lista2

Lucas Nogueira Ribeiro

October 13, 2015

1 Exercise 1

Consider a Bernoulli random variable $x \in \{0, 1\}$ parametrized in μ. Its probability density function is given by $p(x|\mu) = \mu^x(1-\mu)^{1-x}$. * The mean value of x is given by $\mathbb{E}[x] = \sum_{x\in\{0,1\}} xp(x|\mu) = 0.(1-\mu) + 1.\mu = \mu$ * The variance of x is given by $var[x] = \mathbb{E}[x^2] - \mathbb{E}[x]^2 = \sum_{x\in\{0,1\}} xp(x|\mu) = 0^2.(1-\mu) + 1^2.\mu - \mu^2 = \mu(1-\mu)$

2 Exercise 2

2.1 Items a-d

Estimate mean vector and covariance matrix

In [20]:
```
%matplotlib inline
%config InlineBackend.figure_format = 'svg'

import pylab as pl
import numpy as np

# Load data
X = np.loadtxt('02_Assignment_data.dat')
nsamples, nvars = X.shape

# Calculate mean vector
mean_1 = X[:,0].sum()/nsamples
mean_2 = X[:,1].sum()/nsamples
mean_3 = X[:,2].sum()/nsamples

mX = np.array([mean_1, mean_2, mean_3])

# replicate mean vector
MX = np.tile(mX, (nsamples, 1))

# Estimate covariance matrix
Cx = (1.0/nsamples)*np.dot((X-MX).transpose(),(X-MX))
```

Show covariance matrix and variance of variables x_1, x_2, and x_3

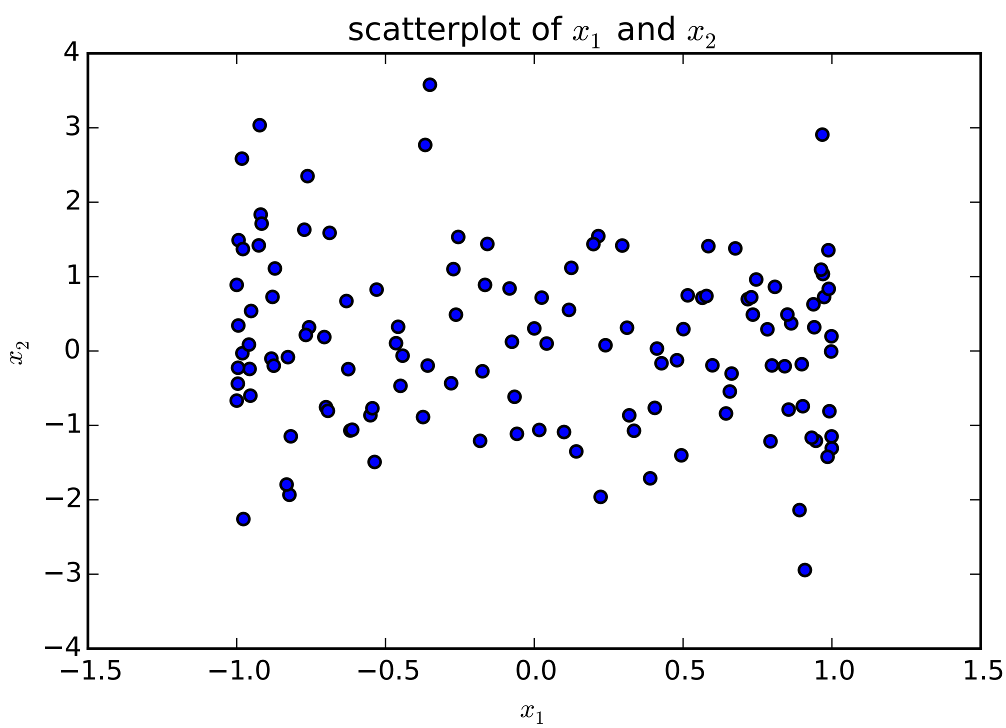
In [21]:
```
print 'var(x_1): %f' % np.var(X[:,0])
print 'var(x_2): %f' % np.var(X[:,1])
print 'var(x_3): %f' % np.var(X[:,2])
print
print 'Covariance matrix:'
print Cx
```

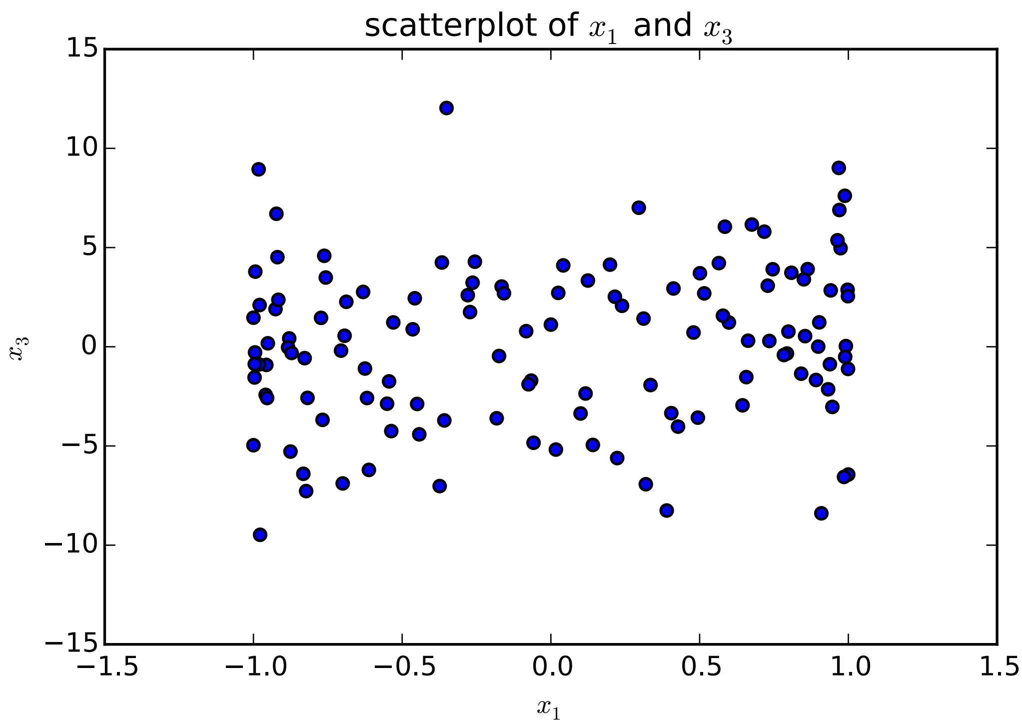
```
var(x_1): 0.498666
var(x_2): 1.339572
var(x_3): 16.131340

Covariance matrix:
[[  0.49866646  -0.10245845   0.4109965 ]
 [ -0.10245845   1.33957189   3.89705712]
 [  0.4109965    3.89705712  16.13133976]]
```

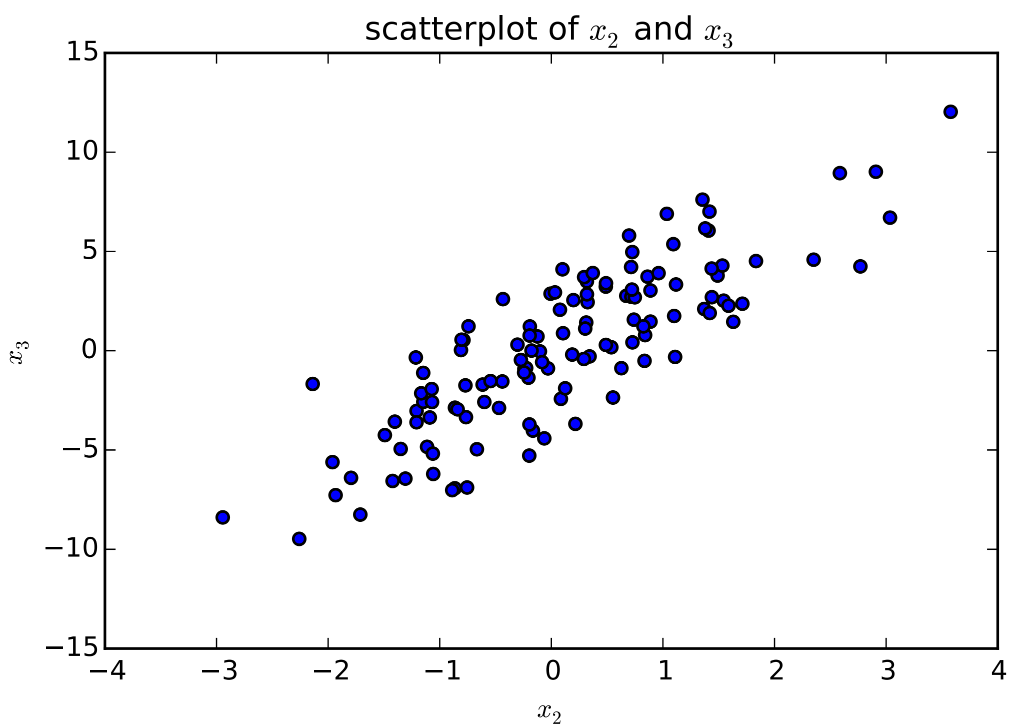
Show Scatterplot diagrams

In [22]:
```
pl.figure(0)
pl.scatter(X[:,0], X[:,1])
pl.xlabel('$x_1$')
pl.ylabel('$x_2$')
pl.title('scatterplot of $x_1$ and $x_2$')

pl.figure(1)
pl.scatter(X[:,0], X[:,2])
pl.xlabel('$x_1$')
pl.ylabel('$x_3$')
pl.title('scatterplot of $x_1$ and $x_3$')

pl.figure(2)
pl.scatter(X[:,1], X[:,2])
pl.xlabel('$x_2$')
pl.ylabel('$x_3$')
pl.title('scatterplot of $x_2$ and $x_3$')
```

