \cos\left(\varphi_2 - \varphi_1\right) + k_{2TR} \end{array} \right\} \; 0 \le t \le 2\pi$$

To find the implicit polynomial coefficients from the parametric form, further transform the locus of points (x_{TR}, y_{TR}) so that they lie on the ellipse $(A_2, B_2, 0, 0, 0)$, which is accomplished by first translating (x_{TR}, y_{TR}) through $(-(h_1 - h_2), -(k_1 - k_2))$ and then rotating the point through the angle $-(\varphi_1 - \varphi_2)$:

$$x = \cos(\varphi_2 - \varphi_1) \cdot (x_{TR} - (h_1 - h_2)) - \sin(\varphi_2 - \varphi_1) \cdot (y_{TR} - (k_1 - k_2))$$
$$y = \sin(\varphi_2 - \varphi_1) \cdot (x_{TR} - (h_1 - h_2)) + \cos(\varphi_2 - \varphi_1) \cdot (y_{TR} - (k_1 - k_2))$$

The locus of points (x, y) should satisfy the standard ellipse equation with the appropriate semi-axis lengths:

$$\frac{x^2}{A_2^2} + \frac{y^2}{B_2^2} = 1 \tag{23}$$

Finally, the implicit polynomial coefficients for Ellipse Eq. 22 are found by substituting the expressions for the point (x, y) into the standard ellipse equation, yielding the following ellipse equation:

$$\frac{[\cos(\varphi_2 - \varphi_1) \cdot (x_{TR} - (h_1 - h_2)) - \sin(\varphi_2 - \varphi_1) \cdot (y_{TR} - (k_1 - k_2))]^2}{A_2^2}$$
$$+ \frac{[\sin(\varphi_2 - \varphi_1) \cdot (x_{TR} - (h_1 - h_2)) + \cos(\varphi_2 - \varphi_1) \cdot (y_{TR} - (k_1 - k_2))]^2}{B_2^2} \tag{24}$$
$$= 1$$

where (xx_{TR}, y_{TR}) are defined as above. Expanding the terms, and then rearranging the order to isolate like terms yields the following expressions for the implicit polynomial coefficients of a general ellipse with the set of parameters $(A_2, B_2, h_{2TR}, k_{2TR}, \varphi_2 - \varphi_1)$:

$$AA = \frac{\cos^2(\varphi_2 - \varphi_1)}{A_2^2} + \frac{\sin^2(\varphi_2 - \varphi_1)}{B_2^2}$$

$$BB = \frac{2 \cdot \sin(\varphi_2 - \varphi_1) \cdot \cos(\varphi_2 - \varphi_1)}{A_2^2} - \frac{2 \cdot \sin(\varphi_2 - \varphi_1) \cdot \cos(\varphi_2 - \varphi_1)}{B_2^2}$$

$$CC = \frac{\sin^2(\varphi_2 - \varphi_1)}{A_2^2} + \frac{\cos^2(\varphi_2 - \varphi_1)}{B_2^2}$$

$$DD = \frac{-2 \cdot \cos(\varphi_2 - \varphi_1) \cdot [h_{2TR} \cdot \cos(\varphi_2 - \varphi_1) + k_{2TR} \cdot \sin(\varphi_2 - \varphi_1)]}{A_2^2}$$
$$+ \frac{2 \cdot \sin(\varphi_2 - \varphi_1) \cdot [k_{2TR} \cdot \cos(\varphi_2 - \varphi_1) - h_{2TR} \cdot \sin(\varphi_2 - \varphi_1)]}{B_2^2} \tag{25}$$

$$EE = \frac{-2 \cdot \sin(\varphi_2 - \varphi_1) \cdot [h_{2TR} \cdot \cos(\varphi_2 - \varphi_1) + k_{2TR} \cdot \sin(\varphi_2 - \varphi_1)]}{A_2^2}$$
$$+ \frac{2 \cdot \cos(\varphi_2 - \varphi_1) \cdot [h_{2TR} \cdot \sin(\varphi_2 - \varphi_1) - k_{2TR} \cdot \cos(\varphi_2 - \varphi_1)]}{B_2^2}$$

$$FF = \frac{[h_{2TR} \cdot \cos(\varphi_2 - \varphi_1) + k_{2TR} \cdot \sin(\varphi_2 - \varphi_1)]^2}{A_2^2}$$
$$+ \frac{[h_{2TR} \cdot \sin(\varphi_2 - \varphi_1) - k_{2TR} \cdot \cos(\varphi_2 - \varphi_1)]^2}{B_2^2} - 1$$

For the area overlap algorithm presented in this paper, the points of intersection between the two general ellipses are found by solving simultaneously the two implicit

polynomials denoted above as Ellipse Eq. 21 and Ellipse Eq. 22. Solving for x in the first equation:

$$\frac{x^2}{A_1^2} + \frac{y^2}{B_1^2} = 1 \quad \Longrightarrow \quad x = \pm\sqrt{A_1^2 \cdot \left(1 - \frac{y^2}{B_1^2}\right)} \tag{26}$$

Substituting these expressions for x into Ellipse Eq. 22 and then collecting terms yields a quartic polynomial in y. It turns out that substituting either the positive or the negative root gives the same quartic polynomial coefficients, which are:

$$cy\,[4] \cdot y^4 + cy\,[3] \cdot y^3 + cy\,[2] \cdot y^2 + cy\,[1] \cdot y + cy\,[0] = 0 \tag{27}$$

where:

$$\frac{cy\,[4]}{B_1} = A_1^4 \cdot AA^2 + B_1^2 \cdot \left[A_1^2 \cdot \left(BB^2 - 2 \cdot AA \cdot CC\right) + B_1^2 \cdot CC^2\right]$$

$$\frac{cy\,[3]}{B_1} = 2 \cdot B_1 \cdot \left[B_1^2 \cdot CC \cdot EE + A_1^2 \cdot (BB \cdot DD - AA \cdot EE)\right]$$

$$\frac{cy\,[2]}{B_1} = A_1^2 \cdot \left\{\left[B_1^2 \cdot \left(2 \cdot AA \cdot CC - BB^2\right) + DD^2 - 2 \cdot AA \cdot FF\right] - 2 \cdot A_1^2 \cdot AA^2\right\}$$
$$\qquad + B_1^2 \cdot \left(2 \cdot CC \cdot FF + EE^2\right)$$

$$\frac{cy\,[1]}{B_1} = 2 \cdot B_1 \cdot \left[A_1^2 \cdot (AA \cdot EE - BB \cdot DD) + EE \cdot FF\right]$$

$$\frac{cy\,[0]}{B_1} = \left[A_1 \cdot (A_1 \cdot AA - DD) + FF\right] \cdot \left[A_1 \cdot (A_1 \cdot AA + DD) + FF\right]$$

$$\tag{28}$$

In theory, the quartic polynomial will have real roots if and only if the two curves intersect. If the ellipses do not intersect, then the quartic will have only complex roots. Furthermore, any real roots of the quartic polynomial will represent y-values of intersection points between the two ellipse curves. As with the quadratic equation that arises in the ellipse-line overlap calculation, the ellipse-ellipse overlap algorithm should handle all possible cases for the types of quartic polynomial roots:

1. Four real roots (distinct or not); the ellipse curves intersect.
2. Two real roots (distinct or not) and one complex-conjugate pair; the ellipse curves intersect.
3. No real roots (two complex-conjugate pairs); the ellipse curves do not intersect.

For the method we present here, polynomial roots are found using Ferrari's quartic formula. A numerical implementation of Ferrari's formula is given in [3]. For complex roots are returned, and any roots whose imaginary part is returned as zero is a real root.

When the polynomial coefficients are constructed as shown above, the general case of two distinct ellipses typically results in a quartic polynomial, i.e., the coefficient $cy[4]$ is non-zero. However, certain cases lead to polynomials of lesser degree. Fortunately, the solver in [3] is conveniently modular, providing separate functions BIQUADROOTS, CUBICROOTS and QUADROOTS to handle all the possible polynomial cases that arise when seeking points of intersection for two ellipses.

If the polynomial solver returns no real roots to the polynomial, then the ellipse *curves* do not intersect. It follows that the two ellipse *areas* are either disjoint, or one ellipse area is fully contained inside the other; all three possibilities are shown in Fig. 6. Each sub-case in Fig. 6 requires a different overlap-area calculation, i.e. either the overlap area is zero (Case 0-3), or the overlap is the area of the first ellipse (Case 0-2), or the overlap is the area of the second ellipse (Case 0-1). When the polynomial has no real roots, geometry can be used to determine which specific sub-case of Fig. 6 is represented. An efficient logic starts by determining the relative size of the two ellipses, e.g., by comparing the product of semi-axis lengths for each ellipse. The area of an ellipse is proportional to the product of its two semi-axis lengths, so the relative size of two ellipses can be determined by comparing the product of semi-axis lengths:

$$(\pi \cdot A_1 \cdot B_1) \, \alpha \, (\pi \cdot A_2 \cdot B_2) \implies (A_1 \cdot B_1) \, \alpha \, (A_2 \cdot B_2), \; \alpha \in \{'<','>'\} \quad (29)$$

Suppose the first ellipse is larger than the second ellipse, then $A_1 B_1 > A_2 B_2$. In this case, if the second ellipse center $(h_{2TR}, \, k_{2TR})$ is inside the first ellipse, then the second ellipse is wholly contained within the first ellipse (Case 0-1); otherwise, the ellipses are disjoint (Case 0-3). The logic relies on the fact that there are no intersection points, which is indicated whenever there are no real solutions to the quartic polynomial. To test whether the second ellipse center $(h_{2TR}, \, k_{2TR})$ is inside the first ellipse, evaluate the first ellipse equation at the point $x = h_{2TR}$, and $y = k_{2TR}$; if the result is less than one, then the point $(h_{2TR}, \, k_{2TR})$ is inside the first ellipse. The complete logic for determining overlap area when $A_1 B_1 > A_2 B_2$ is:

If the polynomial has no real roots, and $A_1 B_1 > A_2 B_2$, and $\frac{h_{2TR}^2}{A_1^2} + \frac{k_{2TR}^2}{B_1^2} < 1$, then the first ellipse wholly contains the second, otherwise the two ellipses are disjoint.
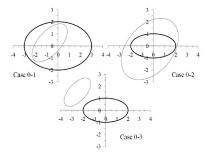

FIGURE 6. When the quartic polynomial has no real roots, the ellipse curves do not intersect. It follows that either one ellipse is fully contained within the other, or the ellipse areas are completely disjoint, resulting in three distinct cases for overlap area.

Alternatively, suppose that the second ellipse is larger than the first ellipse, then $A_1 B_1 < A_2 B_2$. If the first ellipse center $(0, \, 0)$ is inside the second ellipse, then the first ellipse is wholly contained within the second ellipse (Case 0-2); otherwise the ellipses are disjoint (Case 0-3). Again, the logic relies on the fact that there are no intersection points, To test whether $(0, \, 0)$ is inside the second ellipse, evaluate the second ellipse equation at the origin; if the result is less than zero, then the origin is inside the second ellipse. The complete logic for determining overlap area when $A_1 B_1 < A_2 B_2$ is:

If the polynomial has no real roots, and $A_1 B_1 < A_2 B_2$, and $FF < 0$, then the second ellipse wholly contains the first, otherwise the two ellipses are disjoint.

Suppose that the two ellipses are the same size, i.e., $A_1 B_1 = A_2 B_2$. In this case, when no intersection points exist, the ellipses must be disjoint (Case 0-3). It also turns out that the polynomial solver of [3] will return no real solutions if the ellipses are identical. This special case is also handled in the overlap area algorithm presented below. Pseudo-code for a function NOINTPTS that determines overlap area for the cases depicted in Fig. 6 is shown in Fig. 14.

If the polynomial solver returns either two or four real roots to the quartic equation, then the ellipse curves intersect. For the algorithm presented here, all of the various possibilities for the number and type of real roots are addressed by creating a list of distinct real roots. The first step is to loop through the entire array of complex roots returned by the polynomial solver, and retrieve only real roots, i.e., only those roots whose imaginary component is zero. The algorithm presented here then sorts the real roots, allowing for an efficient check for multiple roots. As the sorted list of real roots is traversed, any root that is 'identical' to the previous root can be skipped.

Each distinct real root of the polynomial represents a y-value where the two ellipses intersect. Each y-value can represent either one or two potential points of intersection. In the first case, suppose that the polynomial root is $y = B_1$ (or $y = -B_1$), then the y-value produces a single intersection point, which is at $(0, B_1)$ (or $(0, -B_1)$). In the second case, if the y-value is in the open interval $(-B_1, B_1)$, then there are two potential intersection points where the y-value is on the first ellipse:

$$\left(A_1 \cdot \sqrt{1 - \frac{y^2}{B_1^2}},\ y \right) \text{ and }$$

$$\left(-A_1 \cdot \sqrt{1 - \frac{y^2}{B_1^2}},\ y \right)$$

Each potential intersection point $(x_i,\ y_i)$ is evaluated in the second ellipse equation:

$$AA \cdot x_i^2 + BB \cdot x_i \cdot y_i + CC \cdot y_i^2 + DD \cdot x_i + EE \cdot y_i + FF, \quad i = 1, 2$$

If the expression evaluates to zero, then the point (x, y) is on both ellipses, i.e., it is an intersection point. By checking all points (x, y) for each value of y that is a root of the polynomial, a list of distinct intersection points is generated. The number of distinct intersection points must be either 0, 1, 2, 3 or 4. The case of zero intersection points is described above, with all possible sub-cases illustrated in Fig. 6. If there is only one distinct intersection point, then the two ellipses must be tangent at that point. The three possibilities for a single tangent point are shown in Fig. 7.

For the purpose of determining overlap area, the cases of 0 or 1 intersection points can be handled in the same way. When two intersection points exist, there are three possible sub-cases, shown in Fig. 8. It is possible that both of the intersection points are tangents (Case 2-1 and Case 2-2). In both of these sub-cases, one ellipse must be fully contained within the other. The only other possibility for two intersection points is a partial overlap (Case 2-3).
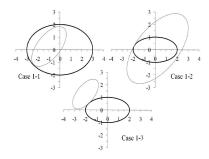
FIGURE 7. When only one intersection point exists, the ellipses must be tangent at the intersection point. As with the case of zero intersection points, either one ellipse is fully contained within the other, or the ellipse areas are disjoint. The algorithm for finding overlap area in the case of zero intersection points can also be used when there is a single intersection point.
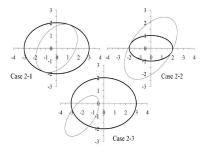

FIGURE 8. When two intersection points exist, either both of the points are tangents, or the ellipse curves cross at both points. For two tangent points, one ellipse must be fully contained within the other. For two crossing points, a partial overlap must exist

Each sub-case in Fig. 8 requires a different overlap-area calculation. When two intersection points exist, either both of the points are tangents, or the ellipse curves cross at both points. Specifically, when there are two intersection points, if one point is a tangent, then both points must be tangents. And, if one point is not a tangent, then neither point is a tangent. So, it suffices to check one of the intersection points for tangency. Suppose the ellipses are tangent at an intersection point; then, points that lie along the first ellipse on either side of the intersection will lie in the same region of the second ellipse (inside or outside). That is, if two points are chosen that lie on the first ellipse, one on each side of the intersection, then both points will either be inside the second ellipse, or they will both be outside the second ellipse. If the ellipse curves cross at the intersection point, then the two chosen points will be in different regions of the second ellipse.

A logic based on testing points that are adjacent to a tangent point can be implemented numerically to test whether an intersection point is a tangent or a cross-point. Starting with an intersection point (x, y), calculate the parametric angle on the first ellipse, by the rules in Table 2.2:

$$\theta = \begin{cases} \arccos(x/A_1) & y \geq 0 \\ 2\pi - \arccos(x/A_1) & y < 0 \end{cases} \tag{30}$$

A small perturbation angle is then calculated. For the method presented here, we seek to establish an angle that corresponds to a point on the first ellipse that is a given distance, approximately $2EPS$, away from the intersection point:

$$EPS_{\text{Radian}} = \arcsin\left(\frac{2 \cdot \text{EPS}}{\sqrt{x^2 + y^2}}\right) \tag{31}$$

The angle EPS_{Radian} is then used with the parametric form of the first ellipse to determine two points adjacent to (x, y):

$$\begin{aligned} x_1 &= A_1 \cdot \cos(\theta + EPS_{\text{Radian}}) \\ y_1 &= B_1 \cdot \sin(\theta + EPS_{\text{Radian}}) \\ x_2 &= A_1 \cdot \cos(\theta - EPS_{\text{Radian}}) \\ y_2 &= B_1 \cdot \sin(\theta - EPS_{\text{Radian}}) \end{aligned} \tag{32}$$

Each of the points is then evaluated in the second ellipse equation:

$$\text{test}_i = AA \cdot x_i^2 + BB \cdot x_i \cdot y_i + CC \cdot y_i^2 + DD \cdot x_i + EE \cdot y_i + FF, \quad i = 1, 2 \tag{33}$$

If the value of test_i is positive, then the point (x_i, y_i) is outside the second ellipse. It follows that the product of the two test-point evaluations $\text{test}_1\text{test}_2$ will be positive if the intersection point is a tangent, since at a tangent point both test points will be on the same side of the ellipse. The product of the test-point evaluations will be negative if the two ellipse curves cross at the intersection point, since the test points will be on opposite sides of the ellipse. The function ISTANPT implements this logic to check whether an intersection point is a tangent or a cross-point; pseudo-code is shown in Fig. 18.

When there are two intersection points, the ISTANPT function can be used to differentiate the case 2-3 (Fig. 8) from the cases 2-1 and 2-2. Either of the two known intersection points can be checked with ISTANPT. If the intersection point is a tangent, then both of the intersection points must be tangents, so the case is either 2-1 or 2-2, and one ellipse must be fully contained within the other. For cases 2-1 and 2-2, the geometric logic used for 0 or 1 intersection points can also be used, i.e., the function NOINTPTS can be used to determine the overlap area for these cases. If the two ellipse curves cross at the tested intersection point, then the case must be 2-3, representing a partial overlap between the two ellipse areas.

For case 2-3, with partial overlap between the two ellipses, the approach for finding overlap area is based on using the two points (x_1, y_1) and (x_2, y_2) with segment the algorithm (Table 2; Fig. 2) to determine the partial overlap area contributed by each ellipse. The total overlap area is the sum of the two segment areas. The two intersection points divide each ellipse into two segment areas (see Fig. 5). Only one sector area from each ellipse contributes to the overlap area. The segment algorithm returns the area between the secant line and the portion of the ellipse from the first point to the second point traversed in a counter-clockwise direction. For the overlap area calculation, the two points must be passed to the segment algorithm in the order that will return the correct segment area. The default order is counter-clockwise from the first point (x_1, y_1) to the second point (x_2, y_2). A check is made to determine whether this order will return the desired segment area. First,

the parametric angles corresponding to (x_1, y_1) and (x_2, y_2) on the first ellipse are determined, by the rules in Table 2.2:

$$\theta_1 = \begin{cases} \arccos(x_1/A_1) & y_1 \geq 0 \\ 2\pi - \arccos(x_1/A_1) & y_1 < 0 \end{cases} \tag{34}$$

$$\theta_2 = \begin{cases} \arccos(x_2/A_1) & y_2 \geq 0 \\ 2\pi - \arccos(x_2/A_1) & y_2 < 0 \end{cases} \tag{35}$$

Then, a point between (x_1, y_1) and (x_2, y_2) that is on the first ellipse is found:

$$\begin{aligned} x_{\text{mid}} &= A_1 \cdot \cos\left(\frac{\theta_1 + \theta_2}{2}\right) \\ y_{\text{mid}} &= B_1 \cdot \sin\left(\frac{\theta_1 + \theta_2}{2}\right) \end{aligned} \tag{36}$$

The point $(x_{\text{mid}}, y_{\text{mid}})$ is on the first ellipse between (x_1, y_1) and (x_2, y_2) when travelling counter- clockwise from (x_1, y_1) and (x_2, y_2). If $(x_{\text{mid}}, y_{\text{mid}})$ is inside the second ellipse, then the desired segment of the first ellipse contains the point $(x_{\text{mid}}, y_{\text{mid}})$. In this case, the segment algorithm should integrate in the default order, counterclockwise from (x_1, y_1) to (x_2, y_2). Otherwise, the order of the points should be reversed before calling the segment algorithm, causing it to integrate counterclockwise from (x_2, y_2) to (x_1, y_1). The area returned by the segment algorithm is the area contributed by the first ellipse to the partial overlap.

The desired segment from the second ellipse is found in a manner to the first ellipse segment. A slight difference in the approach is required because the segment algorithm is implemented for ellipses that are centered at the origin and oriented with the coordinate axes; but, in the general case the intersection points (x_1, y_1) and (x_2, y_2) lie on the second ellipse that is in a displaced and rotated location. The approach presented here translates and rotates the second ellipse to the origin so that the segment algorithm can be used. It suffices to translate then rotate the two intersection points by amounts that put the second ellipse centered at the origin and oriented with the coordinate axes:

$$\begin{aligned} x_{1\text{TR}} &= (x_1 - h_{2\text{TR}}) \cdot \cos(\varphi_1 - \varphi_2) + (y_1 - k_{2\text{TR}}) \cdot \sin(\varphi_2 - \varphi_1) \\ y_{1\text{TR}} &= (x_1 - h_{2\text{TR}}) \cdot \sin(\varphi_1 - \varphi_2) + (y_1 - k_{2\text{TR}}) \cdot \cos(\varphi_1 - \varphi_2) \\ x_{2\text{TR}} &= (x_2 - h_{2\text{TR}}) \cdot \cos(\varphi_1 - \varphi_2) + (y_2 - k_{2\text{TR}}) \cdot \sin(\varphi_2 - \varphi_1) \\ y_{2\text{TR}} &= (x_2 - h_{2\text{TR}}) \cdot \sin(\varphi_1 - \varphi_2) + (y_2 - k_{2\text{TR}}) \cdot \cos(\varphi_1 - \varphi_2) \end{aligned} \tag{37}$$

The new points $(x_{1\text{TR}}, y_{1\text{TR}})$ and $(x_{2\text{TR}}, y_{2\text{TR}})$ lie on the second ellipse after a translation+rotation that puts the second ellipse at the origin, oriented with the coordinate axes. The new points can be used as inputs to the segment algorithm to determine the overlap area contributed by the second ellipse. As with the first ellipse, the order of the points must be determined so that the segment algorithm returns the appropriate area. The default order is counter-clockwise from the first point $(x_{1\text{TR}}, y_{1\text{TR}})$ to the second point $(x_{2\text{TR}}, y_{2\text{TR}})$. A check is made to determine whether this order will return the desired segment area. First, the parametric angles corresponding to points $(x_{1\text{TR}}, y_{1\text{TR}})$ and $(x_{2\text{TR}}, y_{2\text{TR}})$ on the second ellipse are determined, by the rules in Table 2.2:

$$\theta_1 = \begin{cases} \arccos(x_{1\mathrm{TR}}/A_2) & y_{1\mathrm{TR}} \geq 0 \\ 2\pi - \arccos(x_{1\mathrm{TR}}/A_2) & y_{1\mathrm{TR}} < 0 \end{cases} \tag{38}$$

$$\theta_2 = \begin{cases} \arccos(x_{2\mathrm{TR}}/A_2) & y_{2\mathrm{TR}} \geq 0 \\ 2\pi - \arccos(x_{2\mathrm{TR}}/A_2) & y_{2\mathrm{TR}} < 0 \end{cases} \tag{39}$$

Then, a point on the second ellipse between $(x_{1\mathrm{TR}}, y_{1\mathrm{TR}})$ and $(x_{2\mathrm{TR}}, y_{2\mathrm{TR}})$ is found:

$$x_{\mathrm{mid}} = A_2 \cdot \cos\left(\frac{\theta_1 + \theta_2}{2}\right)$$

$$y_{\mathrm{mid}} = B_2 \cdot \sin\left(\frac{\theta_1 + \theta_2}{2}\right)$$

The point $(x_{\mathrm{mid}}, y_{\mathrm{mid}})$ is on the second ellipse between $(x_{1\mathrm{TR}}, y_{1\mathrm{TR}})$ and $(x_{2\mathrm{TR}}, y_{2\mathrm{TR}})$ when travelling counter- clockwise from $(x_{1\mathrm{TR}}, y_{1\mathrm{TR}})$ and $(x_{2\mathrm{TR}}, y_{2\mathrm{TR}})$. The new point (x_{mid}, y_{mid}) lies on the centered second ellipse. To determine the desired segment of the second ellipse, the new point $(x_{\mathrm{mid}}, y_{\mathrm{mid}})$ must be rotated then translated back to a corresponding position on the once-translated+rotated second ellipse:

$$x_{\mathrm{midRT}} = x_{\mathrm{mid}} \cdot \cos(\varphi_2 - \varphi_1) + y_{\mathrm{mid}} \cdot \sin(\varphi_1 - \varphi_2) + h_{2\mathrm{TR}}$$

$$y_{\mathrm{midRT}} = x_{\mathrm{mid}} \cdot \sin(\varphi_2 - \varphi_1) + y_{\mathrm{mid}} \cdot \cos(\varphi_1 - \varphi_2) + k_{2\mathrm{TR}}$$

If $(x_{\mathrm{mid}RT}, y_{\mathrm{midRT}})$ is inside the first ellipse, then the desired segment of the second ellipse contains the point $(x_{\mathrm{mid}}, y_{\mathrm{mid}})$. In this case, the segment algorithm should integrate in the default order, counterclockwise from $(x_{1\mathrm{TR}}, y_{1\mathrm{TR}})$ to $(x_{2\mathrm{TR}}, y_{2\mathrm{TR}})$. Otherwise, the order of the points should be reversed before calling the segment algorithm, causing it to integrate counterclockwise from $(x_{2\mathrm{TR}}, y_{2\mathrm{TR}})$ to $(x_{1\mathrm{TR}}, y_{1\mathrm{TR}})$. The area returned by the segment algorithm is the area contributed by the second ellipse to the partial overlap. The sum of the segment areas from the two ellipses is then equal to the ellipse overlap area. The TWOINTPTS function calculates the overlap area for partial overlap with two intersection points (Case 2-3); pseudo-code is shown in Fig. 15.

There are two possible sub-cases for three intersection points, shown in Fig. 9. One of the three points must be a tangent point, and the ellipses must cross at the other two points. The cases are distinct only in the sense that the tangent point occurs with ellipse 2 on the interior side of ellipse 1 (Case 3-1), or with ellipse 2 on the exterior side of ellipse 1 (Case 3-2). The overlap area calculation is performed in the same manner for both cases, by calling the TWOINTPTS function with the two cross-point intersections. The ISTANPT function can be used to determine which point is a tangent; the remaining two intersection points are then passed to TWOINTPTS. This logic is implemented in the THREEINTPTS function, with pseudo-code in Fig. 16.

There is only one possible case for four intersection points, shown in Fig. 9. The two ellipse curves must cross at all four of the intersection points, resulting in a partial overlap. The overlap area consists of two segments from each ellipse, and a central convex quadrilateral. For the approach presented here, the four intersection points are sorted ascending in a counter-clockwise order around the first ellipse.
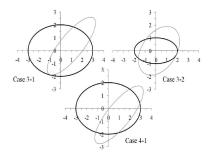
FIGURE 9. When three intersection points exist, one must be a tangent, and the ellipse curves must cross at the other two points, always resulting in a partial overlap. When four intersection points exist, the ellipse curves must cross at all four points, again resulting in a partial overlap

The ordered set of intersection points is (x_1, y_1), (x_2, y_2), (x_3, y_3) and (x_4, y_4). The ordering allows a direct calculation of the quadrilateral area. The standard formula uses the cross-product of the two diagonals:

$$
\begin{aligned}
\text{area} =& \frac{1}{2} \left| (x_3 - x_1, y_3 - y_1) \times (x_4 - x_2, y_4 - y_2) \right| \\
=& \frac{1}{2} \left| (x_3 - x_1) \cdot (y_4 - y_2) - (x_4 - x_2) \cdot (x_3 - x_1) \right|
\end{aligned}
\tag{40}
$$

The point ordering also simplifies the search for the appropriate segments of each ellipse that contribute to the overlap area.

Suppose that the first two sorted points (x_1, y_1) and (x_2, y_2) demarcate a segment of the first ellipse that contributes to the overlap area, as shown in Fig. 9 and Fig. 10. It follows that the contributing segments from the first ellipse are between (x_1, y_1) and (x_2, y_2), and also between (x_3, y_3) and (x_4, y_4). In this case, the contributing segments from the second ellipse are between (x_2, y_2) and (x_3, y_3), and between (x_4, y_4) and (x_1, y_1). To determine which segments contribute to the overlap area, it suffices to test whether a point midway between (x_1, y_1) and (x_2, y_2) is inside or outside the second ellipse. The segment algorithm is used for each of the four areas, and added to the quadrilateral to obtain the total overlap area.

FIGURE 10. Overlap Area with four intersection points (Case 4-1). The overlap area consists of two segments from each ellipse, and a central convex quadrilateral.

An implementation of the ELLIPSE_ELLIPSE_OVERLAP algorithm in c-code is shown in Appendix6. The code compiles under Cygwin-1.7.7-1, and returns the following values for the test cases presented above in Fig. 6, Fig. 7 , Fig. 8 and Fig. 9:

LISTING 5. Return values for the test cases presented above in Fig. 6, Fig. 7 , Fig. 8 and Fig. 9.

```
138  cc call\_ee.c ellipse\_ellipse\_overlap.c −o call\_ee.exe
139
140  ./ call\_ee
141
142  Calling ellipse\_ellipse\_overlap.c
143
144
145
146  Case 0−1: area =       6.28318531, return_value = 111
147
148           ellipse 2 area by pi*a2*b2 =      6.28318531
149
150  Case 0−2: area =       6.28318531, return_value = 110
151
152           ellipse 1 area by pi*a1*b1 =      6.28318531
153
154  Case 0−3: area =       0.00000000, return_value = 103
155
156           Ellipses are disjoint, ovelap area = 0.0
157
158
159
160  Case 1−1: area =       6.28318531, return_value = 111
161
162           ellipse 2 area by pi*a2*b2 =      6.28318531
163
164  Case 1−2: area =       6.28318531, return_value = 110
165
166           ellipse 1 area by pi*a1b1 =      6.28318531
167
168  Case 1−3: area =      −0.00000000, return_value = 107
169
170           Ellipses are disjoint, ovelap area = 0.0
171
172
173
174  Case 2−1: area =      10.60055478, return_value = 109
175
176           ellipse 2 area by pi*a2*b2 =     10.60287521
177
178  Case 2−2: area =       6.28318531, return_value = 110
179
180           ellipse 1 area by pi*a1b1 =      6.28318531
181
182  Case 2−3: area =       3.82254574, return_value = 107
183
184
185
186  Case 3−1: area =       7.55370392, return_value = 107
187
188  Case 3−2: area =       5.67996234, return_value = 107
189
190
191
192  Case 4−1: area =      16.93791852, return_value = 109
```

LISTING 6. The ELLIPSE_ELLIPSE_OVERLAP algorithm is shown for calculating the overlap area between two general ellipses. The algorithm calls several supporting functions, including the polynomial solvers BIQUADROOTS, CUBICROOTS and QUADROOTS, from CACM Algorithm 326 [2] . The remaining functions are outlined in figures below.

```
193
194  (Area, Code) ← ELLIPSE_ELLIPSE_OVERLAP (A1, B1, H1, K1, φ1, A2, B2, H2, K2, φ2)
195
196  do if (A1 = 0 or B1 = 0) OR (A2 = 0 or B2 = 0)
197
198      then return (−1, ERROR_ELLIPSE_PARAMETERS)              :DATA CHECK
199
200  do if (|φ1| > 2π)
201
202      then φ1 ← (φ1modulo 2π)
203
204  do if (|φ2| > 2π)
205
206      then φ2 ← (φ2modulo 2π)
207
208  H2_TR ← (H2 − H1)∗cos (φ1) + (K2 − K1)∗sin (φ1) :TRANS+ROT ELL2
209
210  K2_TR ← (H1 − H2)∗sin (φ1) + (K2 − K1)∗cos (φ1)
211
212  φ2R ← φ2 −− φ1
213
214  do if (|φ2R| > 2π)
215
216      then φ2R ← (φ2Rmodulo2π)
217
218  AA ← cos²(φ2R)/A2² + sin²(φ2R)/B2² :BUILD\, IMPLICIT\, COEFFS ELL2TR
219
220  BB ← 2∗cos (φ2R)∗sin (φ2R)/A2² −− 2∗cos (φ2R)∗sin (φ2R)/B2²
221
222  CC ← sin²(φ2R)/A2² + cos²(φ2R)/B2²
223
224  DD ← −2∗cos (φ2R)∗(cos (φ2R)∗H2_TR + sin (φ2R)∗K2_TR)/A2²
225
226      − 2∗sin (φ2R)∗(sin (φ2R)∗H2_TR − cos (φ2R)∗K2_TR)/B2²
227
228 EE ← −2∗sin (φ2R)∗ (cos (φ2R)∗H2_TR + sin (φ2R)∗K2_TR)/A2²
229
230      + 2∗cos (φ2R)∗ (sin (φ2R)∗H2_TR − cos (φ2R)∗K2_TR)/B2²
231
232 FF ← (−cos (φ2R)∗H2_TR − sin (φ2R)∗K2_TR)²/A2²
233
234      + (sin (φ2R)∗H2_TR − cos (φ2R)∗K2_TR)²/B2² − 1
235
236      :BUILD QUARTIC POLYNOMIAL COEFFICIENTS FROM THE TWO ELLIPSE EQNS
237
238  cy[4] ← A1⁴∗AA² + B1²∗(A1²∗(BB² − 2∗AA∗CC)+ B1²∗CC²)
239
240  cy[3] ← 2∗B1∗(B1²∗CC∗EE + A1²∗(BB∗DD − AA∗EE))
241
242  cy[2] ← A1²∗((B1²∗(2∗AA∗CC −− BB²) + DD² − 2∗AA∗FF)
243
244          −2∗A1²∗AA² + B1²∗(2∗CC∗FF + EE²)
245
246  cy[1] ← 2∗B1∗(A1²∗(AA∗EE −− BB∗DD) + EE∗FF)
247
248  cy[0] ← (A1∗(A1∗AA −−DD) + FF)∗(A1∗(A1∗AA + DD) + FF)
249
250  py[0] ← 1
251
252  do if (|cy[4]| > 0)                          :SOLVE QUARTIC EQ
253
254      then for i ← 0 to 3 by 1
```

```
255
256                      py[4−i]  ←  cy[i]/cy[4]
257
258              r[][]←  BIQUADROOTS  (py[])
259
260              nroots  ←  4
261
262        else  if  (|cy[3]| > 0)                          :SOLVE  CUBIC  EQ
263
264        then  for  i  ←  0  to  2  by  1
265
266                      py[3−i]  ←  cy[i]/cy[3]
267
268              r[][]  ←  CUBICROOTS  (py[])
269
270              nroots  ←  3
271
272        else  if  (|cy[2]| > 0)                    :SOLVE  QUADRATIC  EQ
273
274        then  for  i  ←  0  to  1  by  1
275
276                      py[2−i]  ←  cy[i]/cy[2]
277
278              r[][]  ←  QUADROOTS  (py[])
279
280              nroots  ←  2
281
282        else  if  (|cy[1]| > 0)                        :SOLVE  LINEAR  EQ
283
284        then  r[1][1]  ←  (−cy[0]/cy[1])
285
286              r[2][1]  ←  0
287
288              nroots  ←  1
289
290        else                                          :COMPLETELY  DEGENERATE  EQ
291
292              nroots  ←  0
293
294   nychk  ←  0                                       :IDENTIFY  REAL  ROOTS
295
296   for  i  ←  1  to  nroots  by  1
297
298       do  if  (|r[2][i]| < EPS)
299
300           then  nychk  ←  nychk  +  1
301
302                 ychk[nychk]  ←  r[1][i]*B1
303
304   for  j  ←  2  to  nychk  by  1                       :SORT  REAL  ROOTS
305
306       tmp0  ←  ychk[j]
307
308       for  k  ←  (j — 1)  to  1  by  −1
309
310           do  if  (ychk[k]  =  tmp0)
311
312               then  break
313
314               else  ychk[k+1]  ←  ychk[k]
315
316       ychk[k+1]  ←  tmp0
317
318   nintpts  ←  0                                     :FIND  INTERSECTION  POINTS
319
320   for  i  ←  1  to  nychk  by  1
321
322       do  if  ((i > 1)  and  (|ychk[i] − ychk[i−1]| <  EPS/2))
323
324           then  continue
325
326       do  if  (|ychk[i]| >  −B1)
327
```

```
328          then x1 ← 0
329
330          else x1←? A1∗sqrt (1.0 − ychk[i]²/B1²)
331
332     x2 ← −x1
333
334     do if (|ellipse2tr(x1, ychk[i], AA, BB, CC, DD, EE, FF)| < EPS/2)
335
336        then nintpts ← nintpts + 1
337
338             do if (nintpts > 4)
339
340                then return (−1, ERROR_INTERSECTION_PTS)
341
342             xint[nintpts] ← x1
343
344             yint[nintpts] ← ychk[i]
345
346     do if ((|ellipse2tr(x2, ychk[i], AA, BB, CC, DD, EE, FF)| < EPS/2)
347
348        and (|x2 − x1| > EPS/2))
349
350        then nintpts ← nintpts + 1
351
352             do if (nintpts > 4)
353
354                then return (−1, ERROR_INTERSECTION_PTS)
355
356             xint[nintpts] ← x1
357
358             yint[nintpts] ← ychk[i]
359
360  switch (nintpts)              :HANDLE ALL CASES FOR \# OF INTERSECTION PTS
361
362     case 0:
363
364     case 1:
365
366        (OverlapArea, Code) ← NOINTPTS (A1, B1, A2, B2, H1, K1, H2, K2, AA,
367
368            BB, CC, DD, EE, FF)
369
370        return (OverlapArea, Code)
371
372     case 2:
373
374        Code ← istanpt (xint[1], yint[1], A1, B1, AA, BB, CC, DD, EE, FF)
375
376        do if (Code == TANGENT_POINT)
377
378           then (OverlapArea, Code) ← NOINTPTS (A1, B1, A2, B2, H1, K1,
379
380                   H2, K2, AA, BB, CC, DD, EE, FF)
381
382           else (OverlapArea, Code) ← TWOINTPTS (xint[], yint[], A1,
383
384               PHI_1, A2, B2, H2_TR, K2_TR, PHI_2, AA, BB, CC, DD, EE, FF)
385
386        return (OverlapArea, Code)
387
388     case 3:
389
390        (OverlapArea, Code) ← THREEINTPTS (xint, yint, A1, B1, PHI_1,
391
392            A2, B2, H2_TR, K2_TR, PHI_2, AA, BB, CC, DD, EE, FF)
393
394        return (OverlapArea, Code)
395
396     case 4:
397
398        (OverlapArea, Code) ← FOURINTPTS (xint, yint, A1, B1, PHI_1,
399
400            A2,  B2, H2_TR, K2_TR, PHI_2, AA, BB, CC, DD, EE, FF)
```

```
401
402              return (OverlapArea,Code)
```

