In [2]:
```
import numpy as np
import scipy.linalg as la
```

Lab 9

Key by Spencer Lyon

Problem 1

In [40]:
```
def qrDecomp(A):
    """
    Use the modified Grahm-Schmidt algorithm to compute the QR
    decomposition of a matrix.

    Parameters
    ----------
    A : array_like, dtype=float
        The matrix you want to decompose

    Returns
    -------
    R : array_like, dtype=float
        The Upper triangular matrix R from the decomposition

    Q : array_like, dtype=float
        The orthonormal matrix Q from the decomposition

    """
    _, n = A.shape
    Q = np.zeros_like(A)
    R = np.zeros((n, n))

    # NOTE: the -1 below isn't part of the algorithm, but is needed to get
    #       the same solution as scipy.linalg.qr or MATLAB's qr routine.
    for jj in xrange(n):
        R[jj, jj] = la.norm(A[:, jj])
        Q[:, jj] = A[:, jj] / R[jj, jj]
        for kk in xrange(jj, n):
            R[jj, kk] = A[:, kk].dot(Q[:, jj])
            A[:, kk] -= R[jj, kk] * Q[:, jj]

    return [Q, R]
```

In [41]:
```
A_1 = np.random.randn(3, 3)
```

```
sp_Q, sp_R = la.qr(A_1)
Q, R = qrDecomp(A_1)

print("R's the same as scipy? %s" % np.allclose(R, sp_R))
print("Q's the same as scipy? %s" % np.allclose(Q, sp_Q))
```

```
R's the same as scipy? True
Q's the same as scipy? True
```

Lab 10

Key by Spencer Lyon

Problem 1

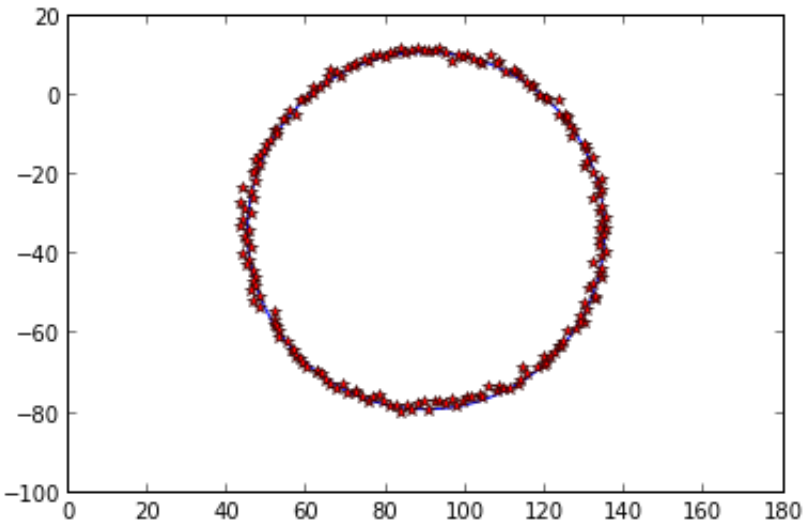
In [71]:
```
def fit_circle(x, y, plot=False):
    """
    Given vectors representing x and y data, find the coefficients
    in the equation for a circle for the best fit of that data. The
    equation for a circle is given below:

    .. math::
        (x - c_1)^2 + (y - c_2)^2 = r^2

    This equation can be expanded to a more useful form:

    .. math::
        2c_1x + 2c_2y +c_3 = x^2 + y^2

    where :math:`c_3 = r^2 - c_1^2 - c_2^2`. This can be written as a
    linear system :math:`Ax=b`:

    Parameters
    ----------
    x, y : array_like, dtype=float
        The x and y coordinates representing a circle.

    plot : bool, optional(default=False)
        A boolean value specifying whether or not to plot the solution

    Returns
    -------
    c_1, c_2, r : float
        The coefficients c1, c2, and the radius in the circle equation

    """
    A = np.column_stack([2 * x, 2 * y, np.ones_like(x)])
```

```
    b = x**2 + y**2
    c1, c2, c3 = la.lstsq(A, b)[0]
    r = np.sqrt(c1**2 + c2**2 + c3)

    if plot:
        plt.ion()
        plt.figure()
        theta = np.linspace(0, np.pi * 2, 200)
        plt.plot(r * np.cos(theta) + c1, r * np.sin(theta) + c2, '-',
                 x, y, 'r*')
        plt.axis('equal')  # So it looks like a circle
        plt.draw()
        plt.ioff()

    return c1, c2, r
```

