Fitting an Ellipse to a Set of Data Points

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.

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.

You can find the code on github.

The Approach

We 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 development I present below.

A General Conic

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

$$f(a, (x, y)) = D \cdot a = 0$$

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

Including Data

Now 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 $S = \sum D_i^T D_i$ is a 6×6 scatter matrix.

A Constrained Minimization Problem

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 $4a_{x,x}a_{y,y} - a_{x,y}^2 > 0$ 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 $a^T Ca > 0$, where C is a 6×6 matrix filled with zeros except $C_{1,3} = C_{3,1} = 2$ and $C_{2,2} = -1$.

Next, observe that the conic condition $f(a, \mathbf{x}) = 0$ is independent of linear scaling in a. We therefore may just as well replace this condition by $a^T Ca = \phi$ for some positive number ϕ.

Combining the above results in the constrained minimization problem

$$\text{argmin}_a \{\Delta(a, \mathbf{x}) | a^T Ca = \phi\}$$

That is, the vector a that solves this problem corresponds to the ellipse we are looking for.

Formulation in terms of Lagrangian Multipliers

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

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

and minimize $L(a)$ as a function of a.

Now $\partial_a L(a) = 0$ results in the equation $Sa = \lambda Ca$. Now multiplying this from the left with a^T and using the constraint $a^T Ca = \phi$ (resulting from the derivative of $L(a)$ with respect to λ) we obtain that

$$a^T Sa = \lambda a^T Ca = \lambda \psi.$$

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

A Generalized Eigenvalue Problem

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

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

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

The Python Code

In code the above results in

```
import numpy as np
from numpy.linalg import eig, inv

def fitEllipse(x,y):
    x = x[:,np.newaxis]
    y = y[:,np.newaxis]
    D =  np.hstack((x*x, x*y, y*y, x, y, np.ones_like(x)))
    S = np.dot(D.T,D)
    C = np.zeros([6,6])
    C[0,2] = C[2,0] = 2; C[1,1] = -1
    E, V =  eig(np.dot(inv(S), C))
    n = np.argmax(np.abs(E))
    a = V[:,n]
    return a
```

Some Further Characteristics of the Fitted Ellips

Now that we have the minimizing a we can use a to compute the center of the ellipse, its angle of rotation and its axes. The relations between a and these characteristics can be found at http://mathworld.wolfram.com/Ellipse.html.

```
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])


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 0.5*np.arctan(2*b/(a-c))


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])
```

At http://mathworld.wolfram.com/Ellipse.html, it is suggested to compute the angle of rotation slightly differently, namely in accordance to the function below. (Thanks to Bjorn Sumner for pointing out an error in a previous version of the code.)

```
def ellipse_angle_of_rotation2( a ):
    b,c,d,f,g,a = a[1]/2, a[2], a[3]/2, a[4]/2, a[5], a[0]
    if b == 0:
        if a > c:
            return 0
        else:
            return np.pi/2
    else:
        if a > c:
            return np.arctan(2*b/(a-c))/2
        else:
            return np.pi/2 + np.arctan(2*b/(a-c))/2
```

An Example

Finally consider the following example:

```
arc = 0.8
R = np.arange(0,arc*np.pi, 0.01)
x = 1.5*np.cos(R) + 2 + 0.1*np.random.rand(len(R))
```

```
y = np.sin(R) + 1. + 0.1*np.random.rand(len(R))
```

This becomes

```
a = fitEllipse(x,y)
center = ellipse_center(a)
#phi = ellipse_angle_of_rotation(a)
phi = ellipse_angle_of_rotation2(a)
axes = ellipse_axis_length(a)

print("center = ",  center)
print("angle of rotation = ",  phi)
print("axes = ", axes)
```

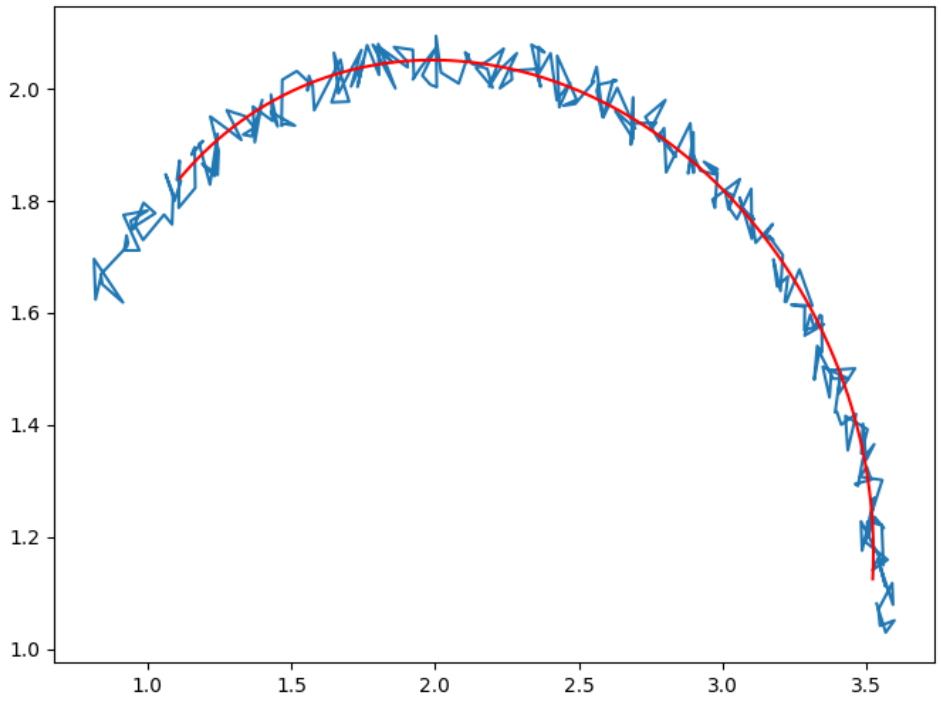
```
center =  [ 2.16218866  1.27017122]
angle of rotation =  -0.106259669841
axes =  [ 1.36751758  0.77201314]
```

As a plot:

```
a, b = axes
xx = center[0] + a*np.cos(R)*np.cos(phi) - b*np.sin(R)*np.sin(phi)
yy = center[1] + a*np.cos(R)*np.sin(phi) + b*np.sin(R)*np.cos(phi)

import matplotlib.pylab as plt

plt.plot(x,y)
plt.plot(xx,yy, color = 'red')
plt.show()
```



Not too bad altogether. By increasing the arc the results improve (that is, some simple testing shows this).