LISTING 7. The NOINTPTS subroutine. If there are either 0 or 1 intersection points, this function determines whether one ellipse is contained within the other (Cases 0-1, 0-2, 1-1 and 1-2), or if the ellipses are disjoint (Cases 0-3 and 1-3). The function returns the appropriate overlap area, and a code describing which case was encountered.

```
403
404   (OverlapArea,Code) ← NOINTPTS (A1,B1,A2,B2,H1,K1,H2_TR,K2_TR,AA,
405
406                                          BB,CC,DD,EE,FF)
407
408    relsize ← A1*B1 − A2*B2
409
410    do if (relsize > 0)
411
412       then do if (((H2_TR*H2_TR)/(A1*A1)+(K2_TR*K2_TR)/(B1*B1)) < 1.0)
413
414               then return (π*A2*B2,ELLIPSE2_INSIDE_ELLIPSE1)
415
416               else return (0, DISJOINT_ELLIPSES)
417
418       else do if (relsize < 0)
419
420               then do if (FF < 0)
421
422                       then return (π*A1*B1,ELLIPSE1_INSIDE_ELLIPSE2)
423
424                       else return (0, DISJOINT_ELLIPSES)
425
426       else do if ((H1 = H2_TR) AND (K1 = K2_TR))
427
428               then return (π*A1*B1, ELLIPSES_ARE_IDENTICAL)
429
430               else return (−1, ERROR_CALCULATIONS
```

LISTING 8. The TWOINTPTS subroutine. If there are 2 intersection points where the ellipse curves cross (Case 2-3), this function uses the ellipse sector algorithm to determine the contribution of each ellipse to the total overlap area. The function returns the appropriate overlap area, and a code indicating two intersection points.

```
431   (OverlapArea,Code) ← TWOINTPTS (xint[],yint[],A1,B1,φ1,A2,B2,H2_TR,
432
433                                        K2_TR,φ2,AA,BB,CC,DD,EE,FF)
434
435 do if (|x[1]| > A1)                     :AVOID INVERSE TRIG ERRORS
436
437    then do if (x[1] < 0)
438
439            then x[1] ← −A1
440
441            else x[1] ← A1
442
443 do if (y[1] < 0)                :FIND PARAMETRIC ANGLE FOR (x[1], y[1])
444
445    then θ1 ← 2π −− arccos (x[1]/A1)
446
447    else θ1 ← arccos (x[1]/A1)
448
449 do if (|x[2]| > A1)                     :AVOID INVERSE TRIG ERRORS
```

```
450
451     then do if (x[2] < 0)
452
453             then x[2] ← −A1
454
455             else x[2] ← A1
456
457 do if (y[2] < 0)                    :FIND PARAMETRIC ANGLE FOR (x[2], y[2])
458
459    then θ2 ← 2π −− arccos (x[2]/A1)
460
461    else θ2 ← arccos (x[2]/A1)
462
463 do if (θ1 > θ2)                              :GO CCW FROM θ1 TO\, θ2
464
465    then tmp ← θ1, θ1 ← θ2, θ2 ← tmp
466
467 xmid ← A1*cos ((θ1 + θ2)/2)
468
469 ymid ← B1*sin ((θ1 + θ2)/2)
470
471 do if (AA*xmid² +BB*xmid*ymid+CC*ymid² +DD*xmid+EE*ymid+FF > 0)
472
473    then tmp ← θ1, θ1← θ2, θ2 ← tmp
474
475 do if (θ1 > θ2)                 :SEGMENT ALGORITHM FOR ELLIPSE 1
476
477    then θ1 ? θ1 − 2π
478
479 do if ((θ2 − θ1) > π)
480
481    then trsign ← 1
482
483    else trsign ← −1
484
485 area1 ← (A1*B1*(θ2 − θ1) + trsign*\textbar x[1]*y[2] − x[2]*y[1])\textbar
        /2
486
487 x1\_tr ← (x[1] − H2\_TR)*cos(φ1 −− φ2) + (y[1] − K2\_TR)*sin(φ2 −− φ1)
488
489 y1\_tr ← (x[1] − H2\_TR)*sin(φ1 −− φ2) + (y[1] − K2\_TR)*cos(φ1 −− φ2)
490
491 x2\_tr ← (x[2] − H2\_TR)*cos(φ1 −− φ2) + (y[2] − K2\_TR)*sin(φ2 −− φ1)
492
493 y2\_tr ? (x[2] − H2\_TR)*sin(φ1 −− φ2) + (y[2] − K2\_TR)*cos(φ1 −− φ2)
494
495 do if (|x1_tr| > A2)              :AVOID INVERSE TRIG ERRORS
496
497    then do if (x1_tr < 0)
498
499            then x1_tr ← −A2
500
501            else x1_tr ← A2
502
503 do if (y1_tr < 0)            :FIND PARAMETRIC ANGLE FOR (x1_tr, y1_tr)
504
505    then θ1 ← 2π −− arccos (x1_tr/A2)
506
507    else θ1 ← arccos (x1_tr/A2)
508
509 do if (|x2_tr| > A2)                :AVOID INVERSE TRIG ERRORS
510
511    then do if (x2_tr < 0)
512
513            then x2_tr ← −A2
514
515            else x2_tr ← A2
516
517 do if (y2_tr < 0)            :FIND PARAMETRIC ANGLE FOR (x2_tr, y2_tr)
518
519    then θ2 ← 2π −− arccos (x2_tr/A2)
520
521    else θ2 ← arccos (x2_tr/A2)
```

```
522
523 do if (θ1 > θ2)                                    :GO CCW FROM θ1 TO\, θ2
524
525     then tmp ← θ1, θ1 ← θ2, θ2 ← tmp
526
527 xmid ← A2∗cos ((θ1 + θ2)/2)
528
529 ymid ← B2∗sin ((θ1 + θ2)/2)
530
531 xmid_rt = xmid∗cos(φ2 −− φ1) + ymid∗sin(φ1 −− φ2) + H2_TR
532
533 ymid_rt = xmid∗sin(φ2 −− φ1) + ymid∗cos(φ2 −− φ1) + K2_TR
534
535 do if (xmid_rt²/A1² + ymid_rt²/B1² > 1)
536
537     then tmp ← θ1, θ1 ← θ2, θ2 ← tmp
538
539 do if (θ1 > θ2)                        :SEGMENT ALGORITHM FOR ELLIPSE 2
540
541     then θ1 ← θ1 − 2π
542
543 do if ((θ2 − θ1) > π)
544
545      then trsign ← 1
546
547      else trsign ← −1
548
549 area2 ← (A2∗B2∗(θ2 − θ1)
550
551 + trsign∗|x1_tr ∗ y2_tr − x2_tr ∗ y1_tr)| /2
552
553  return (area1 + area2, TWO_INTERSECTION_POINTS)
```

LISTING 9. The THREEINTPTS subroutine. When there are three intersection points, one of the points must be a tangent point, and the ellipses must cross at the other two points. For the purpose of determining overlap area, the TWOINTPTS function can be used with the two cross-point intersections. The ISTANPT function can be used to determine which point is a tangent; the remaining two intersection points are then passed to TWOINTPTS. The function returns the appropriate overlap area, and a code indicating three intersection points.

```
554  OverlapArea, Code) ← THREEINTPTS (xint[], yint[], A1, B1, φ1, A2, B2, H2_TR,
555
556                                  K2_TR, φ2, AA, BB, CC, DD, EE, FF)
557 tanpts ← 0
558
559 for i ← 1 to nychk by 1
560
561     code ← ISTANPT ISTANPT (x[i], y[i], A1, B1, AA, BB, CC, DD, EE, FF)
562
563     do if (code = TANGENT_POINT)
564
565         then tanpts ← tanpts + 1
566
567              tanindex ← i
568
569 do if NOT (tanpts = 1)
570
571     then return (−1, ERROR_INTERSECTION_POINTS)
572
573 switch (tanindex)                           :STORE THE INTERSECTION POINTS
574
575     case 1:                              :TANGENT POINT IS IN (x[1], y[1])
576
577         xint[1] ← xint[3]
```

```
578
579          yint[1]  ←  yint[3]
580
581     case  2:                                    :TANGENT  POINT  IS  IN  (x[2],  y[2])
582
583          xint[2]  ←  xint[3]
584
585          yint[2]  ←  yint[3]
586
587 (OverlapArea,code)  ←  TWOINTPTS  (xint[],yint[],A1,B1,φ1,A2,B2,H2_TR,
588
589                                        K2_TR,φ2,AA,BB,CC,DD,EE,FF)
590
591 return  (OverlapArea,THREE_INTERSECTION_POINTS)
```

LISTING 10. The FOURINTPTS subroutine. When there are four intersection points, the ellipse curves must cross at all four points. A partial overlap area exists, consisting of two segments from each ellipse and a central quadrilateral. The function returns the appropriate overlap area, and a code indicating four intersection points.

```
592 verlapArea,Code)  ←  FOURINTPTS  (xint[],yint[],A1,B1,φ1,A2,B2,H2_TR,
593
594                                       K2_TR,φ2,AA,BB,CC,DD,EE,FF)
595
596  for  i  ←  1  to  4  by  1                          :AVOID  INVERSE  TRIG  ERRORS
597
598      do  if  (|xint[i]| > A1)
599
600          then do  if  (xint[i]  <  0)
601
602                   then  xint[i]  ←  −A1
603
604                   else  xint[i]  ←  A1
605
606      do  if  (yint[i]  <  0)                        :FIND  PARAMETRIC  ANGLES
607
608          then  θ[i]  ←  2π  −−  arccos  (xint[i]/A1)
609
610          else  θ[i]  ←  arccos  (xint[i]/A1)
611
612  for  j  ←  2  to  4  by  1                          :PUT  POINTS  IN  CCW  ORDER
613
614      tmp0  ←  θ[j]
615
616      tmp1  ←  xint[j]
617
618      tmp2  ←  yint[j]
619
620      for  k  ←  (j−1)  to  1  by  −1                 :INSERTION  SORT  BY  ANGLE
621
622          do  if  (θ[k]  <=  tmp0)
623
624              then  break
625
626              else  θ[k+1]  ←  θ[k]
627
628                    xint[k+1]  ←  xint[k]
629
630                    yint[k+1]  ←  yint[k]
631
632  area1  ←  (|(xint[3]  −−  xint[1])*(yint[4]  −−  yint[2])  −−
633
634 xint[4]  −  xint[2])*(yint[3]  −−  yint[1])|  /2)      :QUAD  AREA
635
636  for  i  ←  1  to  4  by  1                          :TRANSLATE+ROTATE  ELLIPSE  2
637
638      xint_tr[i]  ←  (xint[i]  −−  H2_TR)*cos  (φ1  −−  φ2)
```

```
639
640        + ( y i n t [ i ]  −−  K2_TR )∗ s i n  ( φ2 −−  φ1 )
641
642     y i n t _ t r [ i ]  ←  ( x i n t [ i ]  −−  H2_TR )∗ s i n  ( φ1 −−  φ2 )
643
644        + ( y i n t [ i ]  −−  K2_TR )∗ c o s  ( φ1 −−  φ2 )
645
646     do  i f  ( |xint_t r[i]| > A2 )      : AVOID  INVERSE  TRIG  ERRORS
647
648         then  do  i f  ( x i n t _ t r [ i ]  <  0 )
649
650                  then  x i n t _ t r [ i ]  ←  −A2
651
652                  e l s e  x i n t _ t r [ i ]  ←  A2
653
654     do  i f  ( y i n t _ t r [ i ]  <  0 )  : FIND  PARAM  ANGLES  FOR  ( x i n t _ t r ,  y i n t _ t r )
655
656         then  θ_t r [ i ]  ←  2π  −−  a r c c o s  ( x i n t _ t r [ i ] / A2 )
657
658         e l s e  θ_t r [ i ]  ←  a r c c o s  ( x i n t _ t r [ i ] / A2 )
659
660  xmid  ←  A1∗ c o s  ( ( θ1 +  θ2 ) / 2 )
661
662  ymid  ←  B1∗ s i n  ( ( θ1 +  θ2 ) / 2 )
663
664  do  i f  ( AA∗xmid² +BB∗xmid∗ymid+CC∗ymid² +DD∗xmid+EE∗ymid+FF  <  0 )
665
666     then  area2  =  ( A1∗B1∗( θ [ 2 ]  −  θ [ 1 ] )
667
668   −  | ( x i n t [ 1 ]∗ y i n t [ 2 ]  −  x i n t [ 2 ]∗ y i n t [ 1 ] ) | ) / 2
669
670         area3  =  ( A1∗B1∗( θ [ 4 ]  −  θ [ 3 ] )
671
672   −  | ( x i n t [ 3 ]∗ y i n t [ 4 ]  −  x i n t [ 4 ]∗ y i n t [ 3 ] ) | ) / 2
673
674         area4  =  ( A2∗B2∗( θ_t r [ 3 ]  −  θ_t r [ 2 ] )
675
676   −  | ( x i n t \_t r [ 2 ]∗ y i n t _ t r [ 3 ]  −  x i n t _ t r [ 3 ]∗ y i n t _ t r [ 2 ] ) |  ) / 2
677
678         area5  =  ( A2∗B2∗( θ_t r [ 1 ]  −  θ_t r [ 4 ]  −  t w o p i ) )
679
680   −  | ( x i n t _ t r [ 4 ]∗ y i n t _ t r [ 1 ]  −  x i n t _ t r [ 1 ]∗ y i n t _ t r [ 4 ] ) |  / 2 )
681
682     e l s e  area2  =  ( A1∗B1∗( θ [ 3 ]  −  θ [ 2 ] )
683
684    −  | ( x i n t [ 2 ]∗ y i n t [ 3 ]  −  x i n t [ 3 ]∗ y i n t [ 2 ] ) | ) / 2
685
686         area3  =  ( A1∗B1∗( θ [ 1 ]  −  ( θ [ 4 ]  −  t w o p i ) )
687
688   −  | ( x i n t [ 4 ]∗ y i n t [ 1 ]  −  x i n t [ 1 ]∗ y i n t [ 4 ] ) | ) / 2
689
690         area4  =  ( A2∗B2∗( θ_t r [ 2 ]  −  θ_t r [ 1 ] )
691
692   −  | ( x i n t _ t r [ 1 ]∗ y i n t _ t r [ 2 ]  −  x i n t _ t r [ 2 ]∗ y i n t _ t r [ 1 ] ) | ) / 2
693
694         area5  =  ( A2∗B2∗( θ_t r [ 4 ]  −  θ_t r [ 3 ] )
695
696   −  | ( x i n t _ t r [ 3 ]∗ y i n t _ t r [ 4 ]  −  x i n t _ t r [ 4 ]∗ y i n t _ t r [ 3 ] ) | ) / 2
697
698  return  ( area1+area2+area3+area4+area5 ,  FOUR_INTERSECTION_POINTS )
```

LISTING 11. The ISTANPT subroutine. Given an intersection point (x, y) that satisfies both Ellipse Eq. 21 and Ellipse Eq. 22, the function determines whether the two ellipse curves are tangent at (x, y), or if the ellipse curves cross at (x, y).

```
699  Code  ←  ISTANPT  ( x , y , A1 , B1 , AA , BB , CC , DD , EE , FF )
700
701  do  i f  ( |x| >  A1 )                      : AVOID  INVERSE  TRIG  ERRORS
702
```

```
703      then do if x < 0
704
705              then x ← −A1
706
707              else x ← A1
708
709  do if (y < 0)                          :FIND PARAMETRIC ANGLE FOR (x, y)
710
711      then θ ← 2π −− arccos (x/A1)
712
713      else θ ← arccos (x/A1)
714
715  branch ← v(x² + y²)                    :DETERMINE PERTURBATION ANGLE
716
717  do if (branch < 100∗EPS)
718
719      then eps_radian ← 2∗EPS
720
721      else eps_radian ← arcsin (2∗EPS/branch)
722
723  x1 ← A1∗cos (θ + eps_radian)     :CREATE TEST POINTS ON EACH SIDE
724
725  y1 ← B1∗cos (θ + eps_radian)     :OF THE INPUT POINT (x, y)
726
727  x2 ← A1∗cos (θ − eps_radian)
728
729  y2 ← B1∗cos (θ − eps_radian)
730
731  test1 ← AA∗x1²+BB∗x1∗y1+CC∗y1²+DD∗x1+EE∗y1+FF
732
733  test2 ← AA∗x2²+BB∗x2∗y2+CC∗y2²+DD∗x2+EE∗y2+FF
734
735  do if (test1∗test2 > 0)
736
737      then return TANGENT_POINT
738
739      else return INTERSECTION_POINT
```

LISTING 12. C-SOURCE CODE FOR ELLIPSE_SEGMENT

4. APPENDIX A.

```
740
741
742  /*
        ****************************************************************************
743
744  *
745
746  *   Function: double ellipse_segment
747
748  *
749
750  *   Purpose:  Given the parameters of an ellipse and two points that lie on
751
752  *             the ellipse, this function calculates the ellipse segment
         area
753
754  *             between the secant line and the ellipse.  Points are input as
755
756  *             (X1, Y1) and (X2, Y2), and the segment area is defined to be
757
758  *             between the secant line and the ellipse from the first point
759
760  *             (X1, Y1) to the second point (X2, Y2) in the counter−
         clockwise
761
762  *             direction.
763
764  *
765
```

```
766   *    Reference:  Hughes  and  Chraibi  (2011),  Calculating  Ellipse  Overlap  Areas
767
768   *
769
770   *    Dependencies:  math.h    for  calls  to  trig  and  absolute  value  functions
771
772   *                    program_constants.h   error  message  codes  and  constants
773
774   *
775
776   *    Inputs:     1.  double  A        ellipse  semi−axis  length  in  x−direction
777
778   *                2.  double  B         ellipse  semi−axis  length  in  y−direction
779
780   *                3.  double  X1       x−value  of  the  first  point  on  the  ellipse
781
782   *                4.  double  Y1       y−value  of  the  first  point  on  the  ellipse
783
784   *                5.  double  X2       x−value  of  the  second  point  on  the  ellipse
785
786   *                6.  double  Y2       y−value  of  the  second  point  on  the  ellipse
787
788   *
789
790   *    Outputs:   1.  int *MessageCode   stores  diagnostic  information
791
792   *                                   integer  codes  in  program_constants.h
793
794   *
795
796   *    Return:    The  value  of  the  ellipse  segment  area:
797
798   *                −1.0  is  returned  in  case  of  an  error  with  input  data
799
800   *
801
802   ****************************************************************************
          */
803
804
805
806   //   ══════════════════════════════════════════════════════════════════════════
807
808   //== INCLUDE  ANSI  C  SYSTEM  AND  USER−DEFINED  HEADER  FILES
          ══════════════════════════
809
810   //   ══════════════════════════════════════════════════════════════════════════
811
812   #include "program_constants.h"
813
814
815
816   double  ellipse_segment  (double  A,  double  B, double  X1,  double  Y1,  double  X2
          ,
817
818                            double  Y2,  int *MessageCode)
819
820   {
821
822       double  theta1;   //−− parametric  angle  of  the  first  point
823
824       double  theta2;   //−− parametric  angle  of  the  second  point
825
826       double  trsign;   //−− sign  of  the  triangle  area
827
828       double  pi = 2.0 * asin \eqref{GrindEQ__1_0_};   //−− a  maximum−
          precision  value  of  pi
829
830       double  twopi = 2.0 * pi;        //−− a  maximum−precision  value  of  2*pi
```

```
831
832
833
834            //--- Check the data first
835
836            //--- Each of the ellipse axis lengths must be positive
837
838            if (!(A > 0.0) \textbar \textbar !(B > 0.0))
839
840            {
841
842                (*MessageCode) = ERROR_ELLIPSE_PARAMETERS;
843
844                return -1.0;
845
846            }
847
848
849
850            //--- Points must be on the ellipse, within EPS, which is defined
851
852            //--- in the header file program_constants.h
853
854            if ( (fabs ((X1*X1)/(A*A) + (Y1*Y1)/(B*B) - 1.0) > EPS) textbar
                     textbar
855
856                (fabs ((X2*X2)/(A*A) + (Y2*Y2)/(B*B) - 1.0) > EPS) )
857
858            {
859
860                (*MessageCode) = ERROR_POINTS_NOT_ON_ELLIPSE;
861
862                return -1.0;
863
864            }
865
866
867
868            //--- Avoid inverse trig calculation errors: there could be an error
869
870            //--- if \textbar X1/A\textbar  > 1.0 or \textbar X2/A\textbar > 1.0
                     when calling acos()
871
872            //--- If execution arrives here, then the point is on the ellipse
873
874            //--- within EPS.  Try to adjust the value of X1 or X2 before giving
875
876            //--- up on the area calculation
877
878            if (fabs (X1)/A > 1.0)
879
880            {
881
882                //--- if execution arrives here, already know that \textbar X1\
                         textbar  > A
883
884                if ((fabs (X1) - A) > EPS)
885
886                {
887
888                    //--- if X1 is not close to A or -A, then give up
889
890                    (*MessageCode) = ERROR\_INVERSE\_TRIG;
891
892                    return -1.0;
893
894                }
895
896                else
897
898                {
899
900                    //--- nudge X1 back to A or -A, so acos() will work
```

```
901
902                    X1 = (X1 < 0) ? −A : A;
903
904            }
905
906        }
907
908
909
910        if (fabs (X2)/A > 1.0)
911
912        {
913
914            //−− if execution arrives here, already know that \textbar X2\
                        textbar  > A
915
916            if ((fabs (X2) − A) > EPS)
917
918            {
919
920                //−− if X2 is not close to A or −A, then give up
921
922                (∗MessageCode) = ERROR_INVERSE_TRIG;
923
924                return −1.0;
925
926            }
927
928            else
929
930            {
931
932                //−− nudge X2 back to A or −A, so acos() will work
933
934                X2 = (X2 < 0) ? −A : A;
935
936            }
937
938        }
939
940
941
942        //−− Calculate the parametric angles on the ellipse
943
944        //−− The parametric angles depend on the quadrant where each point
945
946        //−− is located.  See Table 1 in the reference.
947
948        if (Y1 < 0.0)    //−− Quadrant III or IV
949
950            theta1 = twopi − acos (X1 / A);
951
952        else                  //−− Quadrant I or II
953
954            theta1 = acos (X1 / A);
955
956
957
958        if (Y2 < 0.0)    //−− Quadrant III or IV
959
960            theta2 = twopi − acos (X2 / A);
961
962        else                  //−− Quadrant I or II
963
964            theta2 = acos (X2 / A);
965
966
967
968        //−− need to start the algorithm with theta1 < theta2
969
970        if (theta1 > theta2)
971
972            theta1 −= twopi;
```

```
973
974
975
976        //-- if the integration angle is less than pi, subtract the triangle
977
978        //-- area from the sector, otherwise add the triangle area.
979
980        if ((theta2 - theta1) > pi)
981
982            trsign = 1.0;
983
984        else
985
986            trsign = -1.0;
987
988
989
990        //-- The ellipse segment is the area between the line and the ellipse,
991
992        //-- calculated by finding the area of the radial sector minus the
             area
993
994        //-- of the triangle created by the center of the ellipse and the two
995
996        //-- points.  First term is for the ellipse sector; second term is for
997
998        //-- the triangle between the points and the origin.  Area calculation
999
1000       //-- is described in the reference.
1001
1002       (*MessageCode) = NORMAL_TERMINATION;
1003
1004       return ( 0.5*(A*B*(theta2 - theta1) + trsign*fabs (X1*Y2 - X2*Y1)) );
1005
1006
1007
1008  }
```

LISTING   13. C-SOURCE   CODE   FOR   EL-
LIPSE_LINE_OVERLAP

5. **APPENDIX B.**

```
1010
1011
1012  /*
             ***************************************************************************

1013
1014  *
1015
1016  *   Function: double ellipse_line_overlap
1017
1018  *
1019
1020  *   Purpose:  Given the parameters of an ellipse and two points on a line,
1021
1022  *             this function calculates the area between the two curves.  If
1023
1024  *             the line does not cross the ellipse, or if the line is
             tangent
1025
1026  *             to the ellipse, then this function returns an area of 0.0
1027
1028  *             If the line intersects the ellipse at two points, then the
1029
1030  *             function returns the area between the secant line and the
1031
1032  *             ellipse.  The line is considered to have a direction from
1033
1034  *             the first given point (X1,Y1) to the second given point (X2,
             Y2)
```

```
1035
1036   *            This function determines where the line crosses the ellipse
1037
1038   *            first, and where it crosses second.  The area returned is
1039
1040   *            between the secant line and the ellipse traversed counter-
1041
1042   *            clockwise from the first intersection point to the second
1043
1044   *            intersection point.
1045
1046   *
1047
1048   *    Reference: Hughes and Chraibi (2011), Calculating Ellipse Overlap Areas
1049
1050   *
1051
1052   *    Dependencies: math.h  for calls to trig and absolute value functions
1053
1054   *                  program_constants.h  error message codes and constants
1055
1056   *                  ellipse_segment.c  core algorithm for ellipse segment
1057          area
1058   *
1059
1060   *    Inputs:   1. double PHI    CCW rotation angle of the ellipse, radians
1061
1062   *              2. double A      ellipse semi-axis length in x-direction
1063
1064   *              3. double B      ellipse semi-axis length in y-direction
1065
1066   *              4. double H      horizontal offset of ellipse center
1067
1068   *              5. double K      vertical offset of ellipse center
1069
1070   *              6. double X1     x-value of the first point on the line
1071
1072   *              7. double Y1     y-value of the first point on the line
1073
1074   *              8. double X2     x-value of the second point on the line
1075
1076   *              9. double Y2     y-value of the second point on the line
1077
1078   *
1079
1080   *    Outputs:  1. int *MessageCode   returns diagnostic information
1081
1082   *                                    integer codes in program_constants.h
1083
1084   *
1085
1086   *    Return:   The value of the ellipse segment area:
1087
1088   *              -1.0 is returned in case of an error with the data or
1089
1090   *              calculation
1091
1092   *              0.0 is returned if the line does not cross the ellipse, or if
1093
1094   *              the line is tangent to the ellipse
1095
1096   *
1097
1098   **************************************************************************
1099          */
1100
1101
1102   //  _____
1103
```

```
1104    //== DEFINE PROGRAM CONSTANTS
               ========================================

1105
1106    //
               ================================================================


1107
1108    #include "program_constants.h"   //-- error message codes and constants
1109
1110
1111
1112    //
               ================================================================


1113
1114    //== DEPENDENT FUNCTIONS
               ==========================================

1115
1116    //
               ================================================================


1117
1118    double textbf{ellipse_segment} (double A, double B, double X1, double Y1,
            double X2,
1119
1120                            double Y2, int *MessageCode);
1121
1122
1123
1124    double \textbf{ellipse_line_overlap} (double PHI, double A, double B,
            double H,
1125
1126                                double K, double X1, double Y1, double X2,
1127
1128                                double Y2, int *MessageCode)
1129
1130    \{
1131
1132        //
                ================================================================


1133
1134        //== DEFINE LOCAL VARIABLES
                   ====================================

1135
1136        //
                ================================================================


1137
1138        double X10;        //-- Translated, Rotated x-value of the first point
1139
1140        double Y10;        //-- Translated, Rotated y-value of the first point
1141
1142        double X20;        //-- Translated, Rotated x-value of the second point
1143
1144        double Y20;        //-- Translated, Rotated y-value of the second point
1145
1146        double cosphi = textbf{cos} (PHI);   //-- store cos(PHI) to avoid
                   multiple calcs
1147
1148        double sinphi = \textbf{sin} (PHI);   //-- store sin(PHI) to avoid
                   multiple calcs
1149
1150        double m;          //-- line slope, calculated from input line slope
1151
1152        double a, b, c;    //-- quadratic equation coefficients a*x\^{}2 + b*x
                   + c
1153
1154        double discrim;    //-- quadratic equationdiscriminant b\^{}2 - 4*a*c
1155
1156        double x1, x2;     //-- x-values of intersection points
1157
1158        double y1, y2;     //-- y-values of intersection points
```

```
1159
1160        double mid_X;       //-- midpoint of the rotated x-values on the line
1161
1162        double theta1parm;  //-- parametric angle of first point
1163
1164        double theta2parm;  //-- parametric angle of second point
1165
1166        double xmidpoint;   //-- x-value midpoint of secant line
1167
1168        double ymidpoint;   //-- y-value midpoint of secant line
1169
1170        double root1, root2;  //-- temporary storage variables for roots
1171
1172        double segment_area;  //-- stores the ellipse segment area
1173
1174
1175
1176        //-- Check the data first
1177
1178        //-- Each of the ellipse axis lengths must be positive
1179
1180        if (!(A > 0.0) \textbar \textbar !(B > 0.0))
1181
1182        {
1183
1184            (*MessageCode) = ERROR_ELLIPSE_PARAMETERS;
1185
1186            return -1.0;
1187
1188        }
1189
1190
1191
1192        //-- The rotation angle for the ellipse should be between -2pi and 2pi
                   (?)
1193
1194        if ( (\textbf{fabs} (PHI) > (2.0*pi)) )
1195
1196            PHI = \textbf{fmod} (PHI, twopi);
1197
1198
1199
1200        //-- For this numerical routine, the ellipse will be translated and
1201
1202        //-- rotated so that it is centered at the origin and oriented with
1203
1204        //-- the coordinate axes.
1205
1206        //-- Then, the ellipse will have the implicit (polynomial) form of
1207
1208        //--    x\^{}2/A\^{}2 + y+2/B\^{}2 = 1
1209
1210
1211
1212        //-- For the line, the given points are first translated by the amount
1213
1214        //-- required to put the ellipse at the origin, e.g., by (-H, -K).
1215
1216        //-- Then, the points are rotated by the amount required to orient
1217
1218        //-- the ellipse with the coordinate axes, e.g., through the angle -
                   PHI.
1219
1220        X10 = cosphi*(X1 - H) + sinphi*(Y1 - K);
1221
1222        Y10 = -sinphi*(X1 - H) + cosphi*(Y1 - K);
1223
1224        X20 = cosphi*(X2 - H) + sinphi*(Y2 - K);
1225
1226        Y20 = -sinphi*(X2 - H) + cosphi*(Y2 - K);
1227
1228
1229
```

```
1230          //--- To determine if the line and ellipse intersect, solve the two
1231
1232          //--- equations simultaneously, by substituting y = Y10 + m*(x - X10)
1233
1234          //--- and x = X10 + mxy*(y - Y10) into the ellipse equation,
1235
1236          //--- which results in two quadratic equations in x.  See the reference
1237
1238          //--- for derivations of the quadratic coefficients.
1239
1240
1241
1242          //--- If the new line is not close to being vertical, then use the
1243
1244          //--- first derivation
1245
1246          if (\textbf{fabs} (X20 - X10) > EPS)
1247
1248          {
1249
1250              //---    ((B\^{}2 + A\^{}2*m\^{}2)/(A\^{}2)) * x\^{}2
1251
1252              //---    2*(Y10*m - m\^{}2*X10) * x
1253
1254              //---    (Y10\^{}2 - 2*m*Y10*X10 + m\^{}2*X10\^{}2 - B\^{}2)
1255
1256              m = (Y20 - Y10)/(X20 - X10);
1257
1258              a = (B*B + A*A*m*m)/(A*A);
1259
1260              b = 2.0*(Y10*m - m*m*X10);
1261
1262              c = (Y10*Y10 - 2.0*m*Y10*X10 + m*m*X10*X10 - B*B);
1263
1264          }
1265
1266          //--- If the new line is close to being vertical, then use the
1267
1268          //--- second derivation
1269
1270          else if (\textbf{fabs} (Y20 - Y10) > EPS)
1271
1272          {
1273
1274              //---    ((A\^{}2 + B\^{}2*m\^{}2)/(B\^{}2)) * y\^{}2
1275
1276              //---    2*(X10*m - m\^{}2*Y10) * y
1277
1278              //---    (X10\^{}2 - 2*m*Y10*X10 + m\^{}2*Y10\^{}2 - A\^{}2)
1279
1280              m = (X20 - X10)/(Y20 - Y10);
1281
1282              a = (A*A + B*B*m*m)/(B*B);
1283
1284              b = 2.0*(X10*m - m*m*Y10);
1285
1286              c = (X10*X10 - 2.0*m*Y10*X10 + m*m*Y10*Y10 - A*A);
1287
1288          }
1289
1290          //--- If the two given points on the line are very close together in
1291
1292          //--- both x and y directions, then give up
1293
1294          else
1295
1296          {
1297
1298              (*MessageCode) = ERROR_LINE_POINTS;
1299
1300              return -1.0;
1301
1302          }
```

```
1303
1304
1305
1306        //-- Once the coefficients for the Quadratic Equation in x are
1307
1308        //-- known, the roots of the quadratic polynomial will represent
1309
1310        //-- the x- or y-values of the points of intersection of the line
1311
1312        //-- and the ellipse.  The discriminant can be used to discern
1313
1314        //-- which case has occurred for the given inputs:
1315
1316        //--     1. discr < 0
1317
1318        //--        Quadratic has complex conjugate roots.
1319
1320        //--        The line and ellipse do not intersect
1321
1322        //--     2. discr = 0
1323
1324        //--        Quadratic has one repeated root
1325
1326        //--        The line and ellipse intersect at only one point
1327
1328        //--        i.e., the line is tangent to the ellipse
1329
1330        //--     3. discr > 0
1331
1332        //--        Quadratic has two distinct real roots
1333
1334        //--        The line crosses the ellipse at two points
1335
1336        discrim = b*b - 4.0*a*c;
1337
1338        if (discrim < 0.0)
1339
1340        {
1341
1342            //-- Line and ellipse do not intersect
1343
1344            (*MessageCode) = NO_INTERSECTION_POINTS;
1345
1346            return 0.0;
1347
1348        }
1349
1350        else if (discrim > 0.0)
1351
1352        {
1353
1354            //-- Two real roots exist, so calculate them
1355
1356            //-- The larger root is stored in root2
1357
1358            root1 = (-b - \textbf{sqrt} (discrim)) / (2.0*a);
1359
1360            root2 = (-b + \textbf{sqrt} (discrim)) / (2.0*a);
1361
1362        }
1363
1364        else
1365
1366        {
1367
1368            //-- Line is tangent to the ellipse
1369
1370            (*MessageCode) = LINE_TANGENT_TO_ELLIPSE;
1371
1372            return 0.0;
1373
1374        }
1375
```

```
1376
1377
1378          //-- decide which roots go into which x or y values
1379
1380          if (\textbf{fabs} (X20 - X10) > EPS) //-- roots are x-values
1381
1382          {
1383
1384               //-- order the points in the same direction as X10 -> X20
1385
1386               if (X10 < X20)
1387
1388               {
1389
1390                    x1 = root1;
1391
1392                    x2 = root2;
1393
1394               }
1395
1396               else
1397
1398               {
1399
1400                    x1 = root2;
1401
1402                    x2 = root1;
1403
1404               }
1405
1406
1407
1408               //-- The y-values can be calculated by substituting the
1409
1410               //-- x-values into the line equation y = Y10 + m*(x - X10)
1411
1412               y1 = Y10 + m*(x1 - X10);
1413
1414               y2 = Y10 + m*(x2 - X10);
1415
1416          }
1417
1418          else    //-- roots are y-values
1419
1420          {
1421
1422               //-- order the points in the same direction as Y10 -> Y20
1423
1424               if (Y10 < Y20)
1425
1426               {
1427
1428                    y1 = root1;
1429
1430                    y2 = root2;
1431
1432               }
1433
1434               else
1435
1436               {
1437
1438                    y1 = root2;
1439
1440                    y2 = root1;
1441
1442               }
1443
1444
1445
1446               //-- The x-values can be calculated by substituting the
1447
1448               //-- y-values into the line equation x = X10 + m*(y - Y10)
```

```
1449
1450            x1 = X10 + m*(y1 − Y10);
1451
1452            x2 = X10 + m*(y2 − Y10);
1453
1454        }
1455
1456
1457
1458        //−− Arriving here means that two points of intersection have been
1459
1460        //−− found. Pass the ellipse parameters and intersection points to
1461
1462        //−− the ellipse_segment() routine.
1463
1464        segment_area = \textbf{ellipse_segment} (A, B, x1, y1, x2, y2,
                MessageCode);
1465
1466
1467
1468        //−− The message code will indicate whether the function encountered
1469
1470        //−− any errors
1471
1472        if ((*MessageCode) < 0)
1473
1474        {
1475
1476            return −1;
1477
1478        }
1479
1480        else
1481
1482        {
1483
1484            (*MessageCode) = TWO_INTERSECTION_POINTS;
1485
1486            return segment_area;
1487
1488        }
1489
1490  }
```