In [72]:
```
data = np.genfromtxt('../Data/lab10.txt')
```

In [73]:
```
fit_circle(data[:, 0], data[:, 1], 1)
```

Out[73]: (89.891325666897671, -34.082185502270583, 44.900607648638555)



TODO Finish problem 2

Lab 11

Key by Ryan Brunt

Problem 1

In [154]:
```
import numpy as np
import scipy.linalg as la


def householder(A):
    A = np.asarray(A)
    m, n = A.shape

    R = A.copy()
    Q = np.eye(n)

    for k in xrange(n - 1):
        x = R[k:, k]
        e1 = np.zeros_like(x)
        e1[0] = 1.
        vk = np.sign(x[0]) * la.norm(x) * e1 + x
        vk /= la.norm(vk)
        Pk = np.eye(m)
        Pk[k:, k:] -= 2 * np.outer(vk, vk)
        R = Pk.dot(R)

        if k == 0:
            Q = Pk
        else:
            Q = Pk.dot(Q)

    # Note that we used Pk in such a way to compute Q transpose
    return Q.T, R
```

In [155]:
```
A = np.random.randn(4, 4)
Q, R = householder(A)
sp_Q, sp_R = qr(A)

print("R's the same as scipy? %s" % np.allclose(R, sp_R))
print("Q's the same as scipy? %s" % np.allclose(Q, sp_Q))
```

```
R's the same as scipy? True
Q's the same as scipy? True
```

Problem 2

In [2]:
```
import scipy as sp
from scipy import linalg as la
```

```
import math


def Hessenberg(A):
    """
    Return the upper Hessenberg form of a matrix

    Parameters
    ----------
    A : numpy.ndarray
        n by n matrix to be put in upper Hessenberg form
        Note A must be square

    Returns
    -------
    H, Q : numpy.ndarray
        H is the upper Hessenberg form of the input matrix
        Q is the orthogonal transition matrix

    Notes
    -----
    A = Q.transpose().H.Q

    """
    H =  A
    m = sp.shape(H)[0]
    n = sp.shape(H)[1]
    Q = sp.eye(m,m)
    for k in range(n-1):
        x = H[k+1:m,k]
        e2 = sp.zeros((1,sp.shape(x)[0]))
        e2[0,0] = 1
        v_k = math.copysign(1,x[0])*la.norm(x,2)*e2 + x
        v_k = v_k/la.norm(v_k,2)
        P_k = sp.eye(m,m)
        P_k[k+1:m,k+1:m] = P_k[k+1:m,k+1:m] - 2*sp.outer(v_k,sp.transpose(
        Q = sp.dot(P_k,Q)
        H = sp.dot(P_k,H)
        H = sp.dot(H,sp.transpose(P_k))
    return H,Q

print 'Problem 2'
G = sp.array([[4,3,1,3],[5,2,4,7],[2,3,8,5],[2,8,1,2]])
H = Hessenberg(G)[0]
Q = Hessenberg(G)[1]
print H, 'This is the upper Hessenberg form of the matrix'
Temp = sp.dot(sp.transpose(Q),H)
print sp.dot(Temp,Q), 'This is Q^T.H.Q'
print G, 'This is the original matrix'
```

```
Problem 2
[[  4.00000000e+00  -4.00378609e+00  -1.11218814e+00  -1.31633373e+00]
 [ -5.74456265e+00   1.01212121e+01   6.67576201e+00   7.35337603e-01]
```

```
 [ -6.43018752e-16   6.60696407e+00  -8.45490052e-01  -1.64863116e+00]
 [  7.98724773e-17   1.33226763e-15  -5.82649490e+00   2.72427793e+00]]
This is the upper Hessenberg form of the matrix
[[ 4.  3.  1.  3.]
 [ 5.  2.  4.  7.]
 [ 2.  3.  8.  5.]
 [ 2.  8.  1.  2.]] This is Q^T.H.Q
[[4 3 1 3]
 [5 2 4 7]
 [2 3 8 5]
 [2 8 1 2]] This is the original matrix
```

Lab 15

Key by Ryan Brunt

In [13]:
```
la.qr(la.hessenberg(A))
```

Out[13]:
```
(array([[ 0.15617376, -0.9139952 , -0.3435377 , -0.14901132],
        [-0.9877296 , -0.14451533, -0.05431808, -0.02356076],
        [ 0.        , -0.37911488,  0.84892898,  0.3682275 ],
        [ 0.        ,  0.        , -0.39793338,  0.91741432]]),
 array([[ 6.40312424, -6.3708559 ,  3.4369337 , -4.72044582],
        [ 0.        ,  8.94495361, -3.46630803,  1.91297395],
        [-0.        ,  0.        ,  9.15603981, -0.45907198],
        [-0.        , -0.        ,  0.        ,  5.15619097]]))
```

In [17]:
```
"""
lab15key.py - written by Ryan Brunt

"""
from __future__ import division
import numpy as np
import scipy as sp
from scipy import linalg as la
import sympy
from sympy import Matrix


def qrEig(a, its = 1000):
    """
    Returns eigenvalues of a matrix via QR algorithm

    Parameters
    ----------
    a : numpy.ndarray
        n x n input matrix
    its : integer
```

```
        number of iterations for the QR algorithm to execute

    Returns
    -------
    eVals : numpy.ndarray
        vector of eigenvalues

    """
    if sp.shape(a)[0] != sp.shape(a)[1]:
        print 'ERROR: input matrix is not square'
    else:
        h = la.hessenberg(a)
        b = Matrix(h)
        n = its
        while n>=0:
            q,r = la.qr(h)
            h_new = sp.dot(r,q)
            b = Matrix(h_new)
            if np.allclose(h, h_new):
                break
            h = h_new.copy()
            n -= 1
        offDiag = sp.diag(h,-1)
        eVals = []
        if not(b.is_upper()):
            matI = sp.eye(sp.shape(h)[0])
            for i in range(sp.shape(offDiag)[0]):
                if offDiag[i] >= 1e-250:
                    subMatrix = h[i:i+2,i:i+2]
                    eVals.append(sp.roots([1,-sp.trace(subMatrix),la.det(s
                    eVals.append(sp.roots([1,-sp.trace(subMatrix),la.det(s
                    matI[i,:] *= 0
                    matI[i+1,:] *= 0
            for k in sp.diag(h*matI):
                if k != 0:
                    eVals.append(k)
        else:
            eVals = sp.diag(h)
        eVals = sp.array(eVals).astype(complex)
        return eVals

n = 4
A = sp.random.randn(n, n)       #random real
B = sp.random.randn(n, n)*1j
C = A + B                                   #random complex
D = A + A.T - sp.diagflat(sp.diag(A))   #random symmetric
H = C + C.T.conj()                          #random Hermitian

print 'This tests a random real matrix'
print qrEig(A, 5)
print 'This uses the built in routine'
print sp.sort(la.eig(A)[0])
```

```
print 'This tests a random complex matrix'
print qrEig(C)
print 'This uses the built in routine'
print sp.sort(la.eig(C)[0])
print 'This tests a real symmetric matrix'
print qrEig(D)
print 'This uses the built in routine'
print sp.sort(la.eig(D)[0])
print 'This tests a random hermitian matrix'
print qrEig(H)
print 'This uses the built in routine'
print sp.sort(la.eig(H)[0])
```

```
This tests a random real matrix
[-1.11180092+2.02697677j -1.11180092-2.02697677j -1.52082338+0.j
 -0.50863679+0.j         -0.56474000+0.17421139j -0.56474000-0.17421139j]
This uses the built in routine
[-1.11148588-2.02695487j -1.11148588+2.02695487j -0.56505505-0.1742988j
 -0.56505505+0.1742988j ]
This tests a random complex matrix
[-0.37050175+2.97526822j -2.02643569-1.44828675j -2.02643569-1.44828675j
 -0.77730929-1.81915212j -0.17883511+0.53819926j]
This uses the built in routine
[-2.02643569-1.44828675j -0.77730929-1.81915212j -0.37050175+2.97526822j
 -0.17883511+0.53819926j]
This tests a real symmetric matrix
[-3.13894012+0.j -1.44994086+0.j  0.94664471+0.j  0.28915442+0.j]
This uses the built in routine
[-3.13894012+0.j -1.44994086+0.j  0.28915442+0.j  0.94664471+0.j]
This tests a random hermitian matrix
[-5.99071932 -2.75376198e-16j -1.58081320 +1.42618090e-17j
  1.17537511 -2.08282521e-16j -0.31000628 +2.01919608e-16j]
This uses the built in routine
[-5.99071932 -2.75376198e-16j -1.58081320 +1.42618097e-17j
 -0.31000628 +5.30632755e-17j  1.17537511 -2.08282521e-16j]
```

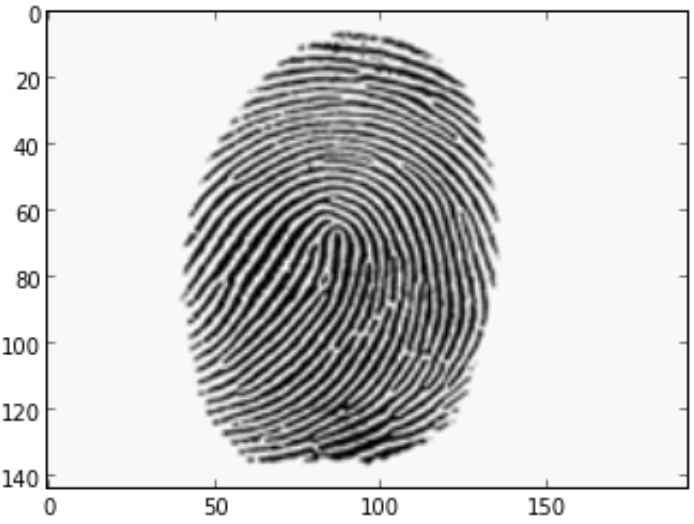
Lab 16

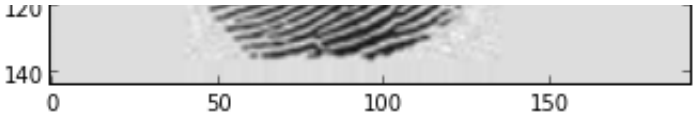
Key by Ryan Brunt

In [3]:
```
"""
lab16key.py - written by Ryan Brunt

"""

import scipy as sp
import matplotlib.pyplot as plt
from scipy import linalg as la
```

```
X = sp.misc.imread('fingerprint.png')[:,:,0].astype(float)
X.nbytes

plt.figure()
plt.gray()
plt.imshow(X)

U,s,Vt = la.svd(X)
S = sp.diag(s)

n = 50
u1, s1, vt1 = U[:,0:n], S[0:n,0:n], Vt[0:n,:]
Xhat = sp.dot(sp.dot(u1,s1),vt1)
(u1.nbytes+sp.diag(s1).nbytes+vt1.nbytes) - X.nbytes #should be negative
y = float((u1.nbytes+sp.diag(s1).nbytes+vt1.nbytes))
print y,'This is how many bytes it would take for each image'
print 'You only use ', y/X.nbytes*100, ' percent of the original memory ne
plt.figure(2)
plt.gray()
plt.imshow(Xhat)
plt.show()
```

```
135200.0 This is how many bytes it would take for each image
You only use  60.8088658607  percent of the original memory needed for
the uncompressed image
```





In [ ]: