Distance from a Point to an Ellipse in 2D

David Eberly
Geometric Tools, LLC
http://www.geometrictools.com/

Created: January 17, 2004
Last Modified: March 1, 2008

Contents

1 Introduction

It is sufficient to solve this problem when the ellipse is centered at the origin and is axis-aligned with the major axis on the x-axis. Other ellipses can be rotated and translated to such an ellipse and the distance can be measured in that system. The basic idea is in Graphics Gems IV in an article by John Hart on computing distance between point and ellipsoid, but that article focuses on a conversion of the problem to computing the largest root of a polynomial. The presentation and implementation here avoids the conversion to polynomials, instead working directly with a rational function that appears in the aforementioned article. However, the polynomial conversion is mentioned to see how it relates to the rational function.

Let (u, v) be the point in question. Let the ellipse be

$$\left(\frac{x}{a}\right)^2 + \left(\frac{y}{b}\right)^2 = 1 \tag{1}$$

with $a > b$. The closest point (x, y) on the ellipse to (u, v) must occur so that $(u - x, v - y)$ is normal to the ellipse. An outward pointing ellipse normal is

$$\frac{1}{2}\nabla\left(\left(\frac{x}{a}\right)^2 + \left(\frac{y}{b}\right)^2 - 1\right) = \left(\frac{x}{a^2}, \frac{y}{b^2}\right) \tag{2}$$

The orthogonality condition is

$$(u - x, v - y) = t\left(\frac{x}{a^2}, \frac{y}{b^2}\right) \tag{3}$$

for some value t. We need to use equations (1) and (3) to determine the value of t. We may restrict our attention to $u \geq 0$ and $v \geq 0$. Points in other quadrants may be handled by symmetry. For example, the distance from $(-u, v)$ to the ellipse is the same as the distance from (u, v) to the ellipse. Also by symmetry, if (u, v) is in the first quadrant, the closest ellipse point (x, y) is in the first quadrant.

2 The Simple Cases

Let $(u, v) = (0, 0)$. The closest point is clearly $(0, b)$ since $a > b$.

> *The closest point to* $(0, 0)$ *is* $(0, b)$ *with distance* $d = b$.

If $u > 0$ and $v = 0$, the test point is on the u-axis, not at the origin. Imagine moving a point starting at the origin towards $(a, 0)$ along the u-axis. When $u = 0$, the closest point is $(0, b)$. As you increase the value of u, the closest point is (x, y) where x is a small positive number and y is slightly smaller than b. The second component of the vector in equation (3) is $-y = ty/b^2$. Since $y > 0$, it must be that $t = -b^2$. Replacing this value of t in the first component of the vector in equation (3) and solving for x, we obtain

$$x = \frac{a^2 u}{a^2 - b^2} \tag{4}$$

By our construction, we know $x > 0$. Equation (4) clearly satisifes this condition since $u > 0$ and $a > b$. We also know that since (x, y) is on the ellipse, $x \leq a$ is required. Thus,

$$\frac{a^2 u}{a^2 - b^2} \leq a$$

in which case

$$u \le a - \frac{b^2}{a} = a\left(1 - \frac{b^2}{a^2}\right) < a$$

What this means geometrically is that as you move $(u, 0)$ to the right until $u = a - b^2/a$, the (x, y) value moves to the right along the ellipse until $x = a$. For $u \ge a - b^2/a$, the point $(a, 0)$ is the closest ellipse point to $(u, 0)$. In summary,

> *The closest point to* $(u, 0)$ *for* $0 < u < a - b^2/a$ *is* (x, y) *with* $x = a^2u/(a^2 - b^2)$ *and* $y = b\sqrt{1 - (x/a)^2}$ *with distance* $d = \sqrt{(x-u)^2 + y^2} = b\sqrt{1 - u^2/(a^2 - b^2)}$. *The closest point to* $(u, 0)$ *for* $a - b^2/a \le u \le a$ *is* $(a, 0)$ *with distance* $d = a - u$.

Let $u = 0$ and $v \neq 0$. Equation (3) implies $x = 0$, where the candidate closest point is $(0, b)$, or $x \neq 0$ and $t = -a^2$, and the candidate closest points satisfy

$$y = \frac{-b^2 v}{a^2 - b^2} \tag{5}$$

For $v > 0$ equation (5) implies $y < 0$, so clearly $(0, b)$ is the closest point to $(0, v)$ for $v > 0$.

> *The closest point to* $(0, v)$ *for* $0 < v < b$ *is* $(0, b)$ *with distance* $d = b$.

The last case of $u > 0$ and $v > 0$ requires quite a bit more analysis, as is shown in the next section.

3 The Hard Case

Consider when $u > 0$ and $v > 0$. Equation (3) implies that $u - x = tx/a^2$ and $v - y = ty/b^2$ for some t. Solving yields

$$x = \frac{a^2 u}{t + a^2} \quad \text{and} \quad y = \frac{b^2 v}{t + b^2} \tag{6}$$

From the symmetry of the ellipse the closest point (x, y) should be in the first quadrant, so we need $x > 0$ and $y > 0$. These constraints force $t > -a^2$ and $t > -b^2$. Since $a > b$, the total constraint on t is $t > -b^2$.

Replacing equation (6) in the ellipse equation (1) yields

$$F(t) = \left(\frac{au}{t + a^2}\right)^2 + \left(\frac{bv}{t + b^2}\right)^2 - 1 = 0 \tag{7}$$

which is a rational function of t. The roots of $F(t)$ provide the candidate points for being closest to (u, v). At this time, the Graphics Gems article multiplies through by the polynomial terms in the denominator to obtain a quartic polynomial in t. The roots of that polynomial are analyzed. A claim is made that the largest root is what leads to the closest point, but in the case when (u, v) is inside the ellipse, it is not immediately clear why this must be the case. The presentation here does make it clear that the largest root of $F(t)$ is what we need.

The first derivative of F is

$$F'(t) = \frac{-2a^2u^2}{(t + a^2)^3} + \frac{-2b^2v^2}{(t + b^2)^3}$$

and the second derivative is

$$F''(t) = \frac{6a^2u^2}{(t+a^2)^4} + \frac{6b^2v^2}{(t+b^2)^4}$$

Observe that for any point (u, v) with $u > 0$ and $v > 0$, $F'(t) < 0$ and $F''(t) > 0$ for $t \in (-b^2, 0]$. Also notice that the one-sided limit of $F(t)$ as t approaches $-b^2$ from the right is

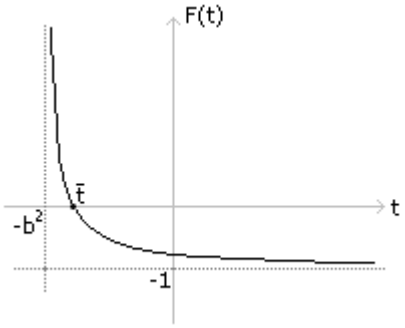
$$\lim_{t \to -b^{2+}} F(t) = +\infty$$

The limit as t becomes unbounded is

$$\lim_{t \to \infty} F(t) = -1$$

What we have shown is that $F(t)$ is a strictly decreasing function for $t \in (-b^2, +\infty)$ that is initially positive, then becomes negative. Consequently it has a *unique root* on the specified domain. Figure 3.1 shows a typical graph of $F(t)$.

Figure 3.1 A typical graph of $F(t)$ for $t > -b^2$. The unique root $\bar{t}$ is shown.



Clearly from the construction, the unique root $\bar{t} \in (-b^2, \infty)$ is the largest root of $F(t)$.

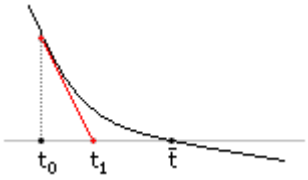
4 Newton's Method

The convexity of F on its domain, namely that $F''(t) > 0$, makes Newton's method an ideal numerical method for locating the root. Given an initial guess t_0, the Newton iterates are

$$t_{n+1} = t_n - \frac{F(t_n)}{F'(t_n)}, \quad n \geq 0$$

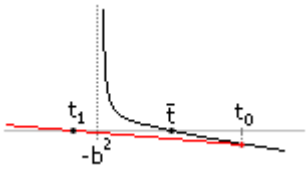
It is important to choose an initial guess for which the method converges. Newton's method has an intuitive geometric appeal to it. The next value t_{n+1} is computed by determining where the tangent line to the graph at $(t_n, F(t_n))$ intersects the t-axis. The intersection point is $(t_{n+1}, 0)$. If we choose an initial guess $t_0 < \bar{t}$, the tangent line to $(t_0, F(t_0))$ intersects the t-axis at $(t_1, 0)$ where $t_0 < t_1 < \bar{t}$. Figure 4.1 illustrates this.