LISTING   14. C-SOURCE   CODE   FOR   EL-
LIPSE_ELLIPSE_OVERLAP

6. APPENDIX C.

```
1491  /*
         ****************************************************************************
1492
1493  *
1494
1495  *   Function: double ellipse_ellipse_overlap
1496
1497  *
1498
1499  *   Purpose:   Given the parameters of two ellipses, this function calculates
1500
1501  *              the area of overlap between the two curves.  If the ellipses
             are
1502
1503  *              disjoint, this function returns 0.0; if one ellipse is
             contained
1504
1505  *              within the other, this function returns the area of the
             enclosed
1506
1507  *              ellipse; if the ellipses intersect, this function returns the
1508
```

```
1509  *              calculated  area  of  overlap.
1510
1511  *
1512
1513  *    Reference:  Hughes  and  Chraibi  (2011),  Calculating  Ellipse  Overlap  Areas
1514
1515  *
1516
1517  *    Dependencies:  math.h   for  calls  to  trig  and  absolute  value  functions
1518
1519  *                     program_constants.h   error  message  codes  and  constants
1520
1521  *
1522
1523  *    Inputs:    1.  double  PHI_1   CCW  rotation  angle  of  first  ellipse ,  radians
1524
1525  *               2.  double  A1      semi−axis  length  in  x−direction  first  ellipse
1526
1527  *               3.  double  B1      semi−axis  length  in  y−direction  first  ellipse
1528
1529  *               4.  double  H1      horizontal  offset  of  center  first  ellipse
1530
1531  *               5.  double  K1      vertical  offset  of  center  first  ellipse
1532
1533  *               6.  double  PHI_2   CCW  rotation  angle  of  second  ellipse ,  radians
1534
1535  *               7.  double  A2      semi−axis  length  in  x−direction  second
1536            ellipse
1537  *               8.  double  B2      semi−axis  length  in  y−direction  second
1538            ellipse
1539  *               9.  double  H2      horizontal  offset  of  center  second  ellipse
1540
1541  *              10.  double  K2      vertical  offset  of  center  second  ellipse
1542
1543  *
1544
1545  *    Outputs:   1.  int  *rtnCode   returns  diagnostic  information  integer  code
1546
1547  *                                   integer  codes  in  program_constants.h
1548
1549  *
1550
1551  *    Return:    The  calculated  value  of  the  overlap  area
1552
1553  *               −1  is  returned  in  case  of  an  error  with  the  calculation
1554
1555  *                0  is  returned  if  the  ellipses  are  disjoint
1556
1557  *                pi*A*B  of  smaller  ellipse  if  one  ellipse  is  contained  within
1558
1559  *                    the  other  ellipse
1560
1561  *
1562
1563  ****************************************************************************
             */
1564
1565
1566
1567  // _____


1568
1569  //== DEFINE PROGRAM CONSTANTS
             _____

1570
1571  // _____


1572
1573  #include  ”program_constants.h”   //−− error  message  codes  and  constants
```

```
1574
1575
1576
1577  //
         ═══════════════════════════════════════════════════════════

1578
1579  //== DEPENDENT  FUNCTIONS
         ═══════════════════════════════════════════════════════

1580
1581  //
         ═══════════════════════════════════════════════════════════

1582
1583  double  nointpts  (double  A1,  double  B1,  double  A2,  double  B2,  double  H1,

1584
1585                    double  K1,  double  H2_TR,  double  K2_TR,  double  AA,  double
                            BB,
1586
1587                    double  CC,  double  DD,  double  EE,  double  FF,  int  *rtnCode);
1588
1589
1590
1591  double  twointpts  (double  xint[],  double  yint[],  double  A1,  double  B1,

1592
1593                     double  PHI_1,  double  A2,  double  B2,  double  H2_TR,

1594
1595                     double  K2_TR,  double  PHI_2,  double  AA,  double  BB,

1596
1597                     double  CC,  double  DD,  double  EE,  double  FF,  int  *rtnCode)
                             ;
1598
1599
1600
1601  double  threeintpts  (double  xint[],  double  yint[],  double  A1,  double  B1,

1602
1603                      double  PHI_1,  double  A2,  double  B2,  double  H2_TR,

1604
1605                      double  K2_TR,  double  PHI_2,  double  AA,  double  BB,

1606
1607                      double  CC,  double  DD,  double  EE,  double  FF,

1608
1609                      int  *rtnCode);
1610
1611
1612
1613  double  fourintpts  (double  xint[],  double  yint[],  double  A1,  double  B1,

1614
1615                     double  PHI_1,  double  A2,  double  B2,  double  H2_TR,

1616
1617                     double  K2_TR,  double  PHI_2,  double  AA,  double  BB,

1618
1619                     double  CC,  double  DD,  double  EE,  double  FF,  int  *rtnCode
                             );
1620
1621
1622
1623  int  istanpt  (double  x,  double  y,  double  A1,  double  B1,  double  AA,  double  BB
           ,
1624
1625            double  CC,  double  DD,  double  EE,  double  FF);
1626
1627
1628
1629  double  ellipse2tr  (double  x,  double  y,  double  AA,  double  BB,

1630
1631                    double  CC,  double  DD,  double  EE,  double  FF);
1632
1633
1634
1635  //—— functions  for  solving  the  quartic  equation  from  Netlib/TOMS
1636
1637  void  BIQUADROOTS  (double  p[],  double  r[][5]);
```

```
1638
1639  void CUBICROOTS (double p[], double r[][5]);
1640
1641  void QUADROOTS (double p[], double r[][5]);
1642
1643
1644
1645  //_____

1646
1647  //== ELLIPSE–ELLIPSE OVERLAP
           _____
1648
1649  //_____

1650
1651  double ellipse_ellipse_overlap (double PHI_1, double A1, double B1,
1652
1653                                    double H1, double K1, double PHI_2,
1654
1655                                    double A2, double B2, double H2, double K2,
1656
1657                                    int *rtnCode)
1658
1659  {
1660
1661      //_____

1662
1663      //== DEFINE LOCAL VARIABLES
               _____
1664
1665      //_____

1666
1667      int i, j, k, nroots, nychk, nintpts, fnRtnCode;
1668
1669      double AA, BB, CC, DD, EE, FF, H2_TR, K2_TR, A22, B22, PHI_2R;
1670
1671      double cosphi, cosphi2, sinphi, sinphi2, cosphisinphi;
1672
1673      double tmp0, tmp1, tmp2, tmp3;
1674
1675      double cy[5] = {0.0}, py[5] = {0.0}, r[3][5] = {0.0};
1676
1677      double x1, x2, y12, y22;
1678
1679      double ychk[5] = {0.0}, xint[5], yint[5];
1680
1681      double Area1, Area2, OverlapArea;
1682
1683
1684
1685      //_____

1686
1687      //== DATA CHECK
               _____
1688
1689      //_____

1690
1691      //-- Each of the ellipse axis lengths must be positive
1692
1693      if ( (!(A1 > 0.0) \textbar \textbar !(B1 > 0.0)) \textbar \textbar (!(
              A2 > 0.0) \textbar \textbar !(B2 > 0.0)) )
1694
```

```
1695        {
1696
1697            (*rtnCode) = ERROR_ELLIPSE_PARAMETERS;
1698
1699            return -1.0;
1700
1701        }
1702
1703
1704
1705        //--- The rotation angles should be between -2pi and 2pi (?)
1706
1707        if ( (fabs (PHI_1) > (twopi)) )
1708
1709            PHI_1 = fmod (PHI_1, twopi);
1710
1711        if ( (fabs (PHI_2) > (twopi)) )
1712
1713            PHI_2 = fmod (PHI_2, twopi);
1714
1715
1716
1717        // ==================================================================

1718
1719        //== DETERMINE THE TWO ELLIPSE EQUATIONS FROM INPUT PARAMETERS
                ==========
1720
1721        // ==================================================================

1722
1723        //--- Finding the points of intersection between two general ellipses
1724
1725        //--- requires solving a quartic equation. Before attempting to solve
                the
1726
1727        //--- quartic, several quick tests can be used to eliminate some cases
1728
1729        //--- where the ellipses do not intersect. Optionally, can whittle away
1730
1731        //--- at the problem, by addressing the easiest cases first.
1732
1733
1734
1735        //--- Working with the translated+rotated ellipses simplifies the
1736
1737        //--- calculations. The ellipses are translated then rotated so that
                the
1738
1739        //--- first ellipse is centered at the origin and oriented with the
1740
1741        //--- coordinate axes. Then, the first ellipse will have the implicit
1742
1743        //--- (polynomial) form of
1744
1745        //---   x\^{}2/A1\^{}2 + y+2/B1\^{}2 = 1
1746
1747
1748
1749        //--- For the second ellipse, the center is first translated by the
                amount
1750
1751        //--- required to put the first ellipse at the origin, e.g., by (-H1, -
                K1)
1752
1753        //--- Then, the center of the second ellipse is rotated by the amount
1754
1755        //--- required to orient the first ellipse with the coordinate axes, e.g
                .,
1756
1757        //--- through the angle -PHI_1.
```

```
1758
1759        //—— The translated and rotated center point coordinates for the second
1760
1761        //—— ellipse are found with the rotation matrix, derivations are
1762
1763        //—— described in the reference.
1764
1765        cosphi = cos (PHI_1);
1766
1767        sinphi = sin (PHI_1);
1768
1769        H2_TR = (H2 − H1)*cosphi + (K2 − K1)*sinphi;
1770
1771        K2_TR = (H1 − H2)*sinphi + (K2 − K1)*cosphi;
1772
1773        PHI_2R = PHI_2 − PHI_1;
1774
1775        if ( (fabs (PHI_2R) > (twopi)) )
1776
1777            PHI_2R = fmod (PHI_2R, twopi);
1778
1779
1780
1781        //—— Calculate implicit (Polynomial) coefficients for the second
                    ellipse
1782
1783        //—— in its translated−by (−H1, −H2) and rotated−by −PHI_1 postion
1784
1785        //——          AA*x^{}2 + BB*x*y + CC*y^{}2 + DD*x + EE*y + FF = 0
1786
1787        //—— Formulas derived in the reference
1788
1789        //—— To speed things up, store multiply−used expressions first
1790
1791        cosphi = cos (PHI_2R);
1792
1793        cosphi2 = cosphi*cosphi;
1794
1795        sinphi = sin (PHI_2R);
1796
1797        sinphi2 = sinphi*sinphi;
1798
1799        cosphisinphi = 2.0*cosphi*sinphi;
1800
1801        A22 = A2*A2;
1802
1803        B22 = B2*B2;
1804
1805        tmp0 = (cosphi*H2_TR + sinphi*K2_TR)/A22;
1806
1807        tmp1 = (sinphi*H2_TR − cosphi*K2_TR)/B22;
1808
1809        tmp2 = cosphi*H2_TR + sinphi*K2_TR;
1810
1811        tmp3 = sinphi*H2_TR − cosphi*K2_TR;
1812
1813
1814
1815        //—— implicit polynomial coefficients for the second ellipse
1816
1817        AA = cosphi2/A22 + sinphi2/B22;
1818
1819        BB = cosphisinphi/A22 − cosphisinphi/B22;
1820
1821        CC = sinphi2/A22 + cosphi2/B22;
1822
1823        DD = −2.0*cosphi*tmp0 − 2.0*sinphi*tmp1;
1824
1825        EE = −2.0*sinphi*tmp0 + 2.0*cosphi*tmp1;
1826
1827        FF = tmp2*tmp2/A22 + tmp3*tmp3/B22 − 1.0;
1828
1829
```

```
1830
1831        //───────────────────────────────────────────────────────────────

1832
1833        //== CREATE AND SOLVE THE QUARTIC EQUATION TO FIND INTERSECTION POINTS
                  ==
1834
1835        //───────────────────────────────────────────────────────────────

1836
1837        //── If execution arrives here, the ellipses are at least 'close' to
1838
1839        //── intersecting.
1840
1841        //── Coefficients for the Quartic Polynomial in y are calculated from
1842
1843        //── the two implicit equations.
1844
1845        //── Formulas for these coefficients are derived in the reference.
1846
1847        cy[4] = pow (A1, 4.0)*AA*AA + B1*B1*(A1*A1*(BB*BB − 2.0*AA*CC)
1848
1849                  + B1*B1*CC*CC);
1850
1851        cy[3] = 2.0*B1*(B1*B1*CC*EE + A1*A1*(BB*DD − AA*EE));
1852
1853        cy[2] = A1*A1*((B1*B1*(2.0*AA*CC − BB*BB) + DD*DD − 2.0*AA*FF)
1854
1855                       − 2.0*A1*A1*AA*AA) + B1*B1*(2.0*CC*FF + EE*EE);
1856
1857        cy[1] = 2.0*B1*(A1*A1*(AA*EE − BB*DD) + EE*FF);
1858
1859        cy[0] = (A1*(A1*AA − DD) + FF)*(A1*(A1*AA + DD) + FF);
1860
1861
1862
1863        //── Once the coefficients for the Quartic Equation in y are known, the
1864
1865        //── roots of the quartic polynomial will represent y−values of the
1866
1867        //── intersection points of the two ellipse curves.
1868
1869        //── The quartic sometimes degenerates into a polynomial of lesser
1870
1871        //── degree, so handle all possible cases.
1872
1873        if (fabs (cy[4]) > 0.0)
1874
1875        {
1876
1877            //== QUARTIC COEFFICIENT NONZERO, USE QUARTIC FORMULA
                   ================
1878
1879            for (i = 0; i <= 3; i++)
1880
1881                py[4−i] = cy[i]/cy[4];
1882
1883            py[0] = 1.0;
1884

1885

1886
1887            BIQUADROOTS (py, r);
1888
1889            nroots = 4;
1890
1891        }
1892
1893        else if (fabs (cy[3]) > 0.0)
1894
1895        {
1896
```

```
1897              //== QUARTIC DEGENERATES TO CUBIC, USE CUBIC FORMULA
                     ================
1898
1899              for (i = 0; i <= 2; i++)
1900
1901                    py[3-i] = cy[i]/cy[3];
1902
1903              py[0] = 1.0;
1904
1905
1906
1907              CUBICROOTS (py, r);
1908
1909              nroots = 3;
1910
1911          }
1912
1913      else if (fabs (cy[2]) > 0.0)
1914
1915          {
1916
1917              //== QUARTIC DEGENERATES TO QUADRATIC, USE QUADRATIC FORMULA
                     ========
1918
1919              for (i = 0; i <= 1; i++)
1920
1921                    py[2-i] = cy[i]/cy[2];
1922
1923              py[0] = 1.0;
1924
1925
1926
1927              QUADROOTS (py, r);
1928
1929              nroots = 2;
1930
1931          }
1932
1933      else if (fabs (cy[1]) > 0.0)
1934
1935          {
1936
1937              //== QUARTIC DEGENERATES TO LINEAR: SOLVE DIRECTLY
                     ==================
1938
1939              //-- cy[1]*Y + cy[0] = 0
1940
1941              r[1][1] = (-cy[0]/cy[1]);
1942
1943              r[2][1] = 0.0;
1944
1945              nroots = 1;
1946
1947          }
1948
1949      else
1950
1951          {
1952
1953              //== COMPLETELY DEGENERATE QUARTIC: ELLIPSES IDENTICAL???
                     ============
1954
1955              //-- a completely degenerate quartic, which would seem to
1956
1957              //-- indicate that the ellipses are identical. However, some
1958
1959              //-- configurations lead to a degenerate quartic with no
1960
1961              //-- points of intersection.
1962
1963              nroots = 0;
1964
1965          }
```

```
1966
1967
1968
1969     //  ═══════════════════════════════════════════════════════════════

1970
1971     //== CHECK ROOTS OF THE QUARTIC: ARE THEY POINTS OF INTERSECTION?
              ═══════
1972
1973     //  ═══════════════════════════════════════════════════════════════

1974
1975     //-- determine which roots are real, discard any complex roots
1976
1977     nychk = 0;
1978
1979     for (i = 1; i <= nroots; i++)
1980
1981     {
1982
1983         if (fabs (r[2][i]) < EPS)
1984
1985         {
1986
1987             nychk++;
1988
1989             ychk[nychk] = r[1][i]*B1;
1990
1991         }
1992
1993     }
1994
1995
1996
1997     //-- sort the real roots by straight insertion
1998
1999     for (j = 2; j <= nychk; j++)
2000
2001     {
2002
2003         tmp0 = ychk[j];
2004
2005
2006
2007         for (k = j - 1; k >= 1; k--)
2008
2009         {
2010
2011             if (ychk[k] <= tmp0)
2012
2013                 break;
2014
2015
2016
2017             ychk[k+1] = ychk[k];
2018
2019         }
2020
2021
2022
2023         ychk[k+1] = tmp0;
2024
2025     }
2026
2027
2028
2029     //-- determine whether polynomial roots are points of intersection
2030
2031     //-- for the two ellipses
2032
2033     nintpts = 0;
```

```
2034
2035        for (i = 1; i <= nychk; i++)
2036
2037        {
2038
2039            //—— check for multiple roots
2040
2041            if ((i > 1) \&\& (fabs (ychk[i] − ychk[i−1]) < (EPS/2.0)))
2042
2043                continue;
2044
2045
2046
2047            //—— check intersection points for ychk[i]
2048
2049            if (fabs (ychk[i]) > B1)
2050
2051                x1 = 0.0;
2052
2053            else
2054
2055                x1 = A1*sqrt (1.0 − (ychk[i]*ychk[i])/(B1*B1));
2056
2057            x2 = −x1;
2058
2059
2060
2061            if (fabs(ellipse2tr(x1, ychk[i], AA, BB, CC, DD, EE, FF)) < EPS
                    /2.0)
2062
2063            {
2064
2065                nintpts++;
2066
2067                if (nintpts > 4)
2068
2069                {
2070
2071                    (*rtnCode) = ERROR_INTERSECTION_PTS;
2072
2073                    return −1.0;
2074
2075                }
2076
2077                xint[nintpts] = x1;
2078
2079                yint[nintpts] = ychk[i];
2080
2081            }
2082
2083
2084
2085            if ((fabs(ellipse2tr(x2, ychk[i], AA, BB, CC, DD, EE, FF)) < EPS
                    /2.0)
2086
2087                \&\& (fabs (x2 − x1) > EPS/2.0))
2088
2089            {
2090
2091                nintpts++;
2092
2093                if (nintpts > 4)
2094
2095                {
2096
2097                    (*rtnCode) = ERROR_INTERSECTION_PTS;
2098
2099                    return −1.0;
2100
2101                }
2102
2103                xint[nintpts] = x2;
2104
```

```
2105                yint[nintpts] = ychk[i];
2106
2107            }
2108
2109        }
2110
2111
2112
2113        //    ═══════════════════════════════════════════════════════════════

2114
2115        //== HANDLE  ALL  CASES  FOR  THE  NUMBER  OF  INTERSCTION  POINTS
                  ══════════════════
2116
2117        //    ═══════════════════════════════════════════════════════════════

2118
2119        switch (nintpts)
2120
2121        {
2122
2123            case  0:
2124
2125            case  1:
2126
2127                OverlapArea = nointpts (A1, B1, A2, B2, H1, K1, H2_TR, K2_TR,
                        AA,
2128
2129                                          BB, CC, DD, EE, FF, rtnCode);
2130
2131            return  OverlapArea;
2132
2133
2134
2135            case  2:
2136
2137                //-- when  there  are  two  intersection  points,  it  is  possible  for
2138
2139                //-- them  to  both  be  tangents,  in  which  case  one  of  the
                        ellipses
2140
2141                //-- is  fully  contained  within  the  other.   Check  the  points  for
2142
2143                //-- tangents;  if  one  of  the  points  is  a  tangent,  then  the
                        other
2144
2145                //-- must  be  as  well,  otherwise  there  would  be  more  than  2
2146
2147                //-- intersection  points.
2148
2149                fnRtnCode = istanpt (xint[1], yint[1], A1, B1, AA, BB, CC, DD,
2150
2151                                        EE, FF);
2152
2153
2154
2155                if (fnRtnCode == TANGENT_POINT)
2156
2157                    OverlapArea = nointpts (A1, B1, A2, B2, H1, K1, H2_TR,
                            K2_TR,
2158
2159                                              AA, BB, CC, DD, EE, FF, rtnCode);
2160
2161                else
2162
2163                    OverlapArea = twointpts (xint, yint, A1, B1, PHI_1, A2, B2,
2164
2165                                              H2_TR, K2_TR, PHI_2, AA, BB, CC,
                                                  DD,
2166
2167                                              EE, FF, rtnCode);
```

```
2168
2169                return OverlapArea;
2170
2171
2172
2173          case 3:
2174
2175                //-- when there are three intersection points, one and only one
2176
2177                //-- of the points must be a tangent point.
2178
2179                OverlapArea = threeintpts (xint, yint,  A1, B1, PHI_1, A2, B2,
2180
2181                                           H2_TR, K2_TR, PHI_2, AA, BB, CC, DD,
2182
2183                                           EE, FF, rtnCode);
2184
2185                return OverlapArea;
2186
2187
2188
2189          case 4:
2190
2191                //-- four intersections points has only one case.
2192
2193                OverlapArea = fourintpts (xint, yint,  A1, B1, PHI_1, A2, B2,
2194
2195                                          H2_TR, K2_TR, PHI_2, AA, BB, CC, DD,
2196
2197                                          EE, FF, rtnCode);
2198
2199                return OverlapArea;
2200
2201
2202
2203          default:
2204
2205                //-- should never get here (but get compiler warning for
                         missing
2206
2207                //-- return value if this line is omitted)
2208
2209                (*rtnCode) = ERROR_INTERSECTION_PTS;
2210
2211                return -1.0;
2212
2213      }
2214
2215 }
2216
2217
2218
2219 double ellipse2tr (double x, double y, double AA, double BB,
2220
2221                    double CC, double DD, double EE, double FF)
2222
2223 {
2224
2225      return (AA*x*x + BB*x*y + CC*y*y + DD*x + EE*y + FF);
2226
2227 }
2228
2229
2230
2231 double nointpts (double A1, double B1, double A2, double B2, double H1,
2232
2233                  double K1, double H2_TR, double K2_TR, double AA, double
                         BB,
2234
2235                  double CC, double DD, double EE, double FF, int *rtnCode)
2236
2237 {
2238
```

```
2239        //--- The relative size of the two ellipses can be found from the axis
2240
2241        //--- lengths
2242
2243        double relsize = (A1*B1) - (A2*B2);
2244
2245
2246
2247        if (relsize > 0.0)
2248
2249        {
2250
2251            //--- First Ellipse is larger than second ellipse.
2252
2253            //--- If second ellipse center (H2_TR, K2_TR) is inside
2254
2255            //--- first ellipse, then ellipse 2 is completely inside
2256
2257            //--- ellipse 1. Otherwise, the ellipses are disjoint.
2258
2259            if ( ((H2_TR*H2_TR) / (A1*A1)
2260
2261                + (K2_TR*K2_TR) / (B1*B1)) < 1.0 )
2262
2263            {
2264
2265                (*rtnCode) = ELLIPSE2_INSIDE_ELLIPSE1;
2266
2267                return (pi*A2*B2);
2268
2269            }
2270
2271            else
2272
2273            {
2274
2275                (*rtnCode) = DISJOINT_ELLIPSES;
2276
2277                return 0.0;
2278
2279            }
2280
2281        }
2282
2283        else if (relsize < 0.0)
2284
2285        {
2286
2287            //--- Second Ellipse is larger than first ellipse
2288
2289            //--- If first ellipse center (0, 0) is inside the
2290
2291            //--- second ellipse, then ellipse 1 is completely inside
2292
2293            //--- ellipse 2. Otherwise, the ellipses are disjoint
2294
2295            //---   AA*x^{}2 + BB*x*y + CC*y\^{}2 + DD*x + EE*y + FF = 0
2296
2297            if (FF < 0.0)
2298
2299            {
2300
2301                (*rtnCode) = ELLIPSE1_INSIDE_ELLIPSE2;
2302
2303                return (pi*A1*B1);
2304
2305            }
2306
2307            else
2308
2309            {
2310
2311                (*rtnCode) = DISJOINT_ELLIPSES;
```

```
2312
2313                    return 0.0;
2314
2315            }
2316
2317        }
2318
2319        else
2320
2321        {
2322
2323            //-- If execution arrives here, the relative sizes are identical.
2324
2325            //-- Are the ellipses the same?  Check the parameters to see.
2326
2327            if ((H1 == H2_TR) \&\& (K1 == K2_TR))
2328
2329            {
2330
2331                (*rtnCode) = ELLIPSES_ARE_IDENTICAL;
2332
2333                return (pi*A1*B1);
2334
2335            }
2336
2337            else
2338
2339            {
2340
2341                //-- should never get here, so return error
2342
2343                (*rtnCode) = ERROR_CALCULATIONS;
2344
2345                return -1.0;
2346
2347            }
2348
2349        }//-- end if (relsize > 0.0)
2350
2351 }
2352
2353
2354
2355 //-- two distinct intersection points (x1, y1) and (x2, y2) find overlap
          area
2356
2357 double twointpts (double x[], double y[], double A1, double B1, double
          PHI_1,
2358
2359                    double A2, double B2, double H2_TR, double K2_TR,
2360
2361                    double PHI_2, double AA, double BB, double CC, double DD,
2362
2363                    double EE, double FF, int *rtnCode)
2364
2365 {
2366
2367        double area1, area2;
2368
2369        double xmid, ymid, xmid_rt, ymid_rt;
2370
2371        double theta1, theta2;
2372
2373        double tmp, trsign;
2374
2375        double x1_tr, y1_tr, x2_tr, y2_tr;
2376
2377        double discr;
2378
2379        double cosphi, sinphi;
2380
2381
2382
```

```
2383        //--- if execution arrives here, the intersection points are not
2384
2385        //--- tangents.
2386
2387
2388
2389        //--- determine which direction to integrate in the ellipse_segment
2390
2391        //--- routine for each ellipse.
2392
2393
2394
2395        //--- find the parametric angles for each point on ellipse 1
2396
2397        if (fabs (x[1]) > A1)
2398
2399            x[1] = (x[1] < 0) ? -A1 : A1;
2400
2401        if (y[1] < 0.0)      //--- Quadrant III or IV
2402
2403            theta1 = twopi - acos (x[1] / A1);
2404
2405        else                 //--- Quadrant I or II
2406
2407            theta1 = acos (x[1] / A1);
2408
2409
2410
2411        if (fabs (x[2]) > A1)
2412
2413            x[2] = (x[2] < 0) ? -A1 : A1;
2414
2415        if (y[2] < 0.0)      //--- Quadrant III or IV
2416
2417            theta2 = twopi - acos (x[2] / A1);
2418
2419        else                 //--- Quadrant I or II
2420
2421            theta2 = acos (x[2] / A1);
2422
2423
2424
2425        //--- logic is for proceeding counterclockwise from theta1 to theta2
2426
2427        if (theta1 > theta2)
2428
2429        {
2430
2431            tmp = theta1;
2432
2433            theta1 = theta2;
2434
2435            theta2 = tmp;
2436
2437        }
2438
2439
2440
2441        //--- find a point on the first ellipse that is different than the two
2442
2443        //--- intersection points.
2444
2445        xmid = A1*cos ((theta1 + theta2)/2.0);
2446
2447        ymid = B1*sin ((theta1 + theta2)/2.0);
2448
2449
2450
2451        //--- the point (xmid, ymid) is on the first ellipse 'between' the two
2452
2453        //--- intersection points (x[1], y[1]) and (x[2], y[2]) when travelling
2454
```

```
2455        //--- counter- clockwise from (x[1], y[1]) to (x[2], y[2]).  If the
                    point
2456
2457        //--- (xmid, ymid) is inside the second ellipse, then the desired
                    segment
2458
2459        //--- of ellipse 1 contains the point (xmid, ymid), so integrate
2460
2461        //--- counterclockwise from (x[1], y[1]) to (x[2], y[2]).  Otherwise,
2462
2463        //--- integrate counterclockwise from (x[2], y[2]) to (x[1], y[1])
2464
2465        if (ellipse2tr (xmid, ymid, AA, BB, CC, DD, EE, FF) > 0.0)
2466
2467        {
2468
2469            tmp = theta1;
2470
2471            theta1 = theta2;
2472
2473            theta2 = tmp;
2474
2475        }
2476
2477
2478
2479        //--- here is the ellipse segment routine for the first ellipse
2480
2481        if (theta1 > theta2)
2482
2483            theta1 -= twopi;
2484
2485        if ((theta2 - theta1) > pi)
2486
2487            trsign = 1.0;
2488
2489        else
2490
2491            trsign = -1.0;
2492
2493        area1 = 0.5*(A1*B1*(theta2 - theta1)
2494
2495                + trsign*fabs (x[1]*y[2] - x[2]*y[1]));
2496
2497
2498
2499        //--- find ellipse 2 segment area.  The ellipse segment routine
2500
2501        //--- needs an ellipse that is centered at the origin and oriented
2502
2503        //--- with the coordinate axes.  The intersection points (x[1], y[1])
                    and
2504
2505        //--- (x[2], y[2]) are found with both ellipses translated and rotated
                    by
2506
2507        //--- (-H1, -K1) and -PHI_1.  Further translate and rotate the points
2508
2509        //--- to put the second ellipse at the origin and oriented with the
2510
2511        //--- coordinate axes.  The translation is (-H2_TR, -K2_TR), and the
2512
2513        //--- rotation is -(PHI_2 - PHI_1) = PHI_1 - PHI_2
2514
2515        cosphi = cos (PHI_1 - PHI_2);
2516
2517        sinphi = sin (PHI_1 - PHI_2);
2518
2519        x1_tr = (x[1] - H2_TR)*cosphi + (y[1] - K2_TR)*-sinphi;
2520
2521        y1_tr = (x[1] - H2_TR)*sinphi + (y[1] - K2_TR)*cosphi;
2522
2523        x2_tr = (x[2] - H2_TR)*cosphi + (y[2] - K2_TR)*-sinphi;
```

```
2524
2525        y2_tr = (x[2] - H2_TR)*sinphi + (y[2] - K2_TR)*cosphi;
2526

2527

2528
2529        //-- determine which branch of the ellipse to integrate by finding a
2530
2531        //-- point on the second ellipse, and asking whether it is inside the
2532
2533        //-- first ellipse (in their once-translated+rotated positions)
2534
2535        //-- find the parametric angles for each point on ellipse 1
2536
2537        if (fabs (x1_tr) > A2)
2538
2539            x1_tr = (x1_tr < 0) ? -A2 : A2;
2540
2541        if (y1_tr < 0.0)     //-- Quadrant III or IV
2542
2543            theta1 = twopi - acos (x1_tr/A2);
2544
2545        else                 //-- Quadrant I or II
2546
2547            theta1 = acos (x1_tr/A2);
2548

2549

2550
2551        if (fabs (x2_tr) > A2)
2552
2553            x2_tr = (x2_tr < 0) ? -A2 : A2;
2554
2555        if (y2_tr < 0.0)     //-- Quadrant III or IV
2556
2557            theta2 = twopi - acos (x2_tr/A2);
2558
2559        else                 //-- Quadrant I or II
2560
2561            theta2 = acos (x2_tr/A2);
2562

2563

2564
2565        //-- logic is for proceeding counterclockwise from theta1 to theta2
2566
2567        if (theta1 > theta2)
2568
2569        {
2570
2571            tmp = theta1;
2572
2573            theta1 = theta2;
2574
2575            theta2 = tmp;
2576
2577        }
2578

2579

2580
2581        //-- find a point on the second ellipse that is different than the two
2582
2583        //-- intersection points.
2584
2585        xmid = A2*cos ((theta1 + theta2)/2.0);
2586
2587        ymid = B2*sin ((theta1 + theta2)/2.0);
2588

2589

2590
2591        //-- translate the point back to the second ellipse in its once-
2592
2593        //-- translated+rotated position
2594
2595        cosphi = cos (PHI_2 - PHI_1);
2596
```

```
2597        sinphi = sin (PHI_2 − PHI_1);
2598
2599        xmid_rt = xmid∗cosphi + ymid∗−sinphi + H2_TR;
2600
2601        ymid_rt = xmid∗sinphi + ymid∗cosphi + K2_TR;
2602
2603
2604
2605        //−− the point (xmid_rt, ymid_rt) is on the second ellipse 'between'
                 the
2606
2607        //−− intersection points (x[1], y[1]) and (x[2], y[2]) when travelling
2608
2609        //−− counterclockwise from (x[1], y[1]) to (x[2], y[2]).  If the point
2610
2611        //−− (xmid_rt, ymid_rt) is inside the first ellipse, then the desired
2612
2613        //−− segment of ellipse 2 contains the point (xmid_rt, ymid_rt), so
2614
2615        //−− integrate counterclockwise from (x[1], y[1]) to (x[2], y[2]).
2616
2617        //−− Otherwise, integrate counterclockwise from (x[2], y[2]) to
2618
2619        //−− (x[1], y[1])
2620
2621        if (((xmid_rt∗xmid_rt)/(A1∗A1) + (ymid_rt∗ymid_rt)/(B1∗B1)) > 1.0)
2622
2623        {
2624
2625            tmp = theta1;
2626
2627            theta1 = theta2;
2628
2629            theta2 = tmp;
2630
2631        }
2632
2633
2634
2635        //−− here is the ellipse segment routine for the second ellipse
2636
2637        if (theta1 > theta2)
2638
2639            theta1 −= twopi;
2640
2641        if ((theta2 − theta1) > pi)
2642
2643            trsign = 1.0;
2644
2645        else
2646
2647            trsign = −1.0;
2648
2649        area2 = 0.5∗(A2∗B2∗(theta2 − theta1)
2650
2651                + trsign∗fabs (x1_tr∗y2_tr − x2_tr∗y1_tr));
2652
2653
2654
2655        (∗rtnCode) = TWO_INTERSECTION_POINTS;
2656
2657        return area1 + area2;
2658
2659 }
2660
2661
2662
2663 //−− three distinct intersection points, must have two intersections
2664
2665 //−− and one tangent, which is the only possibility
2666
2667 double threeintpts (double xint[], double yint[], double A1, double B1,
2668
```

```
2669                        double PHI_1, double A2, double B2, double H2_TR,
2670
2671                        double K2_TR, double PHI_2, double AA, double BB,
2672
2673                        double CC, double DD, double EE, double FF,
2674
2675                        int *rtnCode)
2676
2677  {
2678
2679      int i, tanpts, tanindex, fnRtn;
2680
2681      double OverlapArea;
2682
2683
2684
2685      //-- need to determine which point is a tangent, and which two points
2686
2687      //-- are intersections
2688
2689      tanpts = 0;
2690
2691      for (i = 1; i <= 3; i++)
2692
2693      {
2694
2695          fnRtn = istanpt (xint[i], yint[i], A1, B1, AA, BB, CC, DD, EE, FF);
2696
2697
2698
2699          if (fnRtn == TANGENT_POINT)
2700
2701          {
2702
2703              tanpts++;
2704
2705              tanindex = i;
2706
2707          }
2708
2709      }
2710
2711
2712
2713      //-- there MUST be 2 intersection points and only one tangent
2714
2715      if (tanpts != 1)
2716
2717      {
2718
2719          //-- should never get here unless there is a problem discerning
2720
2721          //-- whether or not a point is a tangent or intersection
2722
2723          (*rtnCode) = ERROR_INTERSECTION_PTS;
2724
2725          return -1.0;
2726
2727      }
2728
2729
2730
2731      //-- store the two interesection points into (x[1], y[1]) and
2732
2733      //-- (x[2], y[2])
2734
2735      switch (tanindex)
2736
2737      {
2738
2739          case 1:
2740
2741              xint[1] = xint[3];
```

```
2742
2743                yint[1] = yint[3];
2744
2745                break;
2746
2747
2748
2749          case 2:
2750
2751                xint[2] = xint[3];
2752
2753                yint[2] = yint[3];
2754
2755                break;
2756
2757
2758
2759          case 3:
2760
2761                //-- intersection points are already in the right places
2762
2763                break;
2764
2765      }
2766
2767
2768
2769      OverlapArea = twointpts (xint, yint, A1, B1, PHI_1, A2, B2, H2_TR,
               K2_TR,
2770
2771                                  PHI_2, AA, BB, CC, DD, EE, FF, rtnCode);
2772
2773      (*rtnCode) = THREE_INTERSECTION_POINTS;
2774
2775      return OverlapArea;
2776
2777 }
2778
2779
2780
2781 //-- four intersection points
2782
2783 double fourintpts (double xint[], double yint[], double A1, double B1,
2784
2785                     double PHI_1, double A2, double B2, double H2_TR,
2786
2787                     double K2_TR, double PHI_2, double AA, double BB,
2788
2789                     double CC, double DD, double EE, double FF, int *rtnCode
                           )
2790
2791 {
2792
2793      int i, j, k;
2794
2795      double xmid, ymid, xint_tr[5], yint_tr[5], OverlapArea;
2796
2797      double theta[5], theta_tr[5], cosphi, sinphi, tmp0, tmp1, tmp2;
2798
2799      double area1, area2, area3, area4, area5;
2800
2801
2802
2803      //-- only one case, which involves two segments from each ellipse, plus
2804
2805      //-- two triangles.
2806
2807      //-- get the parametric angles along the first ellipse for each of the
2808
2809      //-- intersection points
2810
2811      for (i = 1; i <= 4; i++)
2812
```

```
2813        {
2814
2815            if (fabs (xint[i]) > A1)
2816
2817                xint[i] = (xint[i] < 0) ? -A1 : A1;
2818
2819            if (yint[i] < 0.0)  //-- Quadrant III or IV
2820
2821                theta[i] = twopi - acos (xint[i] / A1);
2822
2823            else                //-- Quadrant I or II
2824
2825                theta[i] = acos (xint[i] / A1);
2826
2827        }
2828
2829
2830
2831        //-- sort the angles by straight insertion, and put the points in
2832
2833        //-- counter-clockwise order
2834
2835        for (j = 2; j <= 4; j++)
2836
2837        {
2838
2839            tmp0 = theta[j];
2840
2841            tmp1 = xint[j];
2842
2843            tmp2 = yint[j];
2844
2845
2846
2847            for (k = j - 1; k >= 1; k--)
2848
2849            {
2850
2851                if (theta[k] <= tmp0)
2852
2853                    break;
2854
2855
2856
2857                theta[k+1] = theta[k];
2858
2859                xint[k+1] = xint[k];
2860
2861                yint[k+1] = yint[k];
2862
2863            }
2864
2865
2866
2867            theta[k+1] = tmp0;
2868
2869            xint[k+1] = tmp1;
2870
2871            yint[k+1] = tmp2;
2872
2873        }
2874
2875
2876
2877        //-- find the area of the interior quadrilateral
2878
2879        area1 = 0.5*fabs ((xint[3] - xint[1])*(yint[4] - yint[2])
2880
2881                         - (xint[4] - xint[2])*(yint[3] - yint[1]));
2882
2883
2884
2885        //-- the intersection points lie on the second ellipse in its once
```

```
2886
2887        //−− translated+rotated position.  The segment algorithm is implemented
2888
2889        //−− for an ellipse that is centered at the origin, and oriented with
2890
2891        //−− the coordinate axes; so, in order to use the segment algorithm
2892
2893        //−− with the second ellipse, the intersection points must be further
2894
2895        //−− translated+rotated by amounts that put the second ellipse centered
2896
2897        //−− at the origin and oriented with the coordinate axes.
2898
2899        cosphi = cos (PHI_1 − PHI_2);
2900
2901        sinphi = sin (PHI_1 − PHI_2);
2902
2903        for (i = 1; i <= 4; i++)
2904
2905        {
2906
2907            xint_tr[i] = (xint[i] − H2_TR)*cosphi + (yint[i] − K2_TR)*−sinphi;
2908
2909            yint_tr[i] = (xint[i] − H2_TR)*sinphi + (yint[i] − K2_TR)*cosphi;
2910
2911
2912
2913            if (fabs (xint_tr[i]) > A2)
2914
2915                xint_tr[i] = (xint_tr[i] < 0) ? −A2 : A2;
2916
2917            if (yint_tr[i] < 0.0)    //−− Quadrant III or IV
2918
2919                theta_tr[i] = twopi − acos (xint_tr[i]/A2);
2920
2921            else                //−− Quadrant I or II
2922
2923                theta_tr[i] = acos (xint_tr[i]/A2);
2924
2925        }
2926
2927
2928
2929        //−− get the area of the two segments on ellipse 1
2930
2931        xmid = A1*cos ((theta[1] + theta[2])/2.0);
2932
2933        ymid = B1*sin ((theta[1] + theta[2])/2.0);
2934
2935
2936
2937        //−− the point (xmid, ymid) is on the first ellipse 'between' the two
2938
2939        //−− sorted intersection points (xint[1], yint[1]) and (xint[2], yint
2940             [2])
2941        //−− when travelling counter− clockwise from (xint[1], yint[1]) to
2942
2943        //−− (xint[2], yint[2]).  If the point (xmid, ymid) is inside the
2944             second
2945        //−− ellipse, then one desired segment of ellipse 1 contains the point
2946
2947        //−− (xmid, ymid), so integrate counterclockwise from (xint[1], yint
2948             [1])
2949        //−− to (xint[2], yint[2]) for the first segment, and from
2950
2951        //−− (xint[3], yint[3] to (xint[4], yint[4]) for the second segment.
2952
2953        if (ellipse2tr (xmid, ymid, AA, BB, CC, DD, EE, FF) < 0.0)
2954
2955        {
```

```
2956
2957            area2 = 0.5*(A1*B1*(theta[2] - theta[1])
2958
2959                    - fabs (xint[1]*yint[2] - xint[2]*yint[1]));
2960
2961            area3 = 0.5*(A1*B1*(theta[4] - theta[3])
2962
2963                    - fabs (xint[3]*yint[4] - xint[4]*yint[3]));
2964
2965            area4 = 0.5*(A2*B2*(theta_tr[3] - theta_tr[2])
2966
2967                    - fabs (xint_tr[2]*yint_tr[3] - xint_tr[3]*yint_tr[2]));
2968
2969            area5 = 0.5*(A2*B2*(theta_tr[1] - (theta_tr[4] - twopi))
2970
2971                    - fabs (xint_tr[4]*yint_tr[1] - xint_tr[1]*yint_tr[4]));
2972
2973        }
2974
2975        else
2976
2977        {
2978
2979            area2 = 0.5*(A1*B1*(theta[3] - theta[2])
2980
2981                    - fabs (xint[2]*yint[3] - xint[3]*yint[2]));
2982
2983            area3 = 0.5*(A1*B1*(theta[1] - (theta[4] - twopi))
2984
2985                    - fabs (xint[4]*yint[1] - xint[1]*yint[4]));
2986
2987            area4 = 0.5*(A2*B2*(theta[2] - theta[1])
2988
2989                    - fabs (xint_tr[1]*yint_tr[2] - xint_tr[2]*yint_tr[1]));
2990
2991            area5 = 0.5*(A2*B2*(theta[4] - theta[3])
2992
2993                    - fabs (xint_tr[3]*yint_tr[4] - xint_tr[4]*yint_tr[3]));
2994
2995        }
2996
2997
2998
2999        OverlapArea = area1 + area2 + area3 + area4 + area5;
3000
3001        (*rtnCode) = FOUR_INTERSECTION_POINTS;
3002
3003        return OverlapArea;
3004
3005 }
3006
3007
3008
3009 //-- check whether an intersection point is a tangent or a cross-point
3010
3011 int istanpt (double x, double y, double A1, double B1, double AA, double BB
        ,
3012
3013              double CC, double DD, double EE, double FF)
3014
3015 {
3016
3017     double x1, y1, x2, y2, theta, test1, test2, branch, eps_radian;
3018
3019
3020
3021     //-- Avoid inverse trig calculation errors: there could be an error
3022
3023     //-- if \textbar x1/A\textbar  > 1.0 when calling acos(). If execution
            arrives here,
3024
3025     //-- then the point is on the ellipse within EPS.
3026
```

```
3027        if (fabs (x) > A1)
3028
3029            x = (x < 0) ? -A1 : A1;
3030
3031
3032
3033        //-- Calculate the parametric angle on the ellipse for (x, y)
3034
3035        //-- The parametric angles depend on the quadrant where each point
3036
3037        //-- is located.  See Table 1 in the reference.
3038
3039        if (y < 0.0)      //-- Quadrant III or IV
3040
3041            theta = twopi - acos (x / A1);
3042
3043        else                    //-- Quadrant I or II
3044
3045            theta = acos (x / A1);
3046
3047
3048
3049        //-- determine the distance from the origin to the point (x, y)
3050
3051        branch = sqrt (x*x + y*y);
3052
3053
3054
3055        //-- use the distance to find a small angle, such that the distance
3056
3057        //-- along ellipse 1 is approximately 2*EPS
3058
3059        if (branch < 100.0*EPS)
3060
3061            eps_radian = 2.0*EPS;
3062
3063        else
3064
3065            eps_radian = asin (2.0*EPS/branch);
3066
3067
3068
3069        //-- determine two points that are on each side of (x, y) and lie on
3070
3071        //-- the first ellipse
3072
3073        x1 = A1*cos (theta + eps_radian);
3074
3075        y1 = B1*sin (theta + eps_radian);
3076
3077        x2 = A1*cos (theta - eps_radian);
3078
3079        y2 = B1*sin (theta - eps_radian);
3080
3081
3082
3083        //-- evaluate the two adjacent points in the second ellipse equation
3084
3085        test1 = ellipse2tr (x1, y1, AA, BB, CC, DD, EE, FF);
3086
3087        test2 = ellipse2tr (x2, y2, AA, BB, CC, DD, EE, FF);
3088
3089
3090
3091        //-- if the ellipses are tangent at the intersection point, then
3092
3093        //-- points on both sides will either both be inside ellipse 1, or
3094
3095        //-- they will both be outside ellipse 1
3096
3097        if ((test1*test2) > 0.0)
3098
3099            return TANGENT_POINT;
```

```
3100
3101        else
3102
3103            return INTERSECTION_POINT;
3104
3105 }
3106
3107
3108
3109 //

3110
3111 //—— CACM Algorithm 326: Roots of low order polynomials.
3112
3113 //—— Nonweiler, Terence R.F., CACM Algorithm 326: Roots of low order
3114
3115 //—— polynomials, Communications of the ACM, vol. 11 no. 4, pages
3116
3117 //—— 269−270 (1968). Translated into c and programmed by M. Dow, ANUSF,
3118
3119 //—— Australian National University, Canberra, Australia.
3120
3121 //—— Accessed at http://www.netlib.org/toms/326.
3122
3123 //—— Modified to void functions, integers replaced with floating point
3124
3125 //—— where appropriate, some other slight modifications for readability
3126
3127 //—— and debugging ease.
3128
3129 //

3130
3131 void QUADROOTS (double p[], double r[][5])
3132
3133 {
3134
3135        /*
3136
3137        Array r[3][5]   p[5]
3138
3139        Roots of poly p[0]*x^{}2 + p[1]*x + p[2]=0
3140
3141        x=r[1][k] + i r[2][k]   k=1,2
3142
3143        */
3144
3145        double b,c,d;
3146
3147        b=−p[1]/(2.0*p[0]);
3148
3149        c=p[2]/p[0];
3150
3151        d=b*b−c;
3152
3153        if(d>=0.0)
3154
3155        {
3156
3157            if(b>0.0)
3158
3159                b=(r[1][2]=(sqrt(d)+b));
3160
3161            else
3162
3163                b=(r[1][2]=(−sqrt(d)+b));
3164
3165            r[1][1]=c/b;
3166
3167            r[2][1]=(r[2][2]=0.0);
3168
```

```
3169        }
3170
3171        else
3172
3173        {
3174
3175            d=(r[2][1]=sqrt(-d));
3176
3177            r[2][2]=-d;
3178
3179            r[1][1]=(r[1][2]=b);
3180
3181        }
3182
3183        return;
3184
3185 }
3186
3187
3188
3189 void CUBICROOTS(double p[], double r[][5])
3190
3191 {
3192
3193        /*
3194
3195        Array r[3][5]   p[5]
3196
3197        Roots of poly p[0]*x\^{}3 + p[1]*x\^{}2 + p[2]*x + p[3] = 0
3198
3199        x=r[1][k] + i r[2][k]   k=1,...,3
3200
3201        Assumes 0<arctan(x)<pi/2 for x>0
3202
3203        */
3204
3205        double s,t,b,c,d;
3206
3207        int k;
3208
3209        if(p[0]!=1.0)
3210
3211        {
3212
3213            for(k=1;k<4;k++)
3214
3215                p[k]=p[k]/p[0];
3216
3217            p[0]=1.0;
3218
3219        }
3220
3221        s=p[1]/3.0;
3222
3223        t=s*p[1];
3224
3225        b=0.5*(s*(t/1.5-p[2])+p[3]);
3226
3227        t=(t-p[2])/3.0;
3228
3229        c=t*t*t;
3230
3231        d=b*b-c;
3232
3233        if(d>=0.0)
3234
3235        {
3236
3237            d=pow((sqrt(d)+fabs(b)),1.0/3.0);
3238
3239            if(d!=0.0)
3240
3241            {
```

```
3242
3243              if (b>0.0)
3244
3245                   b=-d;
3246
3247            else
3248
3249                   b=d;
3250
3251            c=t/b;
3252
3253         }
3254
3255         d=r[2][2]=sqrt\eqref{GrindEQ__0_75_}*(b-c);
3256
3257         b=b+c;
3258
3259         c=r[1][2]=-0.5*b-s;
3260
3261         if((b>0.0 \&\& s<=0.0) \textbar \textbar (b<0.0 \&\& s>0.0))
3262
3263         {
3264
3265             r[1][1]=c;
3266
3267             r[2][1]=-d;
3268
3269             r[1][3]=b-s;
3270
3271             r[2][3]=0.0;
3272
3273         }
3274
3275         else
3276
3277         {
3278
3279             r[1][1]=b-s;
3280
3281             r[2][1]=0.0;
3282
3283             r[1][3]=c;
3284

3285             r[2][3]=-d;
3286
3287         }
3288
3289      }   /* end 2 equal or complex roots */
3290
3291      else
3292
3293      {
3294
3295          if(b==0.0)
3296
3297              d=atan\eqref{GrindEQ__1_0_}/1.5;
3298
3299          else
3300
3301              d=atan(sqrt(-d)/fabs(b))/3.0;
3302
3303          if(b<0.0)
3304
3305              b=2.0*sqrt(t);
3306
3307          else
3308
3309              b=-2.0*sqrt(t);
3310
3311          c=cos(d)*b;
3312
3313          t=-sqrt\eqref{GrindEQ__0_75_}*sin(d)*b-0.5*c;
3314
```

```
3315              d=−t−c−s ;
3316
3317              c=c−s ;
3318
3319              t=t−s ;
3320
3321              if ( fabs ( c)>fabs ( t ) )
3322
3323              {
3324
3325                   r [1][3]= c ;
3326
3327              }
3328
3329              else
3330
3331              {
3332
3333                   r [1][3]= t ;
3334
3335                   t=c ;
3336
3337              }
3338
3339              if ( fabs ( d)>fabs ( t ) )
3340
3341              {
3342
3343                   r [1][2]= d ;
3344
3345              }
3346
3347              else
3348
3349              {
3350
3351                   r [1][2]= t ;
3352
3353                   t=d ;
3354
3355              }
3356
3357              r [1][1]= t ;
3358
3359              for (k=1;k< 4;k++)
3360
3361                   r [2][k]=0.0;
3362
3363         }
3364
3365      return ;
3366
3367 }
3368
3369
3370
3371 void BIQUADROOTS( double p [ ] , double r [ ] [ 5 ] )
3372
3373 {
3374
3375      /*
3376
3377      Array r [3][5]  p [5]
3378
3379      Roots of poly p[0]* x\^{}4 + p[1]* x\^{}3 + p[2]* x\^{}2 + p[3]* x + p[4] =
                0
3380
3381      x=r [1][k] + i r [2][k]  k=1,...,4
3382
3383      */
3384
3385      double a , b , c , d , e ;
3386
```

```
3387        int k,j;

3389        if(p[0] != 1.0)

3391        {

3393            for(k=1;k<5;k++)

3395                    p[k]=p[k]/p[0];

3397            p[0]=1.0;

3399        }

3401        e=0.25*p[1];

3403        b=2.0*e;

3405        c=b*b;

3407        d=0.75*c;

3409        b=p[3]+b*(c-p[2]);

3411        a=p[2]-d;

3413        c=p[4]+e*(e*a-p[3]);

3415        a=a-d;

3417        p[1]=0.5*a;

3419        p[2]=(p[1]*p[1]-c)*0.25;

3421        p[3]=b*b/(-64.0);

3423        if(p[3]<0.0)

3425        {

3427            CUBICROOTS(p,r);

3429            for(k=1;k<4;k++)

3431            {

3433                if(r[2][k]==0.0 \&\& r[1][k]>0.0)

3435                {

3437                    d=r[1][k]*4.0;

3439                    a=a+d;

3441                    if(a>=0.0 \&\& b>=0.0)

3443                        p[1]=sqrt(d);

3445                    else if(a<=0.0 \&\& b<=0.0)

3447                        p[1]=sqrt(d);

3449                    else

3451                        p[1]=-sqrt(d);

3453                    b=0.5*(a+b/p[1]);

3455                    goto QUAD;

3457                }

3459            }
```

```
3460
3461        }
3462
3463        if ( p [ 2 ] < 0 . 0 )
3464
3465        {
3466
3467            b=sqrt ( c ) ;
3468
3469            d=b+b−a ;
3470
3471            p [ 1 ]= 0 . 0 ;
3472
3473            if ( d > 0 . 0 )
3474
3475                p [ 1 ]= sqrt ( d ) ;
3476
3477        }
3478
3479        else
3480
3481        {
3482
3483            if ( p [ 1 ] > 0 . 0 )
3484
3485                b=sqrt ( p [ 2 ] ) ∗ 2 . 0 + p [ 1 ] ;
3486
3487            else
3488
3489                b=−sqrt ( p [ 2 ] ) ∗ 2 . 0 + p [ 1 ] ;
3490
3491            if ( b != 0 . 0 )
3492
3493            {
3494
3495                p [ 1 ]= 0 . 0 ;
3496
3497            }
3498
3499            else
3500
3501            {
3502
3503                for ( k=1;k < 5 ; k++)
3504
3505                {
3506
3507                    r [ 1 ] [ k]=−e ;
3508
3509                    r [ 2 ] [ k ]= 0 . 0 ;
3510
3511                }
3512
3513                goto  END;
3514
3515            }
3516
3517        }
3518
3519 QUAD:
3520
3521        p [ 2 ]= c / b ;
3522
3523        QUADROOTS( p , r ) ;
3524
3525        for ( k=1;k < 3 ; k++)
3526
3527            for ( j =1; j < 3 ; j ++)
3528
3529                r [ j ] [ k+2]=r [ j ] [ k ] ;
3530
3531        p [ 1 ]=−p [ 1 ] ;
3532
```

```
3533        p[2]=b;
3534
3535        QUADROOTS(p,r);
3536
3537        for(k=1;k<5;k++)
3538
3539            r[1][k]=r[1][k]-e;
3540
3541 END:
3542
3543        return;
3544
3545 }
```

LISTING 15. C-SOURCE CODE FOR UTILITY FUNCTIONS

7. APPENDIX D.

```
3547 program\_constants.h:
3548
3549
3550
3551 //────────────────────────────────────────────────
3552
3553 //== INCLUDE ANSI C SYSTEM HEADER FILES
3554         ───────────────────────────────
3555 //────────────────────────────────────────────────
3556
3557 #include <math.h> //-- for calls to trig, sqrt and power functions
3558
3559
3560
3561 //────────────────────────────────────────────────
3562
3563 //== DEFINE PROGRAM CONSTANTS
3564         ───────────────────────────────────
3565 //────────────────────────────────────────────────
3566
3567 #define NORMAL_TERMINATION                     0
3568
3569 #define NO_INTERSECTION_POINTS                 100
3570
3571 #define ONE_INTERSECTION_POINT                 101
3572
3573 #define LINE_TANGENT_TO_ELLIPSE                102
3574
3575 #define DISJOINT_ELLIPSES                      103
3576
3577 #define ELLIPSE2_OUTSIDETANGENT_ELLIPSE1       104
3578
3579 #define ELLIPSE2_INSIDETANGENT_ELLIPSE1        105
3580
3581 #define ELLIPSES_INTERSECT                     106
3582
3583 #define TWO_INTERSECTION_POINTS                107
3584
3585 #define THREE_INTERSECTION_POINTS              108
3586
3587 #define FOUR_INTERSECTION_POINTS               109
3588
3589 #define ELLIPSE1_INSIDE_ELLIPSE2               110
3590
3591 #define ELLIPSE2_INSIDE_ELLIPSE1               111
```

```
3592
3593 #define ELLIPSES_ARE_IDENTICAL                112
3594
3595 #define INTERSECTION_POINT                    113
3596
3597 #define TANGENT_POINT                         114
3598
3599
3600
3601 #define ERROR_ELLIPSE_PARAMETERS             -100
3602
3603 #define ERROR_DEGENERATE_ELLIPSE             -101
3604
3605 #define ERROR_POINTS_NOT_ON_ELLIPSE          -102
3606
3607 #define ERROR_INVERSE_TRIG                   -103
3608
3609 #define ERROR_LINE_POINTS                    -104
3610
3611 #define ERROR_QUARTIC_CASE                   -105
3612
3613 #define ERROR_POLYNOMIAL_DEGREE              -107
3614
3615 #define ERROR_POLYNOMIAL_ROOTS               -108
3616
3617 #define ERROR_INTERSECTION_PTS               -109
3618
3619 #define ERROR_CALCULATIONS                   -112
3620
3621
3622
3623 #define EPS                                +1.0E-07
3624
3625 #define pi      (2.0*asin (1.0))  //-- a maximum-precision value of pi
3626
3627 #define twopi   (2.0*pi)             //-- a maximum-precision value of 2*pi
3628
3629
3630
3631
3632
3633
3634
3635 call_es.c:
3636
3637
3638
3639 #include <stdio.h>
3640
3641 #include <math.h>
3642
3643 #include "program_constants.h"
3644
3645 double ellipse_segment (double A, double B, double X1, double Y1, double X2
         ,
3646
3647                         double Y2, int *MessageCode);
3648
3649
3650
3651 int main (int argc, char ** argv)
3652
3653 {
3654
3655     double A, B;
3656
3657     double X1, Y1;
3658
3659     double X2, Y2;
3660
3661     double area1, area2;
3662
```

```
3663      double pi = 2.0 * asin eqref{GrindEQ__1_0_};      //--- a maximum-precision
                value of pi
3664
3665      int rtn;
3666
3667      char msg[1024];
3668
3669      printf ("Calling ellipse_segment.ctextbackslash n");
3670
3671
3672
3673      //--- case shown in Fig. 1
3674
3675      A = 4.;
3676
3677      B = 2.;
3678
3679      X1 = 4./sqrt (5.);
3680
3681      Y1 = 4./sqrt (5.);
3682
3683      X2 = -3.;
3684
3685      Y2 = -sqrt (7.)/2.;
3686
3687
3688
3689      area1 = ellipse_segment (A, B, X1, Y1, X2, Y2, &rtn);
3690
3691      sprintf (msg,"Fig 1: segment area = %15.8f, return_value = %d\
                textbackslash n", area1, rtn);
3692
3693      printf (msg);
3694
3695
3696
3697      //--- case shown in Fig. 2
3698
3699      A = 4.;
3700
3701      B = 2.;
3702
3703      X1 = -3.;
3704
3705      Y1 = -sqrt (7.)/2.;
3706
3707      X2 = 4./sqrt (5.);
3708
3709      Y2 = 4./sqrt (5.);
3710
3711
3712
3713      area2 = ellipse_segment (A, B, X1, Y1, X2, Y2, &rtn);
3714
3715      sprintf (msg,"Fig 2: segment area = %15.8f, return_value = %
                dtextbackslash n", area2, rtn);
3716
3717      printf (msg);
3718
3719
3720
3721      sprintf (msg,"sum of ellipse segments = %15.8ftextbackslash n", area1 +
                area2);
3722
3723      printf (msg);
3724
3725      sprintf (msg,"total ellipse area by pi*a*b = %15.8ftextbackslash n", pi*
                A*B);
3726
3727      printf (msg);
3728
3729
3730
```

```
3731      return rtn;
3732
3733  }
3734
3735
3736
3737
3738
3739  call_el.c:
3740
3741
3742
3743  #include <stdio.h>
3744
3745  #include <math.h>
3746
3747  #include "program_constants.h"
3748
3749  double \textbf{ellipse_segment} (double A, double B, double X1, double Y1,
          double X2,
3750
3751                                  double Y2, int *MessageCode);
3752
3753
3754
3755  double \textbf{ellipse_line_overlap} (double PHI, double A, double B,
          double H,
3756
3757                                        double K, double X1, double Y1, double X2,
3758
3759                                        double Y2, int *MessageCode);
3760
3761
3762
3763  int \textbf{main} (int argc, char ** argv)
3764
3765  {
3766
3767      double A, B;
3768
3769      double H, K, PHI;
3770
3771      double X1, Y1;
3772
3773      double X2, Y2;
3774
3775      double area1, area2;
3776
3777      double pi = 2.0 * \textbf{asin} \eqref{GrindEQ__1_0_};     //--- a maximum
              -precision value of pi
3778
3779      int rtn;
3780
3781      char msg[1024];
3782
3783      \textbf{printf} ("Calling ellipse_line_overlap.c\textbackslash n");
3784
3785
3786
3787      //--- case shown in Fig. 4
3788
3789      A = 4.;
3790
3791      B = 2.;
3792
3793      H = -6;
3794
3795      K = 3;
3796
3797      PHI = 3.*pi/8.0;
3798
3799      X1 = -3.;
3800
```

```
3801     Y1 = 3.;
3802
3803     X2 = −7.;
3804
3805     Y2 = 7.;
3806
3807
3808
3809     area1 = \textbf{ellipse\_line\_overlap} (PHI, A, B, H, K, X1, Y1, X2, Y2
             , \&rtn);
3810
3811     \textbf{sprintf} (msg,"Fig 4: area = \%15.8f, return_value = \%d\
             textbackslash n", area1, rtn);
3812
3813     \textbf{printf} (msg);
3814
3815
3816
3817     //—— case shown in Fig. 4, points reversed
3818
3819     A = 4.;
3820
3821     B = 2.;
3822
3823     H = −6;
3824
3825     K = 3;
3826
3827     PHI = 3.*pi/8.0;
3828
3829     X1 = −7.;
3830
3831     Y1 = 7.;
3832
3833     X2 = −3.;
3834
3835     Y2 = 3.;
3836
3837
3838
3839     area2 = \textbf{ellipse\_line\_overlap} (PHI, A, B, H, K, X1, Y1, X2, Y2
             , \&rtn);
3840
3841     \textbf{sprintf} (msg,"Fig 4 reverse: area = %15.8f, return_value = \%d\
             textbackslash n", area2, rtn);
3842
3843     \textbf{printf} (msg);
3844
3845
3846
3847     \textbf{sprintf} (msg,"sum of ellipse segments = %15.8ftextbackslash n",
             area1 + area2);
3848
3849     \textbf{printf} (msg);
3850
3851     \textbf{sprintf} (msg,"total ellipse area by pi*a*b = %15.8
             ftextbackslash n", pi*A*B);
3852
3853     \textbf{printf} (msg);
3854
3855
3856
3857     return rtn;
3858
3859   }
3860
3861
3862
3863
3864
3865   call_ee.c:
3866
3867
```

```c
3868
3869   #include <stdio.h>
3870
3871   #include "program_constants.h"
3872
3873   double ellipse_ellipse_overlap (double PHI_1, double A1, double B1,
3874
3875                                   double H1, double K1, double PHI_2,
3876
3877                                   double A2, double B2, double H2, double K2
                                       ,
3878
3879                                   int *rtnCode);
3880
3881
3882
3883   int main (int argc, char ** argv)
3884
3885   {
3886
3887       double A1, B1, H1, K1, PHI_1;
3888
3889       double A2, B2, H2, K2, PHI_2;
3890
3891       double area;
3892
3893       int rtn;
3894
3895       char msg[1024];
3896
3897       printf ("Calling ellipse_ellipse_overlap.c\textbackslash n\
                  textbackslash n");
3898
3899
3900
3901       //-- case 0-1
3902
3903       A1 = 3.; B1 = 2.; H1 = 0.; K1 = 0.; PHI_1 = 0.;
3904
3905       A2 = 2.; B2 = 1.; H2 = -.75; K2 = 0.25; PHI_2 = pi/4.;
3906
3907       area = ellipse_ellipse_overlap (PHI_1, A1, B1, H1, K1,
3908
3909                                       PHI_2, A2, B2, H2, K2, \&rtn);
3910
3911       sprintf (msg,"Case 0-1: area = \%15.8f, return_value = \%d\
                  textbackslash n", area, rtn);
3912
3913       printf (msg);
3914
3915       sprintf (msg,"          ellipse 2 area by pi*a2*b2 = \%15.8f\
                  textbackslash n", pi*A2*B2);
3916
3917       printf (msg);
3918
3919
3920
3921       //-- case 0-2
3922
3923       A1 = 2.; B1 = 1.; H1 = 0.; K1 = 0.; PHI_1 = 0.;
3924
3925       A2 = 3.; B2 = 2.; H2 = -.3; K2 = -.25; PHI_2 = pi/4.;
3926
3927       area = ellipse_ellipse_overlap (PHI_1, A1, B1, H1, K1,
3928
3929                                       PHI_2, A2, B2, H2, K2, &rtn);
3930
3931       sprintf (msg,"Case 0-2: area = %15.8f, return\_value = \%d\
                  textbackslash n", area, rtn);
3932
3933       printf (msg);
3934
```

```
3935      sprintf (msg,"              ellipse 1 area by pi*a1*b1 = \%15.8f\
              textbackslash n", pi*A1*B1);
3936
3937      printf (msg);
3938
3939
3940
3941      //-- case 0-3
3942
3943      A1 = 2.; B1 = 1.; H1 = 0.; K1 = 0.; PHI_1 = 0.;
3944
3945      A2 = 1.5; B2 = 0.75; H2 = -2.5; K2 = 1.5; PHI_2 = pi/4.;
3946
3947      area = ellipse_ellipse_overlap (PHI_1, A1, B1, H1, K1,
3948
3949                                      PHI_2, A2, B2, H2, K2, &rtn);
3950
3951      sprintf (msg,"Case 0-3: area = \%15.8f, return_value = \%d\
              textbackslash n", area, rtn);
3952
3953      printf (msg);
3954
3955      printf ("            Ellipses are disjoint, ovelap area = 0.0\
              textbackslash n\textbackslash n");
3956
3957
3958
3959      //-- case 1-1
3960
3961      A1 = 3.; B1 = 2.; H1 = 0.; K1 = 0.; PHI\_1 = 0.;
3962
3963      A2 = 2.; B2 = 1.; H2 = -1.0245209260022; K2 = 0.25; PHI_2 = pi/4.;
3964
3965      area = ellipse_ellipse_overlap (PHI_1, A1, B1, H1, K1,
3966
3967                                      PHI_2, A2, B2, H2, K2, \&rtn);
3968
3969      sprintf (msg,"Case 1-1: area = \%15.8f, return\_value = \%d\
              textbackslash n", area, rtn);
3970
3971      printf (msg);
3972
3973      sprintf (msg,"            ellipse 2 area by pi*a2*b2 = \%15.8f\
              textbackslash n", pi*A2*B2);
3974
3975      printf (msg);
3976
3977
3978
3979      //-- case 1-2
3980
3981      A1 = 2.; B1 = 1.; H1 = 0.; K1 = 0.; PHI_1 = 0.;
3982
3983      A2 = 3.5; B2 = 1.8; H2 = .22; K2 = .1; PHI_2 = pi/4.;
3984
3985      area = ellipse_ellipse_overlap (PHI_1, A1, B1, H1, K1,
3986
3987                                      PHI_2, A2, B2, H2, K2, \&rtn);
3988
3989      sprintf (msg,"Case 1-2: area = \%15.8f, return_value = \%d\
              textbackslash n", area, rtn);
3990
3991      printf (msg);
3992
3993      sprintf (msg,"            ellipse 1 area by pi*a1b1 = \%15.8f\
              textbackslash n", pi*A1*B1);
3994
3995      printf (msg);
3996
3997
3998
3999      //-- case 1-3
4000
```

```
4001    A1 = 2.;  B1 = 1.;  H1 = 0.;  K1 = 0.;  PHI_1 = 0.;
4002
4003    A2 = 1.5;  B2 = 0.75;  H2 = -2.01796398085;  K2 = 1.25;  PHI_2 = pi/4.;
4004
4005    area = ellipse_ellipse_overlap (PHI_1, A1, B1, H1, K1,
4006
4007                                        PHI_2, A2, B2, H2, K2, \&rtn);
4008
4009    sprintf (msg,"Case 1-3: area = %15.8f, return\_value = \%d\
                textbackslash n", area, rtn);
4010
4011    printf (msg);
4012
4013    printf ("            Ellipses are disjoint, ovelap area = 0.0\
                textbackslash n\textbackslash n");
4014
4015
4016
4017    //-- case 2-1
4018
4019    A1 = 3.;  B1 = 2.;  H1 = 0.;  K1 = 0.;  PHI_1 = 0.;
4020
4021    A2 = 2.25;  B2 = 1.5;  H2 = 0.;  K2 = 0.;  PHI_2 = pi/4.;
4022
4023    area = ellipse_ellipse_overlap (PHI_1, A1, B1, H1, K1,
4024
4025                                        PHI_2, A2, B2, H2, K2, \&rtn);
4026
4027    sprintf (msg,"Case 2-1: area = \%15.8f, return_value = \%d\
                textbackslash n", area, rtn);
4028
4029    printf (msg);
4030
4031    sprintf (msg,"            ellipse 2 area by pi*a2*b2 = \%15.8f\
                textbackslash n", pi*A2*B2);
4032
4033    printf (msg);
4034
4035
4036
4037    //-- case 2-2
4038
4039    A1 = 2.;  B1 = 1.;  H1 = 0.;  K1 = 0.;  PHI_1 = 0.;
4040
4041    A2 = 3.;  B2 = 1.7;  H2 = 0.;  K2 = 0.;  PHI_2 = pi/4.;
4042
4043    area = ellipse_ellipse_overlap (PHI_1, A1, B1, H1, K1,
4044
4045                                        PHI_2, A2, B2, H2, K2, \&rtn);
4046
4047    sprintf (msg,"Case 2-2: area = \%15.8f, return_value = \%d\
                textbackslash n", area, rtn);
4048
4049    printf (msg);
4050
4051    sprintf (msg,"            ellipse 1 area by pi*a1b1 = \%15.8f\
                textbackslash n", pi*A1*B1);
4052
4053    printf (msg);
4054
4055
4056
4057    //-- case 2-3
4058
4059    A1 = 3.;  B1 = 2.;  H1 = 0.;  K1 = 0.;  PHI_1 = 0.;
4060
4061    A2 = 2.;  B2 = 1.;  H2 = -2.;  K2 = -1.;  PHI_2 = pi/4.;
4062
4063    area = ellipse_ellipse_overlap (PHI_1, A1, B1, H1, K1,
4064
4065                                        PHI_2, A2, B2, H2, K2, \&rtn);
4066
```

```
4067        sprintf (msg,"Case 2−3: area = \%15.8f, return\_value = \%d\
                    textbackslash n\textbackslash n", area, rtn);
4068
4069        printf (msg);
4070
4071
4072
4073        //−− case 3−1
4074
4075        A1 = 3.; B1 = 2.; H1 = 0.; K1 = 0.; PHI_1 = 0.;
4076
4077        A2 = 3.; B2 = 1.; H2 = 1.; K2 = 0.35; PHI_2 = pi/4.;
4078
4079        area = ellipse_ellipse_overlap (PHI_1, A1, B1, H1, K1,
4080
4081                                        PHI_2, A2, B2, H2, K2, \&rtn);
4082
4083        sprintf (msg,"Case 3−1: area = \%15.8f, return\_value = \%d\
                    textbackslash n", area, rtn);
4084
4085        printf (msg);
4086
4087
4088
4089        //−− case 3−2
4090
4091        A1 = 2.; B1 = 1.; H1 = 0.; K1 = 0.; PHI_1 = 0.;
4092
4093        A2 = 2.25; B2 = 1.5; H2 = 0.3; K2 = 0.; PHI_2 = pi/4.;
4094
4095        area = ellipse_ellipse_overlap (PHI_1, A1, B1, H1, K1,
4096
4097                                        PHI_2, A2, B2, H2, K2, \&rtn);
4098
4099        sprintf (msg,"Case 3−2: area = \%15.8f, return\_value = \%d\
                    textbackslash n\textbackslash n", area, rtn);
4100
4101        printf (msg);
4102
4103
4104
4105        //−− case 4−1
4106
4107        A1 = 3.; B1 = 2.; H1 = 0.; K1 = 0.; PHI_1 = 0.;
4108
4109        A2 = 3.; B2 = 1.; H2 = 1.; K2 = −0.5; PHI_2 = pi/4.;
4110
4111        area = ellipse_ellipse_overlap (PHI_1, A1, B1, H1, K1,
4112
4113                                        PHI_2, A2, B2, H2, K2, \&rtn);
4114
4115        sprintf (msg,"Case 4−1: area = \%15.8f, return_value = \%d\
                    textbackslash n", area, rtn);
4116
4117        printf (msg);
4118
4119
4120
4121        return rtn;
4122
4123    }
```

REFERENCES

[1] Kent, S., Kaiser, M. E., Deustua, S. E., Smith, J. A. *Photometric calibrations for 21st century science*, Astronomy 2010 **8** (2009).

[2] M. Chraibi, A. Seyfried, and A. Schadschneider, *Generalized centrifugal force model for pedestrian dynamics*, Phys. Rev. E, **82** (2010), 046111.

[3] Nonweiler, Terence R.F., *CACM Algorithm 326: Roots of low order polynomials*, Communications of the ACM, vol. **11** no. 4, pages 269-270 (1968). Translated into c and programmed by M. Dow, ANUSF, Australian National University, Canberra, Australia. Accessed at http://www.netlib.org/toms/326.

[4] Abramowitz, M. and Stegun, I. A. (Eds.). *Solutions of Quartic Equations.*

E-mail address: gbhughes@calpoly.edu

E-mail address: m.chraibi@fz-juelich.de