Distance from a Point to an Ellipsoid

David Eberly
Magic Software, Inc.
http://www.magic-software.com

Created: October 13, 2002 (moved here from LeastSquaresFitting.pdf)

It is sufficient to solve this problem when the ellipsoid is axis–aligned. For other ellipsoids, they can be rotated and translated to an axis–aligned ellipsoid centered at the origin and the distance can be measured in that system. The basic idea can be found in Graphics Gems IV (an article by John Hart on computing distance between point and ellipsoid).

Let (u, v, w) be the point in question. Let the ellipse be $(x/a)^2 + (y/b)^2 + (z/c)^2 = 1$. The closest point (x, y, z) on the ellipsoid to (u, v) must occur so that $(x - u, y - v, z - w)$ is normal to the ellipsoid. Since an ellipsoid normal is $\nabla((x/a)^2 + (y/b)^2 + (z/c)^2) = (x/a^2, y/b^2, z/c^2)$, the orthogonality condition implies that $u - x = t * x/a^2$, $v - y = t * y/b^2$, and $w - z = t * z/c^2$ for some t. Solving yields $x = a^2u/(t + a^2)$, $y = b^2v/(t + b^2)$, and $z = c^2w/(t + c^2)$. Replacing in the ellipsoid equation yields

$$\left(\frac{au}{t+a^2}\right)^2 + \left(\frac{bv}{t+b^2}\right)^2 + \left(\frac{cw}{t+c^2}\right)^2 = 1.$$

Multiplying through by the denominators yields the sixth degree polynomial

$$F(t) = (t+a^2)^2(t+b^2)^2(t+c^2)^2 - a^2u^2(t+b^2)^2(t+c^2)^2 - b^2v^2(t+a^2)^2(t+c^2)^2 - c^2w^2(t+a^2)^2(t+b^2)^2 = 0.$$

The largest root $\bar{t}$ of the polynomial corresponds to the closest point on the ellipse.

The largest root can be found by a Newton's iteration scheme. If (u, v, w) is inside the ellipse, then $t_0 = 0$ is a good initial guess for the iteration. If (u, v, w) is outside the ellipse, then $t_0 = \max\{a, b, c\}\sqrt{u^2 + v^2 + w^2}$ is a good initial guess. The iteration itself is

$$t_{i+1} = t_i - F(t_i)/F'(t_i), \quad i \geq 0.$$

Some numerical issues need to be addressed. For (u, v, w) near the coordinate planes, the algorithm is ill–conditioned. You need to handle those cases separately. Also, if a, b, and c are large, then $F(t_i)$ can be quite large. In these cases you might consider uniformly scaling the data to $O(1)$ as floating point numbers first, compute distance, then rescale to get the distance in the original coordinates.