In [2]:

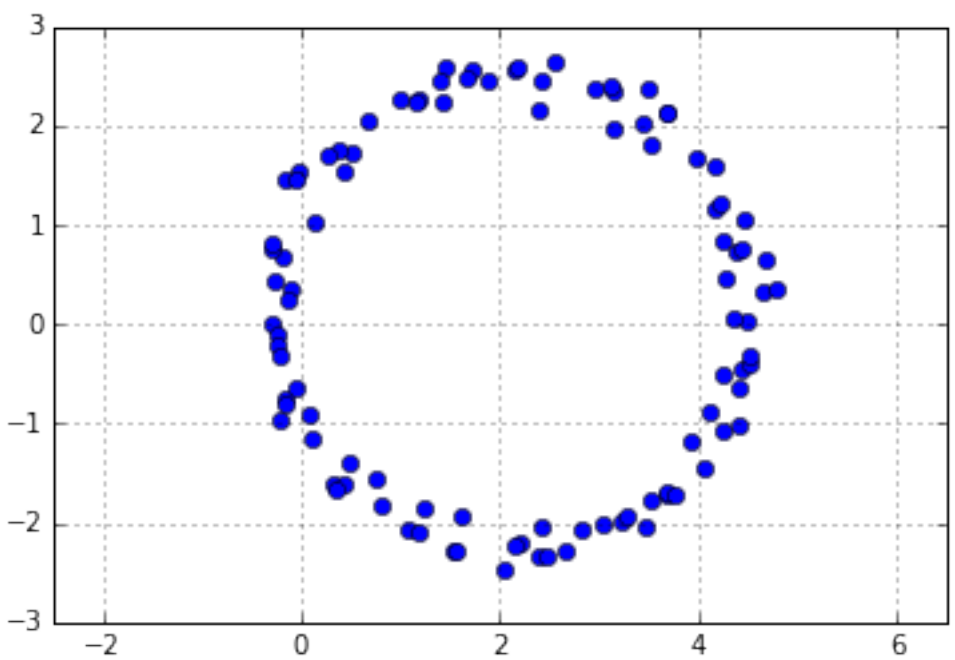
```
%matplotlib inline

import matplotlib
import numpy as np
import matplotlib.pyplot as pt
import numpy.linalg as la
```

create x and y coordinates of a circle using polar coordinates

introduce "noise" by adding normally distributed random numbers

In [96]:

```
np.random.seed(555)
t = np.linspace(0,2*np.pi,100)
xc = 3*np.random.rand() #x-coordinate of center of circle
yc = 3*np.random.rand() #y-coordinate of center of circle
r = 2*np.random.rand() + 0.5 #radius of circle

x = r*np.cos(t) + np.random.normal(scale = 1.0/7, size = len(t)) + xc
y = r*np.sin(t) + np.random.normal(scale = 1.0/7, size = len(t)) + yc


pt.plot(x,y,'o')
pt.grid(True)
pt.axes().set_aspect('equal','datalim')
```

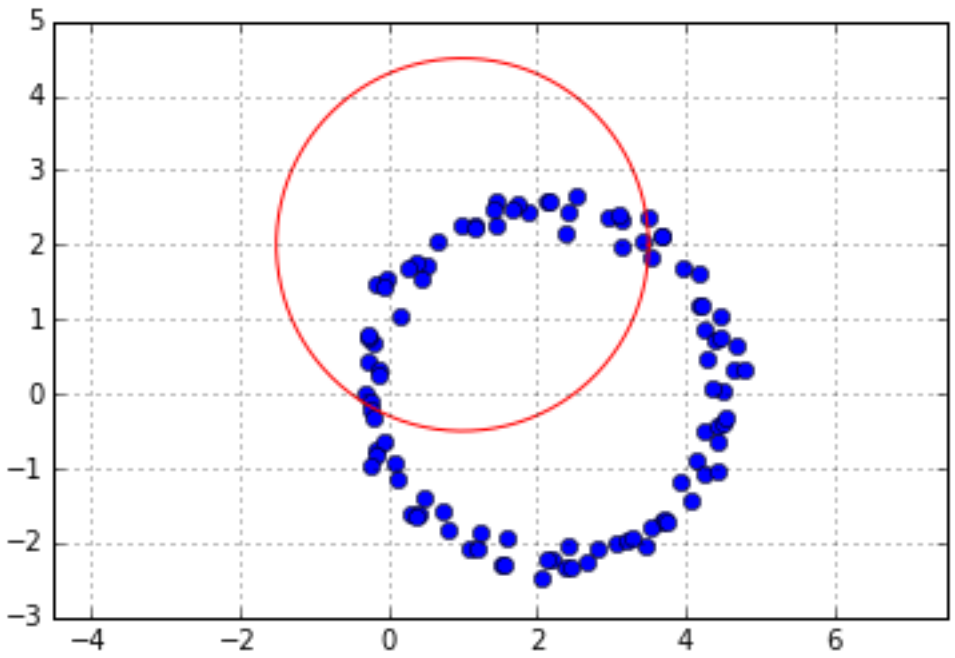


what circle best describes this data??

Let's use the data to "guess" the radius and center

In [97]:

```
xc = 1.0
yc = 2.0
r = 2.5
xapprox = r*np.cos(t) + xc
yapprox = r*np.sin(t) + yc

pt.plot(x,y,'o')
pt.plot(xapprox,yapprox,'r')
pt.grid(True)
pt.axes().set_aspect('equal','datalim')
```



In [98]:

```
def g(x, y, xc, yc, r):
    return (x-xc)**2 + (y-yc)**2 - r**2

def F(x, y, xc, yc, r):
    return np.sum(g(x, y, xc, yc, r)**2)

def gradF(x, y, xc, yc, r):
    gg = g(x,y,xc,yc,r)
    F1 = -4 * np.sum(gg*(x-xc))
    F2 = -4 * np.sum(gg*(y-yc))
    F3 = -4 * np.sum(gg*r)
    return np.array([F1,F2,F3])

def HessF(x,y,x0,y0,r):
    gg = g(x,y,xc,yc,r)
    g11 = np.sum(4*gg + 8*(x-xc)**2)
    g12 = np.sum(8*(x-xc)*(y-yc))
    g13 = np.sum(8*(x-xc)*r)
    g22 = np.sum(4*gg + 8*(y-yc)**2)
    g23 = np.sum(8*(y-yc)*r)
    g33 = np.sum(-4*gg + 8*r**2)
    H = np.array([[g11, g12, g13],[g12,g22,g23],[g13,g23,g33]])
    return H
```

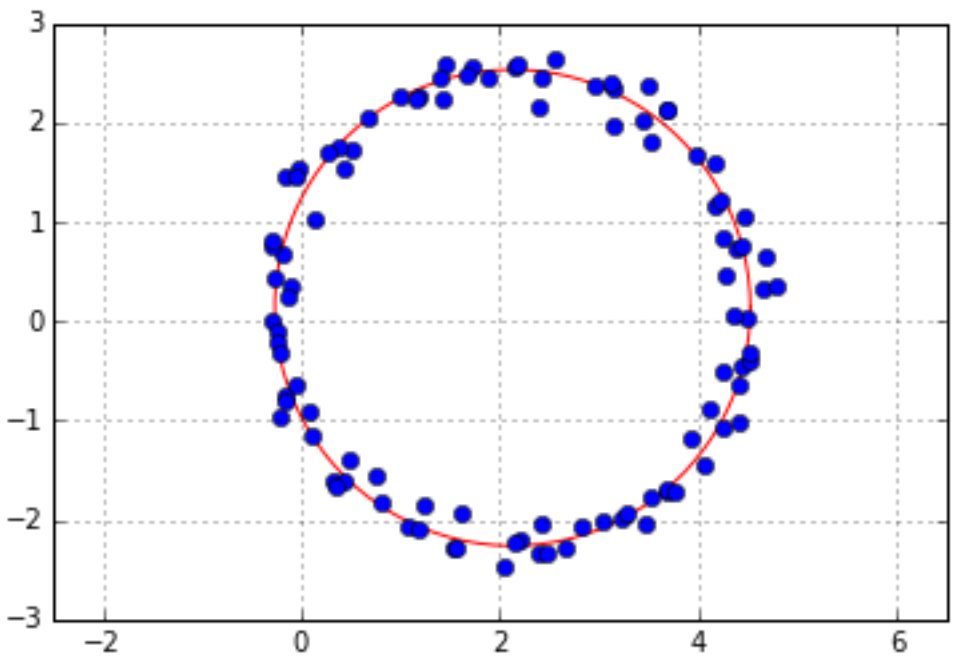
In [114]:

```
xc = 2.0
yc = 1.0
r = 5.5

xx = np.array([xc,yc,r])# initial guess for a,b, and r
tol = 1e-15

k = 0
```

In [134]:

```
xc = xx[0]
yc = xx[1]
r = xx[2]
HF = HessF(x,y,xc,yc,r)    #compute current Hessian
gF = -gradF(x,y,xc,yc,r)   #compute current gradient
step = la.solve(HF,gF)     #solve H*step = g
xx = xx + step             #next iterate
xapprox = xx[2]*np.cos(t) + xx[0]
yapprox = xx[2]*np.sin(t) + xx[1]
pt.plot(xapprox,yapprox,'r')
pt.plot(x,y,'o')
pt.grid(True)
pt.axes().set_aspect('equal','datalim')

print(k, gradF(x, y, xc, yc, r))
```

```
0 [ -2.80664381e-13  -4.01706446e-13   7.26529947e-13]
```



In [ ]: