Fitting an Ellipse to a Set of Data Points

Nicky van Foreest

2020-09-22

1 THE PROBLEM

Given a set of points $\mathbf{x}_i = (x_i, y_i)$ find the best (in a least squares sense) ellipse that fits the points.

Here I follow an approach suggested by Fitzgibbon, Pilu and Fischer in Fitzgibbon, A.W., Pilu, M., and Fischer R.B., *Direct least squares fitting of ellipsees*, Proc. of the 13th Internation Conference on Pattern Recognition, pp 253-{}-257, Vienna, 1996. I learned of this approach from Peter Snoeren, whose matlab I rewrote to the python code here.

Other interesting pages that discuss this topic:

- http://bec.physics.monash.edu.au/wiki/Main/FittingEllipses
- http://exnumerus.blogspot.com/2011/03/python-script-to-fit-ellipse-to-noisy.html

Note, the code below is much shorter than the code discussed on this last page, but perhaps less generic. For instance, it does not seem to work well for circles.

Even though I never use this code myself, it turns out to be quite a 'big' hit. I wrote it in 2006 to test a numerical integrator for the following case. Suppose we are flying in a rocket in a circular orbit around the earth. If we hit the brake for a very short time, the orbit must become elliptical: we lose kinetic energy, but as the potential energy remains the same, the rocket must reach the same height after completing an orbit.

I used python's numerical integrator to compute the rocket's orbit after hitting the brake. The orbit seems to be elliptical, but to be sure I used the code here.

2 THE APPROACH

An ellipse can be defined as a general conic, that is, as the set of points $\mathbf{x} = (x, y)$ such that

$$f(a, (x, y)) = D \cdot a = 0 \tag{1}$$

where $D = (x^2, xy, y^2, x, y, 1)$ and $a = (a_{x,x}, a_{x,y}, a_{y,y}, a_x, a_y, a_1)$, cf., http://mathworld.wolfram.com/QuadraticCurve.html.

We can fit the ellipse to N data points $\mathbf{x}_i$, $i = 1,\ldots,N$, by minimizing the distance

$$\Delta(a,\mathbf{x}) = \sum_{i=1}^{N} (f(a,\mathbf{x}_i))^2.$$

Rewriting this we obtain:

$$\Delta(a,\mathbf{x}) = \sum_{i=1}^{N} a^T D_i^T D_i a = a^T S a$$

where $S_i = \sum D_i^T D_i$ is a 6×6 scatter matrix.

The aim is find the a that minimizes $\Delta(a,\mathbf{x})$ but such that an ellipse results. Now it is known that the components of the vector a have to satisfy the constraint

$$4a_{x,x}a_{y,y} - a_{x,y}^2 > 0$$

if the corresponding conic is to be an ellipse. Written as a matrix equation and using the above vectorial form for a, this condition can be rewritten to $a^T C a > 0$ with

$$C = \begin{pmatrix} 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Next, observe that the conic condition, i.e., eq. (1), is independent of linear scaling in a. We therefore may just as well replace this condition by $a^T C a = \phi$, where ϕ is some positive number.

Combining the above results in the constrained minimization problem

$$\text{argmin}_a \{\Delta(a,\mathbf{x}) : a^T C a = \phi\}.$$

In other words, we are looking for the vector a that solves this minimization problem.

The standard method to solve the above constrained problem is to introduce a Lagrange multiplier λ and a Lagrangian

$$L(a,\lambda) = \Delta(a,\mathbf{x}) - \lambda(a^T C a - \phi) = a^T S a - \lambda(a^T C a - \phi)$$

and minimize $L(a,\lambda)$ as a function of a.

To solve this, take the gradient with respect to a and the derivative with respect to λ:

$$\begin{aligned} \partial_a L(a,\lambda) = 0 &\implies 2Sa - \lambda Ca = 0 \implies Sa = \lambda Ca \\ \partial_\lambda L(a,\lambda) = 0 &\implies a^T C a = \phi. \end{aligned}$$

Multiplying the first equation from the left with a^T and using the second equation we obtain

$$a^T S a = \lambda a^T C a = \lambda\phi.$$

Cleary, ϕ is arbitrary, but constant. Thus, to minimize $\Delta(a, \mathbf{x}) = a^T S a$ we are looking for the smallest λ that satisfies this equation.

Finally, observe that $Sa = \lambda Ca$ can be rewritten as a generalized eigenvalue problem

$$\frac{1}{\lambda} a = S^{-1} C a.$$

Thus, we have to solve this eigenvalue problem, and the a we are looking for is the eigenvector corresponding to the largest eigenvalue $1/\lambda$.

Once we have the minimizing a, we can compute the relevant aspects of the ellipse, namely, its center, its angle of rotation, and its axes.

3 THE CODE

First compute a along the lines as explained above.

```
from numpy.linalg import eig, inv, svd
from math import atan2
import numpy as np


def __fit_ellipse(x, y):
    x, y = x[:, np.newaxis], y[:, np.newaxis]
    D = np.hstack((x * x, x * y, y * y, x, y, np.ones_like(x)))
    S, C = np.dot(D.T, D), np.zeros([6, 6])
    C[0, 2], C[2, 0], C[1, 1] = 2, 2, -1
    U, s, V = svd(np.dot(inv(S), C))
    a = U[:, 0]
    return a
```

Here we just implement the functions of http://mathworld.wolfram.com/Ellipse.html that allow us to compute the ellipse from a.

The center.

```
def ellipse_center(a):
    b, c, d, f, g, a = a[1] / 2, a[2], a[3] / 2, a[4] / 2, a[5], a[0]
    num = b * b - a * c
    x0 = (c * d - b * f) / num
    y0 = (a * f - b * d) / num
    return np.array([x0, y0])
```

The major and minor axes of the ellipse.

```
def ellipse_axis_length(a):
    b, c, d, f, g, a = a[1] / 2, a[2], a[3] / 2, a[4] / 2, a[5], a[0]
    up = 2 * (a * f * f + c * d * d + g * b * b - 2 * b * d * f - a * c * g)
    down1 = (b * b - a * c) * (
        (c - a) * np.sqrt(1 + 4 * b * b / ((a - c) * (a - c))) - (c + a)
    )
    down2 = (b * b - a * c) * (
        (a - c) * np.sqrt(1 + 4 * b * b / ((a - c) * (a - c))) - (c + a)
    )
    res1 = np.sqrt(up / down1)
    res2 = np.sqrt(up / down2)
    return np.array([res1, res2])
```

The angle of rotation.

```
def ellipse_angle_of_rotation(a):
    b, c, d, f, g, a = a[1] / 2, a[2], a[3] / 2, a[4] / 2, a[5], a[0]
    return atan2(2 * b, (a - c)) / 2
```

And here is the main function to call.

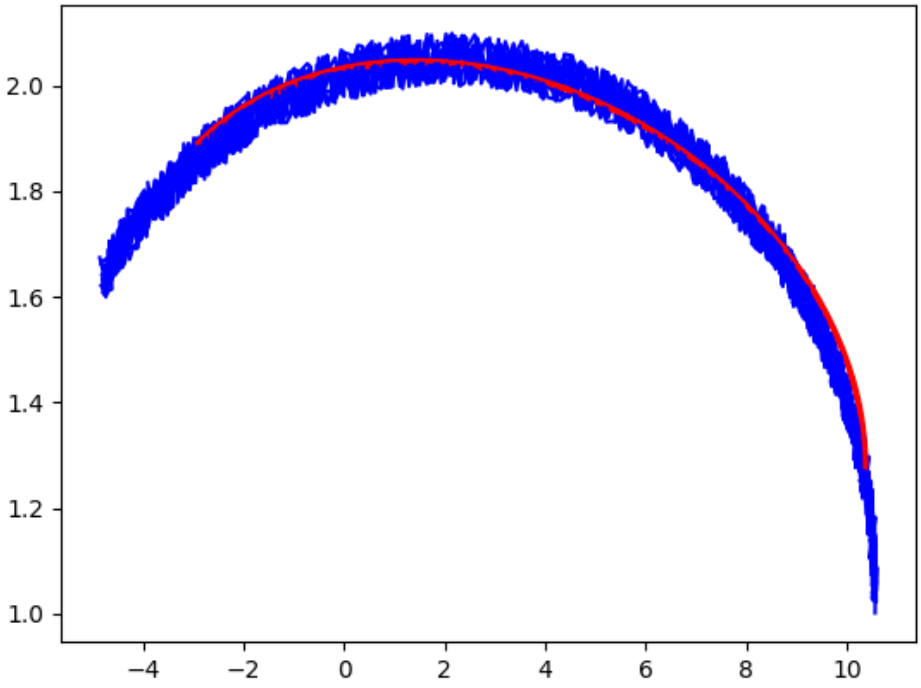
```
def fit_ellipse(x, y):
    """@brief fit an ellipse to supplied data points: the 5 params
        returned are:

        M - major axis length
        m - minor axis length
        cx - ellipse centre (x coord.)
        cy - ellipse centre (y coord.)
        phi - rotation angle of ellipse bounding box

    @param x first coordinate of points to fit (array)
    @param y second coord. of points to fit (array)
    """
    a = __fit_ellipse(x, y)
    centre = ellipse_center(a)
    phi = ellipse_angle_of_rotation(a)
    M, m = ellipse_axis_length(a)
    # assert that the major axix M > minor axis m
    if m > M:
        M, m = m, M
```

```
    # ensure the angle is betwen 0 and 2*pi
    phi -= 2 * np.pi * int(phi / (2 * np.pi))
    return [M, m, centre[0], centre[1], phi]
```

4 AN EXAMPLE

Here is an example to show how to use the code.

```
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt

arc = 0.8
R = np.arange(0,arc*np.pi, 0.01)
x = 8.5*np.cos(R) + 2 + 0.1*np.random.rand(len(R))
y = np.sin(R) + 1. + 0.1*np.random.rand(len(R))

M, m, c_x, c_y, phi = fit_ellipse(x,y)

xx = c_x + M*np.cos(R)*np.cos(phi) - m*np.sin(R)*np.sin(phi)
yy = c_y + M*np.cos(R)*np.sin(phi) + m*np.sin(R)*np.cos(phi)


plt.plot(x,y, color = 'blue')
plt.plot(xx,yy, color = 'red')
plt.savefig("ellipse_test.png")
# plt.close()
# plt.clf()

"ellipse_test.png"
```



Not too bad altogether.