Figure 4.1 An initial guess t_0 to the left of $\bar{t}$ guarantees $t_0 < t_1 < \bar{t}$.



An initial guess $t_0 > \bar{t}$ leads to a new iterate $t_1 < \bar{t}$, but potentially $t_1 < -b^2$ which puts it outside the domain of interest, namely $(-b^2, \infty)$. Figure 4.2 illustrates this.

Figure 4.2 An initial guess t_0 to the right of $\bar{t}$ guarantees $t_1 < \bar{t}$, but does not guarantee $t_1 > -b^2$.



To avoid this potential problem, it is better to choose an initial guess to the left of the root. Any t_0 for which $F(t_0) > 0$ will work. In particular, you can choose $t_0 = bv - b^2$, in which case $F(t_0) = [au/(bv - b^2 + a^2)]^2 > 0$.

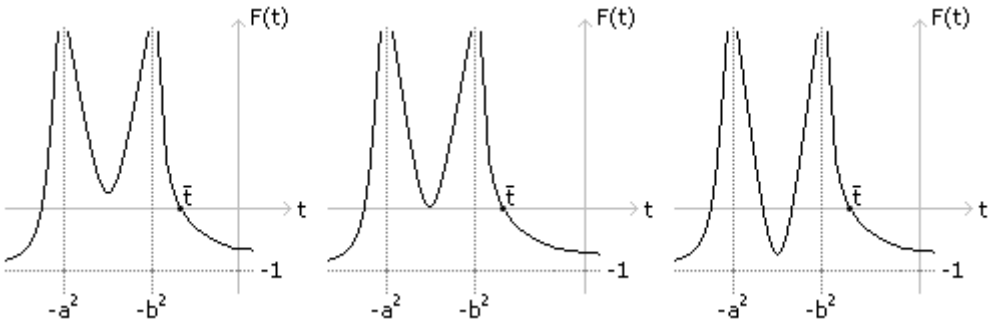
5 Reduction to a Polynomial

The method discussed in Graphics Gems IV eliminates the denominators in $F(t)$ by multiplication to obtain a quartic polynomial equation

$$P(t) = (t + a^2)^2(t + b^2)^2 F(t) = a^2u^2(t + b^2)^2 + b^2v^2(t + a^2)^2 - (t + a^2)^2(t + b^2)^2 = 0$$

Excluding $t = -a^2$ and $t = -b^2$, the roots of $P(t)$ and $F(t)$ must be the same. A quartic polynomial has at most 4 roots, so F has at most 4 roots. Since we already saw that the largest root of $F(t)$ produces the closest point on the ellipse, the largest root of $P(t)$ produces that closest point. Figure 5.1 shows three possibilities for the graph of $F(t)$. The roots in the figure are necessarily those of $P(t)$.

Figure 5.1 Three possibilities for the graph of $F(t)$.



The three possibilities are based on where (u, v) is located with respect to the *evolute* of the ellise

$$(ax)^{2/3} + (by)^{2/3} = (a^2 - b^2)^{2/3}$$

The left image corresponds to (u, v) outside the evolute, so $(au)^{2/3}+(bv)^{2/3} > (a^2-b^2)^{2/3}$. The middle image corresponds to (u, v) on the evolute itself, so $(au)^{2/3} + (bv)^{2/3} = (a^2 - b^2)^{2/3}$. The right image corresponds to (u, v) inside the evolute, so $(au)^{2/3} + (bv)^{2/3} < (a^2 - b^2)^{2/3}$. [You can generate the evolute equation by eliminating t from the simultaneous equations $F(t) = 0$ and $F'(t) = 0$.]

The Graphics Gems IV article shows how to choose an initial guess that is larger than the largest root of $P(t)$. The idea is to guarantee convergence of the iterates, analogous to Figure 2 for $F(t)$. Notice that

$$P(0) = a^4 b^4 \left(\frac{u^2}{a^2} + \frac{v^2}{b^2} - 1 \right)$$

If (u, v) is outside the ellipse, then $P(0) > 0$. Notice that the orthogonality conditions is

$$(u, v) = (x, y) + t\vec{N}$$

where $\vec{N} = (x/a^2, y/b^2)$. The point (u, v) is on the normal line to the ellipse at (x, y). Since (u, v) is outside, it must be that $\bar{t} > 0$ for the closest point. The other candidate points have normals that point away from (u, v), so their t-values are negative. The distance to the closest point is

$$d = |(u, v) - (x, y)| = \bar{t}|\vec{N}|$$

so that

$$\bar{t} = \frac{d}{|\vec{N}|}$$

The distance from (u, v) to (x, y) is clearly larger than the distance from (u, v) to the origin $(0, 0)$,

$$d < |(u, v)|$$

The normal squared length is

$$\begin{aligned} |\vec{N}|^2 &= \frac{x^2}{a^4} + \frac{y^2}{b^4} \\ &= \frac{1}{a^2}\left(\frac{x^2}{a^2} + \frac{a^2}{b^2}\frac{y^2}{b^2}\right) \\ &\geq \frac{1}{a^2}\left(\frac{x^2}{a^2} + \frac{y^2}{b^2}\right) && \text{since } a > b \\ &= \frac{1}{a^2} && \text{since } (x,y) \text{ is on the ellipse} \end{aligned}$$

Consequently,

$$\bar{t} = \frac{d}{|\vec{N}|} < \frac{|(u,v)|}{1/a} = a|(u,v)| = t_0$$

In general, the initial guess is $t_0 = \max\{a,b\}|(u,v)|$.

If (u,v) is inside the ellipse, then $P(0) < 0$ and all roots of $P(t)$ are negative (we showed this for $F(t)$). In this case, $t_0 = 0$ is a point larger than the maximum root, so is a good initial guess.

In the application of Newton's method to $F(t) = 0$, we chose the initial guess t_0 to be smaller than the root $\bar{t}$, whereas the Graphics Gems IV article chooses t_0 larger than the root. A simple explanation for this. We know $F''(t) > 0$ for $t > -b^2$, so the graph of $F(t)$ is convex (opens upwards). The graph of a quartic polynomial for t larger than the maximum root is concave (opens downwards). The tangent lines for the graph of the quartic, but to the right of the maximum root will generate new iterates to the right of the root, but closer than the current iterate.

6 An Implementation

The following code is an implementation of Newton's method for $F(t)$, not the polynomial $P(t)$. The main program uses the image classes in Wild Magic.

```
#include "WmlImages.h"
#include "WmlMath.h"
using namespace Wml;

//----------------------------------------------------------------------------
double DistancePointEllipseSpecial (double dU, double dV, double dA,
    double dB, const double dEpsilon, const int iMax, int& riIFinal,
    double& rdX, double& rdY)
{
    // initial guess
    double dT = dB*(dV - dB);

    // Newton's method
    int i;
    for (i = 0; i < iMax; i++)
    {
        double dTpASqr = dT + dA*dA;
        double dTpBSqr = dT + dB*dB;
        double dInvTpASqr = 1.0/dTpASqr;
```

```
        double dInvTpBSqr = 1.0/dTpBSqr;
        double dXDivA = dA*dU*dInvTpASqr;
        double dYDivB = dB*dV*dInvTpBSqr;
        double dXDivASqr = dXDivA*dXDivA;
        double dYDivBSqr = dYDivB*dYDivB;
        double dF = dXDivASqr + dYDivBSqr - 1.0;
        if (dF < dEpsilon)
        {
            // F(t0) is close enough to zero, terminate the iteration
            rdX = dXDivA*dA;
            rdY = dYDivB*dB;
            riIFinal = i;
            break;
        }

        double dFDer = 2.0*(dXDivASqr*dInvTpASqr + dYDivBSqr*dInvTpBSqr);
        double dRatio = dF/dFDer;
        if (dRatio < dEpsilon)
        {
            // t1-t0 is close enough to zero, terminate the iteration
            rdX = dXDivA*dA;
            rdY = dYDivB*dB;
            riIFinal = i;
            break;
        }

        dT += dRatio;
    }

    if (i == iMax)
    {
        // method failed to converge, let caller know
        riIFinal = -1;
        return -FLT_MAX;
    }

    double dDelta0 = rdX - dU, dDelta1 = rdY - dV;
    return sqrt(dDelta0*dDelta0 + dDelta1*dDelta1);
}
//----------------------------------------------------------------------------
double DistancePointEllipse (
    double dU, double dV,      // test point (u,v)
    double dA, double dB,      // ellipse is (x/a)^2 + (y/b)^2 = 1
    const double dEpsilon,     // zero tolerance for Newton's method
    const int iMax,            // maximum iterations in Newton's method
    int& riIFinal,             // number of iterations used
    double& rdX, double& rdY)  // a closest point (x,y)
{
    // special case of circle
    if (fabs(dA-dB) < dEpsilon)
    {
        double dLength = sqrt(dU*dU+dV*dV);
```

```
    return fabs(dLength - dA);
}

// reflect U = -U if necessary, clamp to zero if necessary
bool bXReflect;
if (dU > dEpsilon)
{
    bXReflect = false;
}
else if (dU < -dEpsilon)
{
    bXReflect = true;
    dU = -dU;
}
else
{
    bXReflect = false;
    dU = 0.0;
}

// reflect V = -V if necessary, clamp to zero if necessary
bool bYReflect;
if (dV > dEpsilon)
{
    bYReflect = false;
}
else if (dV < -dEpsilon)
{
    bYReflect = true;
    dV = -dV;
}
else
{
    bYReflect = false;
    dV = 0.0;
}

// transpose if necessary
double dSave;
bool bTranspose;
if (dA >= dB)
{
    bTranspose = false;
}
else
{
    bTranspose = true;
    dSave = dA;
    dA = dB;
    dB = dSave;
    dSave = dU;
    dU = dV;
```

```
    dV = dSave;
}

double dDistance;
if (dU != 0.0)
{
    if (dV != 0.0)
    {
        dDistance = DistancePointEllipseSpecial(dU,dV,dA,dB,dEpsilon,iMax,
            riIFinal,rdX,rdY);
    }
    else
    {
        double dBSqr = dB*dB;
        if (dU < dA - dBSqr/dA)
        {
            double dASqr = dA*dA;
            rdX = dASqr*dU/(dASqr-dBSqr);
            double dXDivA = rdX/dA;
            rdY = dB*sqrt(fabs(1.0-dXDivA*dXDivA));
            double dXDelta = rdX - dU;
            dDistance = sqrt(dXDelta*dXDelta+rdY*rdY);
            riIFinal = 0;
        }
        else
        {
            dDistance = fabs(dU - dA);
            rdX = dA;
            rdY = 0.0;
            riIFinal = 0;
        }
    }
}
else
{
    dDistance = fabs(dV - dB);
    rdX = 0.0;
    rdY = dB;
    riIFinal = 0;
}

if (bTranspose)
{
    dSave = rdX;
    rdX = rdY;
    rdY = dSave;
}

if (bYReflect)
{
    rdY = -rdY;
    }
```

```
    if (bXReflect)
    {
        rdX = -rdX;
        }

    return dDistance;
}
//----------------------------------------------------------------------------
int main ()
{
    // The ellipse is (x/2)^2 + (y/1)^2 = 1.  Compute distances for points
    // (u,v) with |u| <= 3 and |v| <= 3.
    double dA = 2.0, dB = 1.0;
    const double dEpsilon = 1e-08;
    const int iMax = 32;
    double dUMin = -3.0, dUMax = 3.0;
    double dVMin = -3.0, dVMax = 3.0;
    double dU, dV, dX, dY, dDistance;
    int iX, iY, iIFinal, iMaxIFinal = 0;

    const int iXBound = 256, iYBound = 256;
    ImageDouble2D kImage(iXBound,iYBound);
    ImageInt2D kIndex(iXBound,iYBound);
    for (iY = 0; iY < iYBound; iY++)
    {
        dV = dVMin + (dVMax-dVMin)*iY/iYBound;
        for (iX = 0; iX < iXBound; iX++)
        {
            dU = dUMin + (dUMax-dUMin)*iX/iXBound;
            dDistance = DistancePointEllipse(dU,dV,dA,dB,dEpsilon,iMax,
                iIFinal,dX,dY);
            kImage(iX,iY) = dDistance;
            kIndex(iX,iY) = iIFinal;
            if (iIFinal > iMaxIFinal)
            {
                iMaxIFinal = iIFinal;
            }
        }
    }

    // The point (umax,vmax) has maximal distance.  Color the image so that
    // points inside the ellipse are blue with intensity proportional to
    // distance.  Color points outside red with intensity proportional to
    // distance.
    double dMaxDistance = kImage(255,0);
    ImageRGB82D kColor(iXBound,iYBound);
    ImageDouble2D kTest(iXBound,iYBound);

    for (iY = 0; iY < iYBound; iY++)
    {
        dV = dVMin + (dVMax-dVMin)*iY/iYBound;
```

```
    for (iX = 0; iX < iXBound; iX++)
    {
        dU = dUMin + (dUMax-dUMin)*iX/iXBound;
        dDistance = kImage(iX,iY);

        double dURatio = dU/dA;
        double dVRatio = dV/dB;
        double dTest = dURatio*dURatio + dVRatio*dVRatio - 1.0;
        kTest(iX,iYBound-1-iY) = dTest;
        unsigned char ucGray;
        if (dTest > 0.0)
        {
            ucGray = (unsigned char)(255.0f*dDistance/dMaxDistance);
            kColor(iX,iY) = GetColor24(ucGray,0,0);
        }
        else
        {
            ucGray = (unsigned char)(255.0f*dDistance);
            kColor(iX,iY) = GetColor24(0,0,ucGray);
        }
    }
}

// draw ellipse
iMaxIFinal++;
for (iY = 1; iY < iYBound-1; iY++)
{
    for (iX = 1; iX < iXBound-1; iX++)
    {
        if (kTest(iX,iY) <= 0.0)
        {
            if (kTest(iX-1,iY-1) > 0.0
            ||  kTest(iX  ,iY-1) > 0.0
            ||  kTest(iX+1,iY-1) > 0.0
            ||  kTest(iX-1,iY  ) > 0.0
            ||  kTest(iX+1,iY  ) > 0.0
            ||  kTest(iX-1,iY+1) > 0.0
            ||  kTest(iX  ,iY+1) > 0.0
            ||  kTest(iX+1,iY+1) > 0.0)
            {
                kColor(iX,iY) = GetColor24(128,128,128);
                kIndex(iX,iY) = iMaxIFinal;
            }
        }
    }
}

kImage.Save("distance.im");
kIndex.Save("index.im");
kColor.Save("color.im");

return 0;
```

```
}
//----------------------------------------------------------------------------
```

The output consists of three images. The image `distance.im` stores the distances from the test points to the ellipse. The image `index.im` stores the number of iterations that Newton's method required for convergence according to the specified threshold. The image `color.im` is a pseudocoloring of the distance values to give a different visualization of distance than the gray scale `distance.im`. The following figures show screen shots of the images.

Figure 6.1 The distances from points to the ellipse shown as a gray scale image. The brighter the image value, the larger the distance.

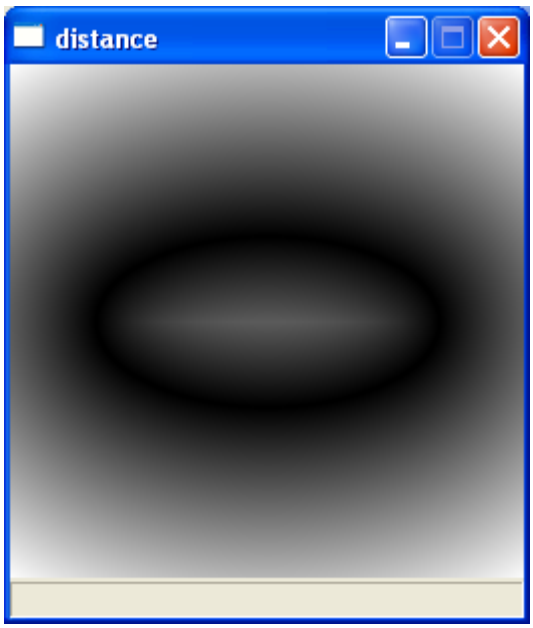


Figure 6.2 The number of iterations in Newton's method to obtain convergence to the distance. (The text in the image was added separately from the program.)

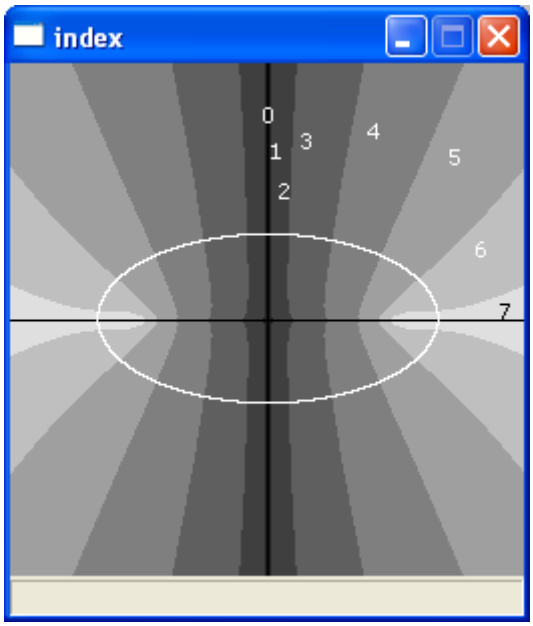


Figure 6.3 The distances from points to the ellipse shown as a color scale.