Out [22]:
```
<matplotlib.text.Text at 0x7f8217892410>
```

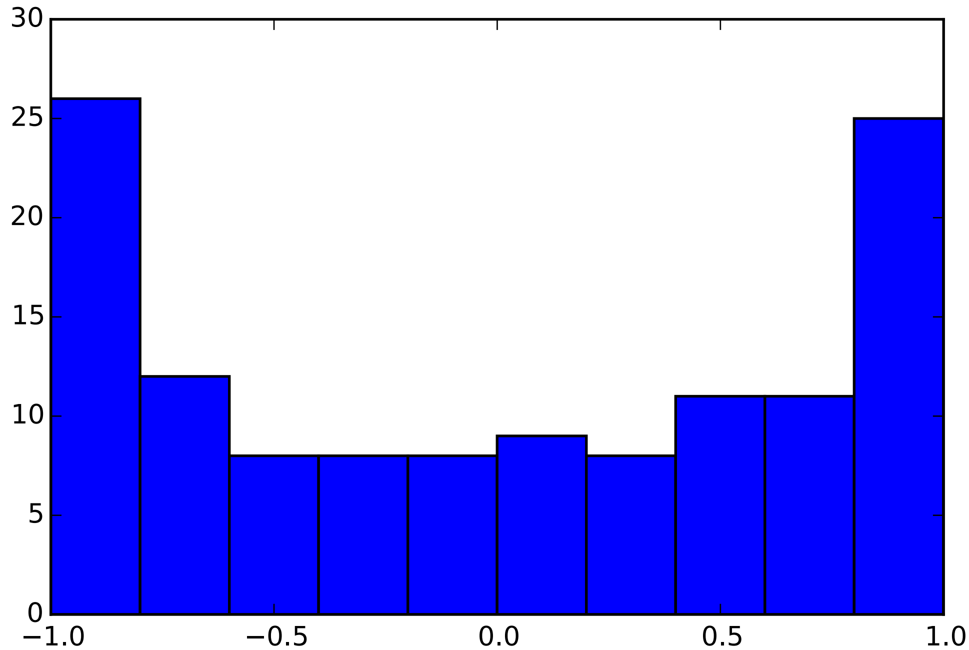


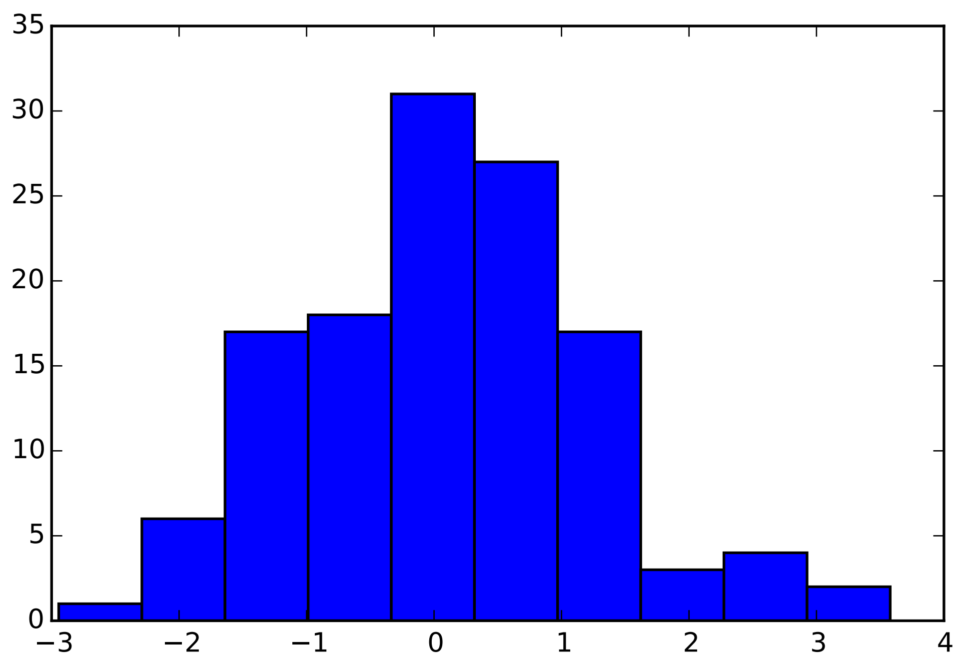

scatterplot of x_1 and x_3
15
10
5
0
−5
−10
−15
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
x_1
x_3

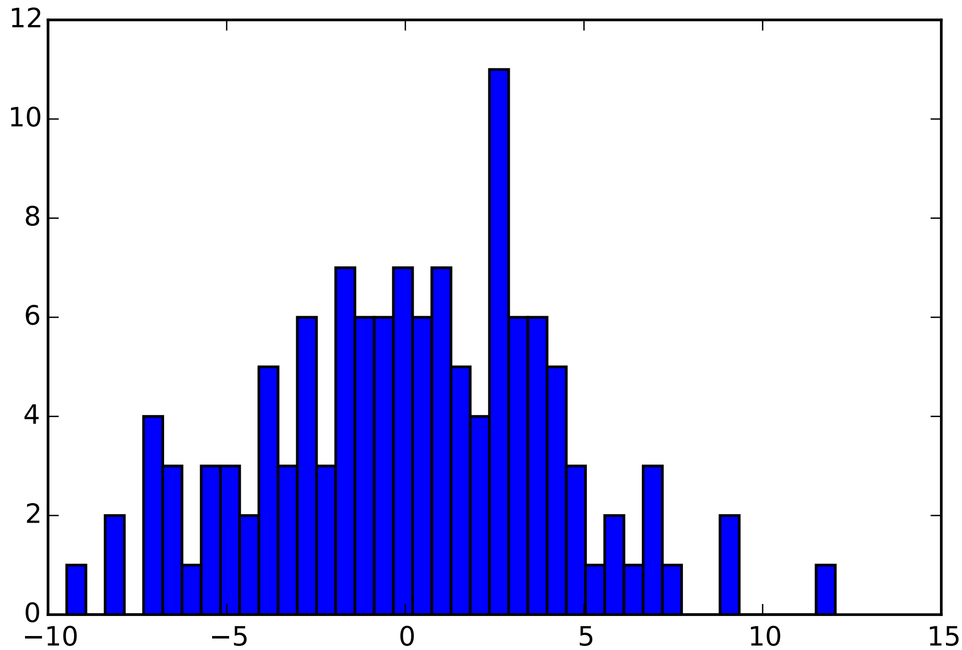

scatterplot of x_2 and x_3
15
10
5
0
−5
−10
−15
−4
−3
−2
−1
0
1
2
3
4
x_2
x_3

2.2 Item e

In [23]:
```
pl.figure()
n1, bins1, patches1 = pl.hist(X[:,0])

pl.figure()
n2, bins2, patches2 = pl.hist(X[:,1])

pl.figure()
n3, bins3, patches3 = pl.hist(X[:,2],bins=40)
```







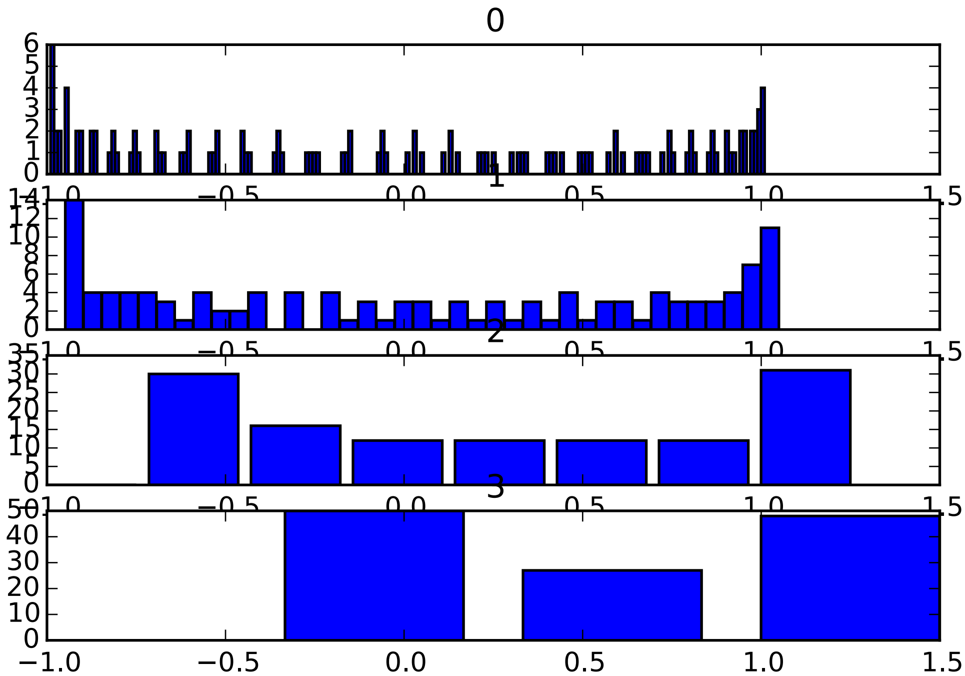
3 Exercise 3

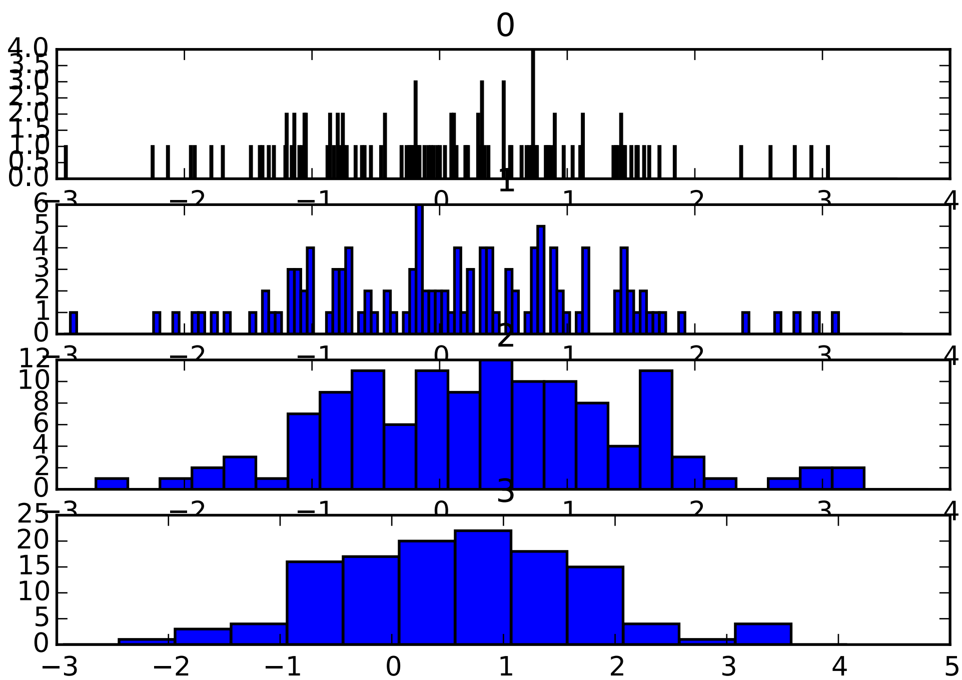
Consider our histogram implementation:

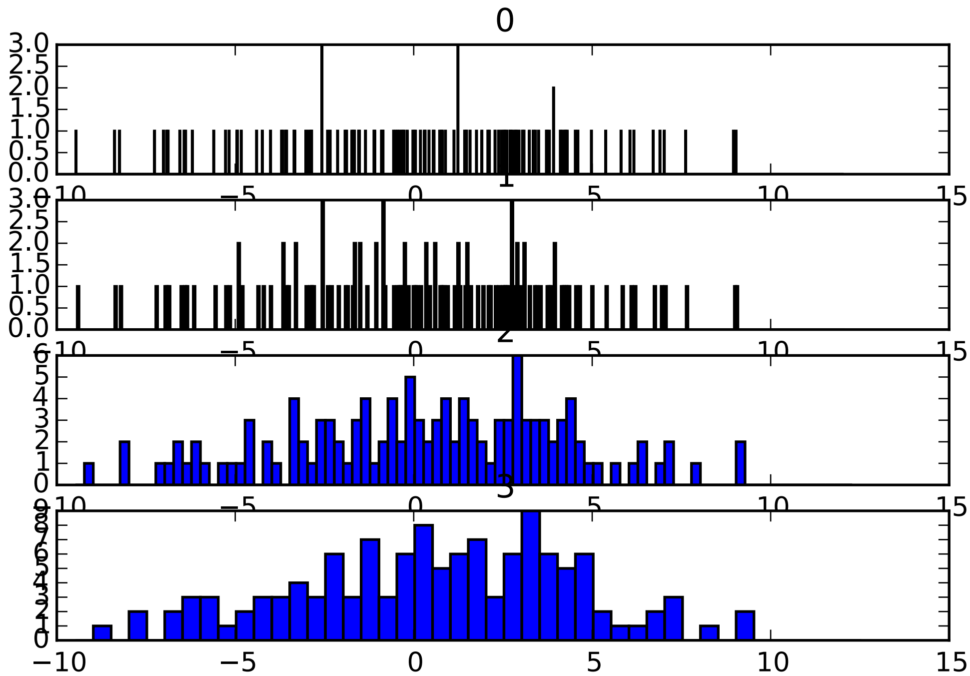
In [24]:
```
def myhistogram(X, width):
    data = X.flatten() # transform multidimensional array into a vector
    nbins = np.ceil((data.max() - data.min())/width)
    bins = np.linspace(data.min(), data.max(), nbins)

    hist = np.zeros((nbins, 1))
    for datum in data:
        k = 1

        while k < nbins:
            if datum >= bins[k-1] and datum < bins[k]:
                hist[k] += 1
            k += 1
    return bins, hist

def plotHistograms(data, widths):
    pl.figure()
    for k in range(len(widths)):
        bins, hist = myhistogram(data, widths[k])
        pl.subplot(len(widths), 1, k+1)
        pl.bar(bins, hist, widths[k])
        pl.title(k)
```

Histograms

In [25]:
```
widths = [0.01, 0.05, 0.25, 0.5]

plotHistograms(X[:,0], widths)
plotHistograms(X[:,1], widths)
plotHistograms(X[:,2], widths)
```


0
1
2
3
−1.0
−0.5
0.0
0.5
1.0
1.5

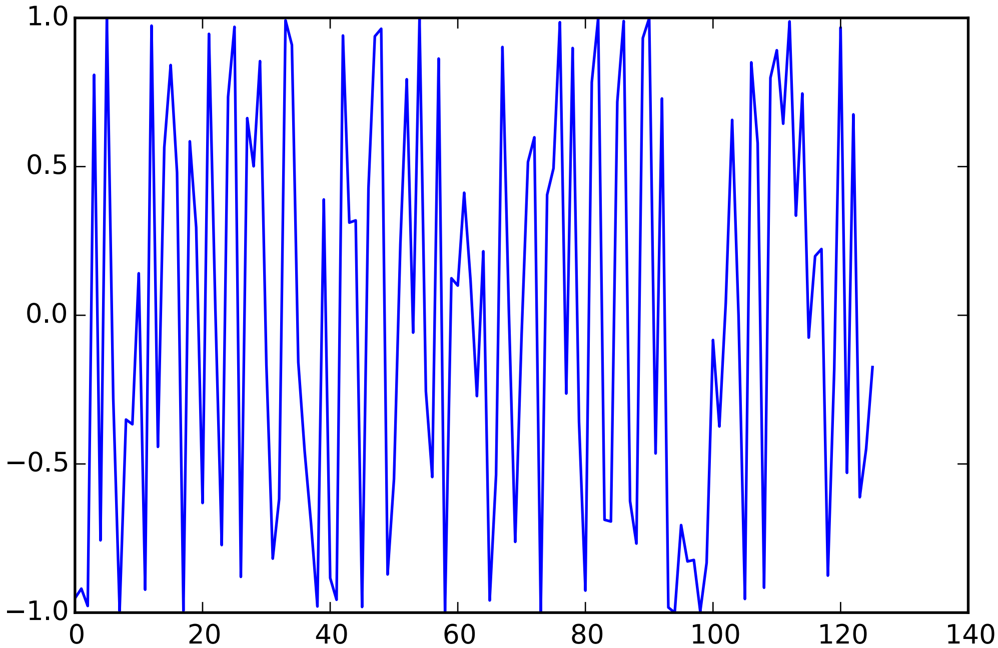

0
1
2
3
−3
−2
−1
0
1
2
3
4
5

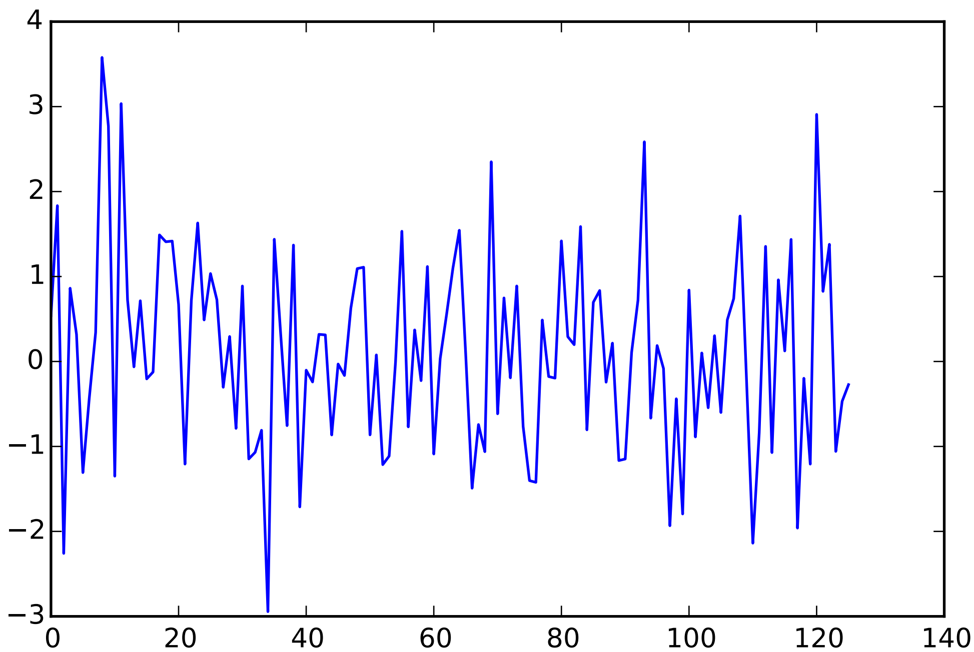


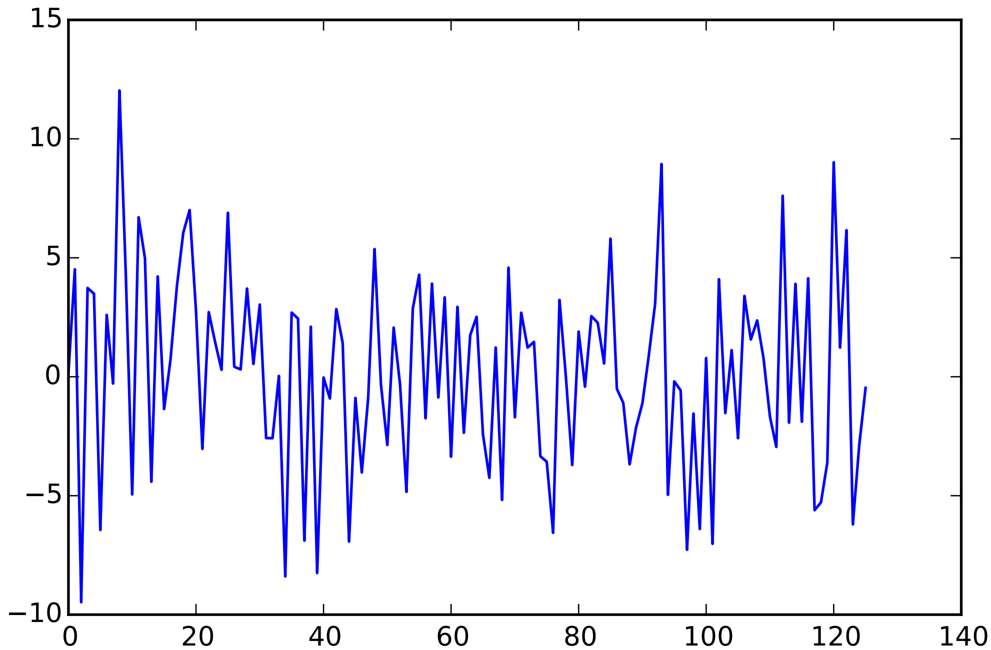
4 Exercise 4

In [26]:
```
pl.figure()
pl.plot(X[:,0])

pl.figure()
pl.plot(X[:,1])

pl.figure()
pl.plot(X[:,2])
```

Out [26]:
```
[<matplotlib.lines.Line2D at 0x7f8217278c50>]
```


1.0
0.5
0.0
−0.5
−1.0
0
20
40
60
80
100
120
140

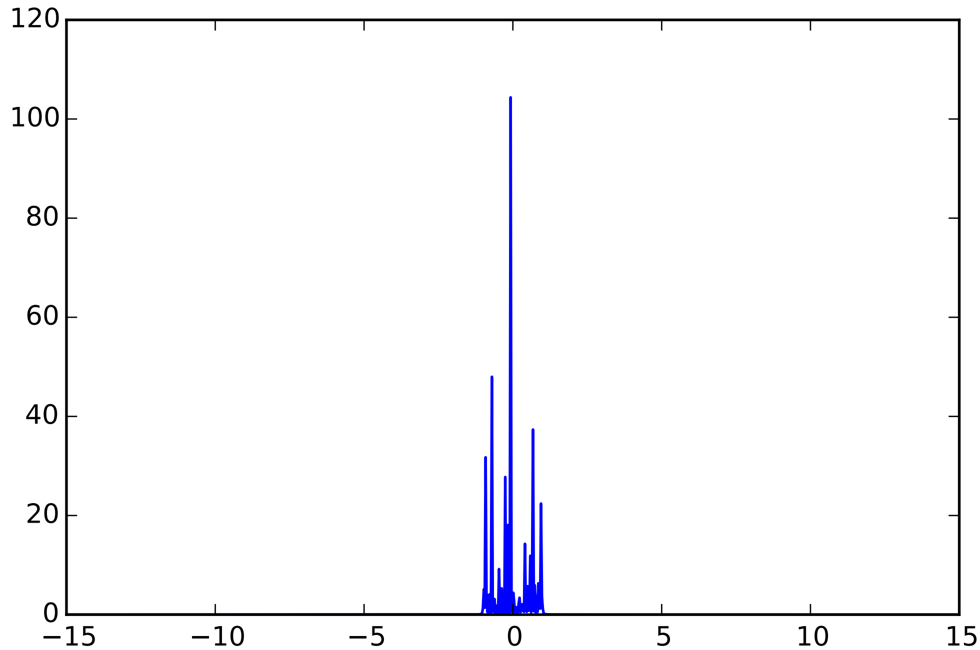

4
3
2
1
0
−1
−2
−3
0
20
40
60
80
100
120
140



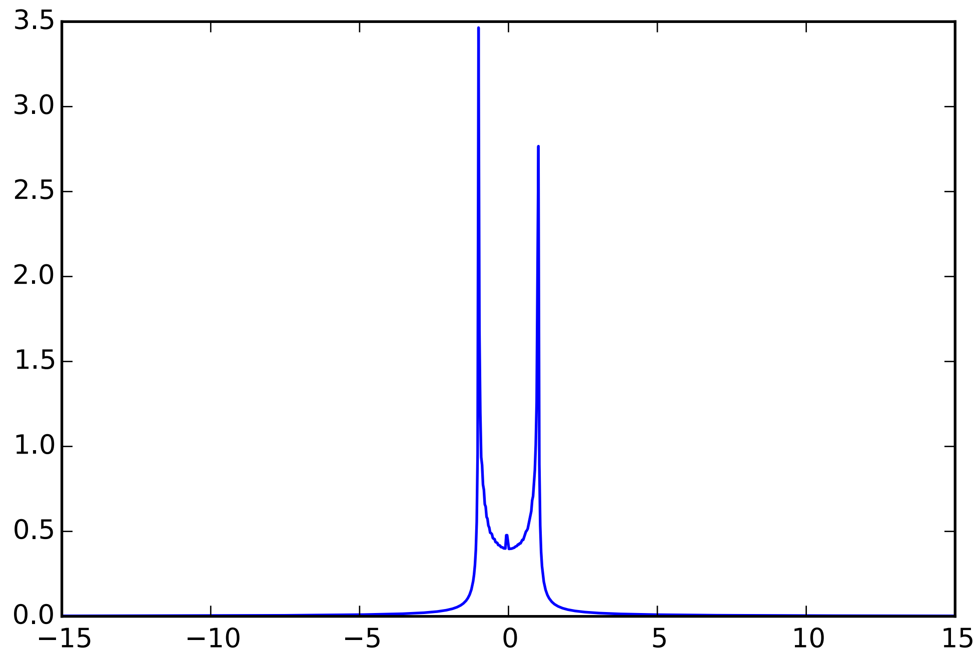
In [27]:
```
def getKNN(data, ref, K):
    ''' Calculates closest K neighbors of ref '''
    distance = np.abs(data - ref)
    KNN_idx = distance.argsort()[:K]
    return data[KNN_idx]

def pdfKNN(data, K, dlim = 1, npoints = 1000):
    ''' Estimates the prob density function of data using the kNN method '''
    N = len(data)
    prange = np.linspace(-dlim, dlim, npoints)
    p = np.zeros((npoints,))
    for i in range(npoints):
        ref = prange[i]
        neighbors = getKNN(data, ref, K)
        V = getVolume(data, neighbors, ref)
        p[i] = K/(N*V)
    return prange, p

def getVolume(v, neighbors, ref):
    ''' Calculates the volume that contais all neighbors and reference elements '''
    points = np.append(neighbors, ref)
    V = points.max() - points.min()
    return V
```

4.1 kNN density estimation for x_1

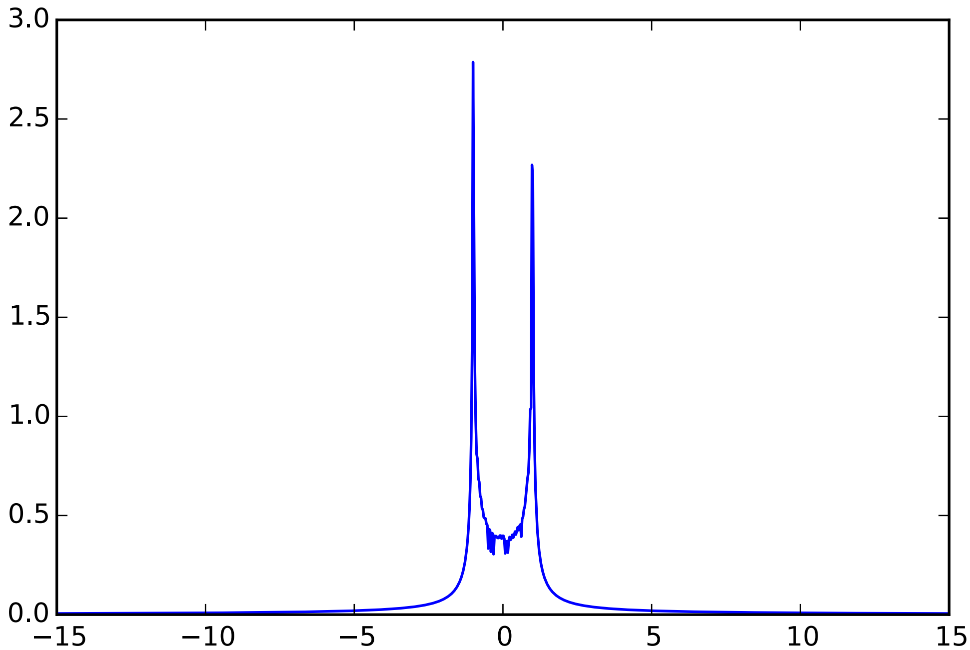
In [28]:
```
dlim = 15
for K in (1,5,10,15):
    pl.figure()
    prange, p = pdfKNN(X[:,0], K, dlim)
    pl.plot(prange, p)

    filename = './figures/knn_x1_{}.eps'.format(K)
    pl.savefig(filename, format='eps', dpi=1000)
```

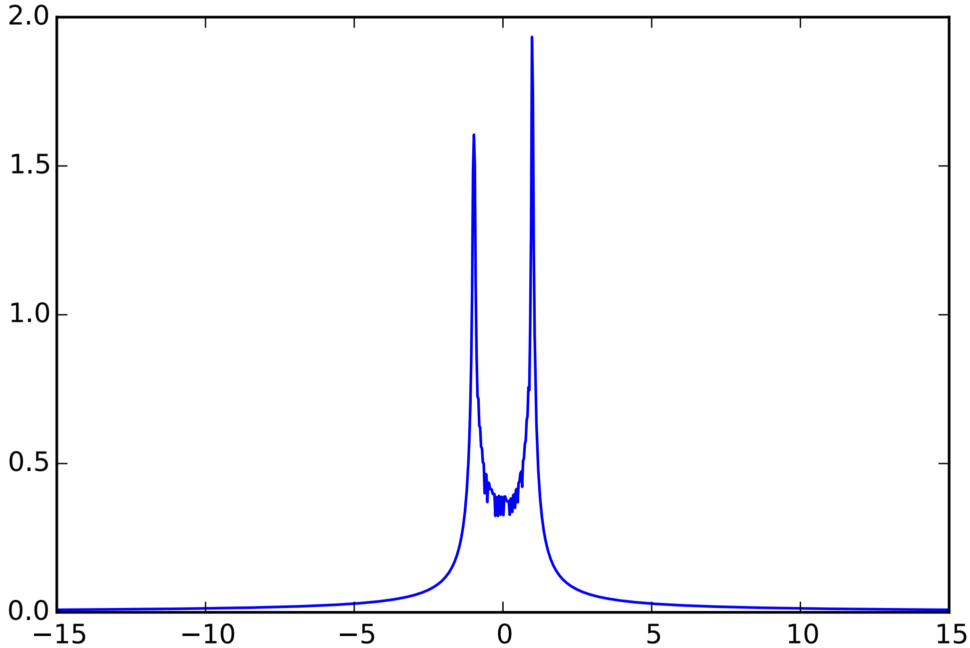


120
100
80
60
40
20
0
−15
−10
−5
0
5
10
15



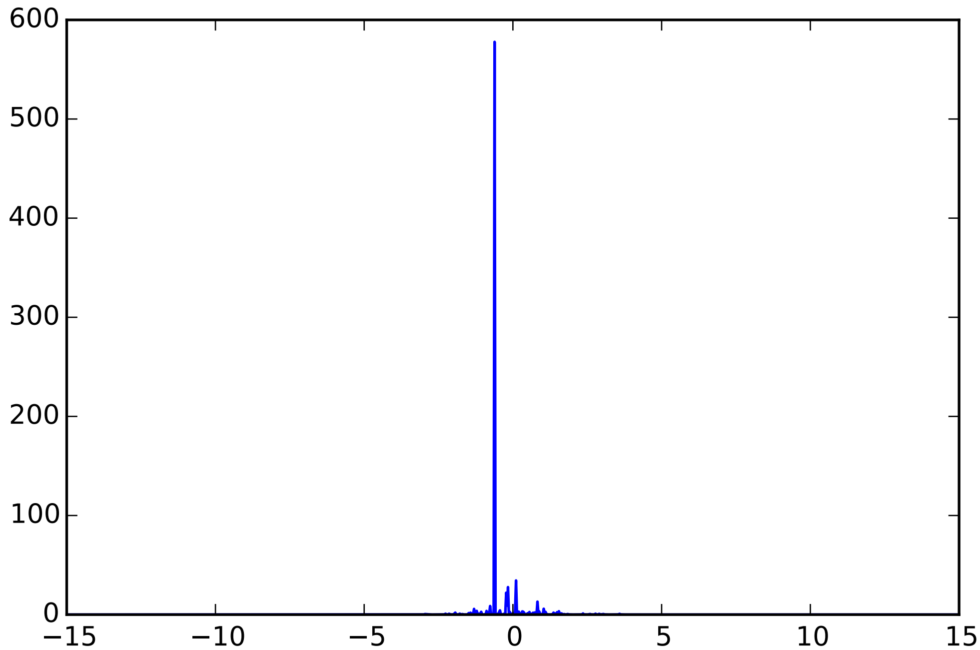
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−15
−10
−5
0
5
10
15

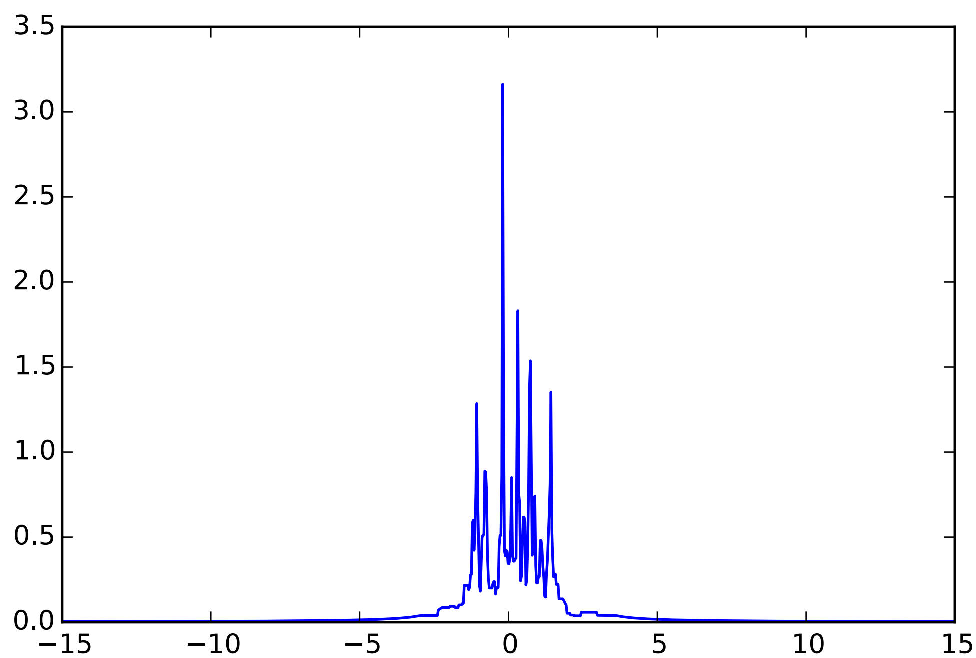


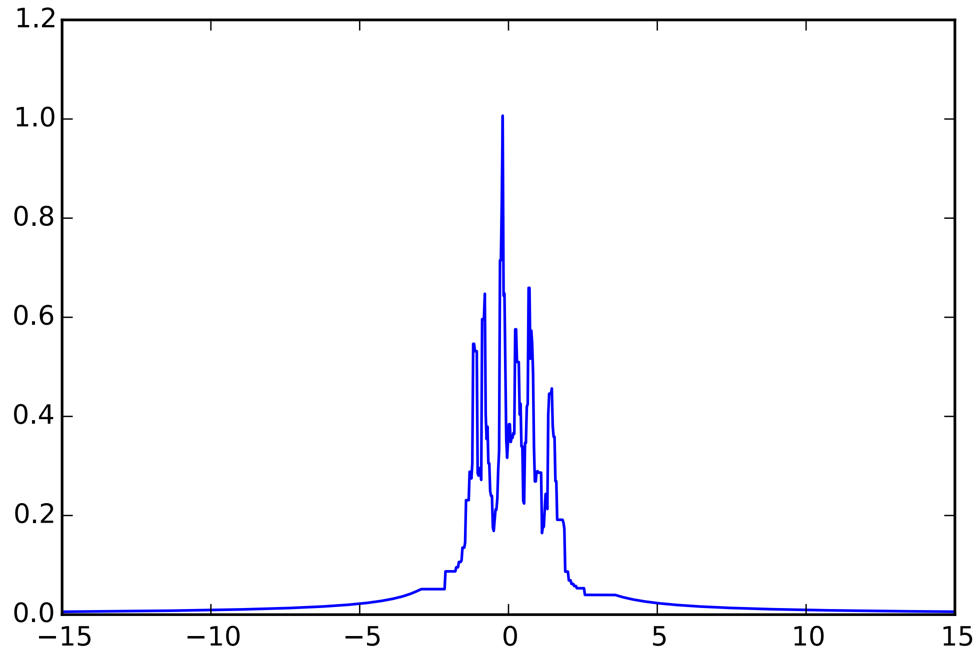


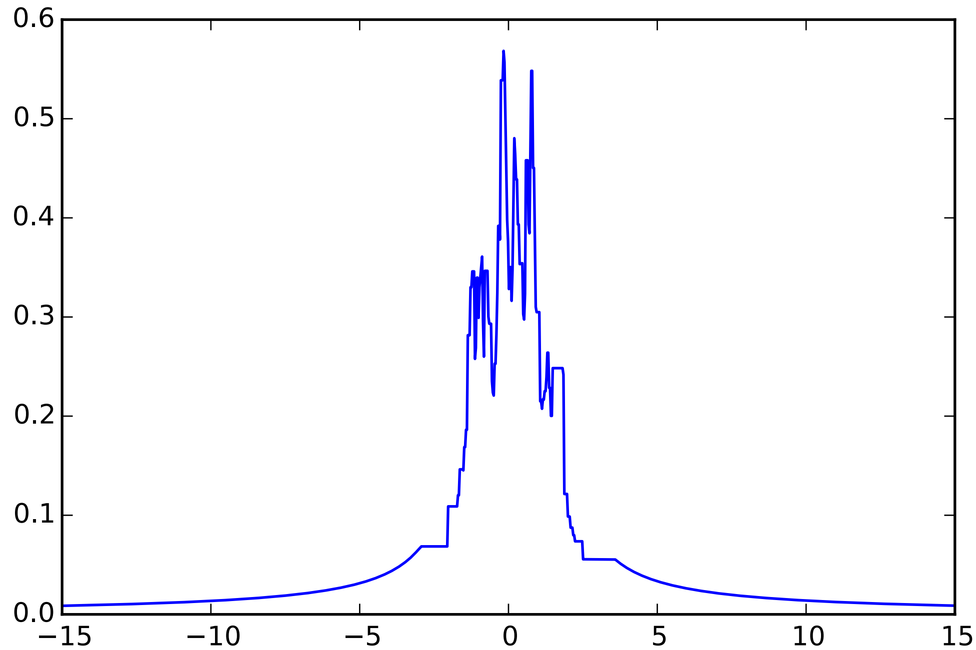
4.2 kNN density estimation for x_2

In [29]:
```
dlim = 15
for K in (1,5,10,15):
    pl.figure()
    prange, p = pdfKNN(X[:,1], K, dlim)
    pl.plot(prange, p)

    filename = './figures/knn_x2_{}.eps'.format(K)
    pl.savefig(filename, format='eps', dpi=1000)
```


600
500
400
300
200
100
0
−15
−10
−5
0
5
10
15

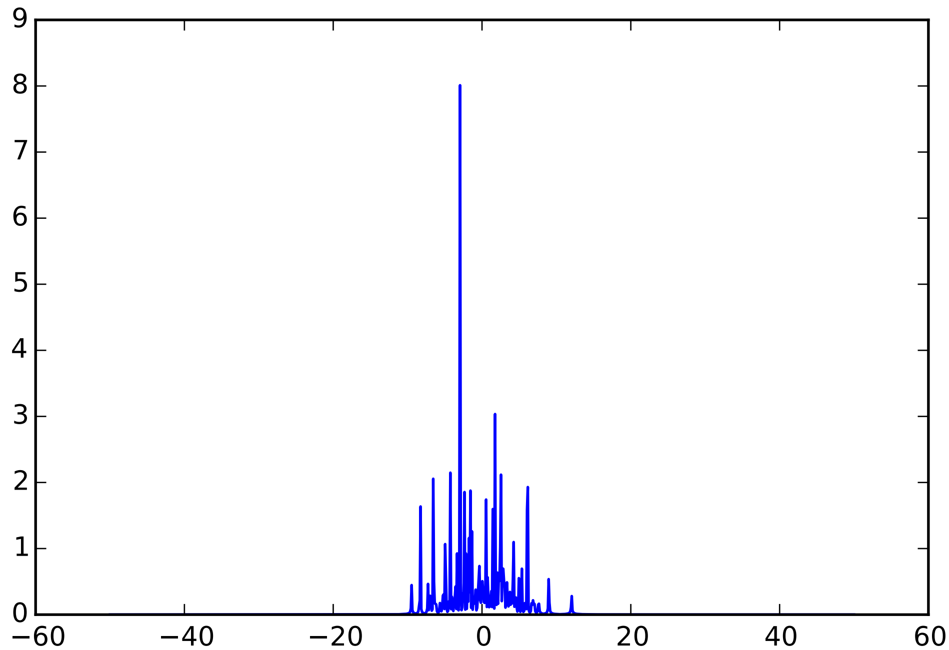

3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−15
−10
−5
0
5
10
15



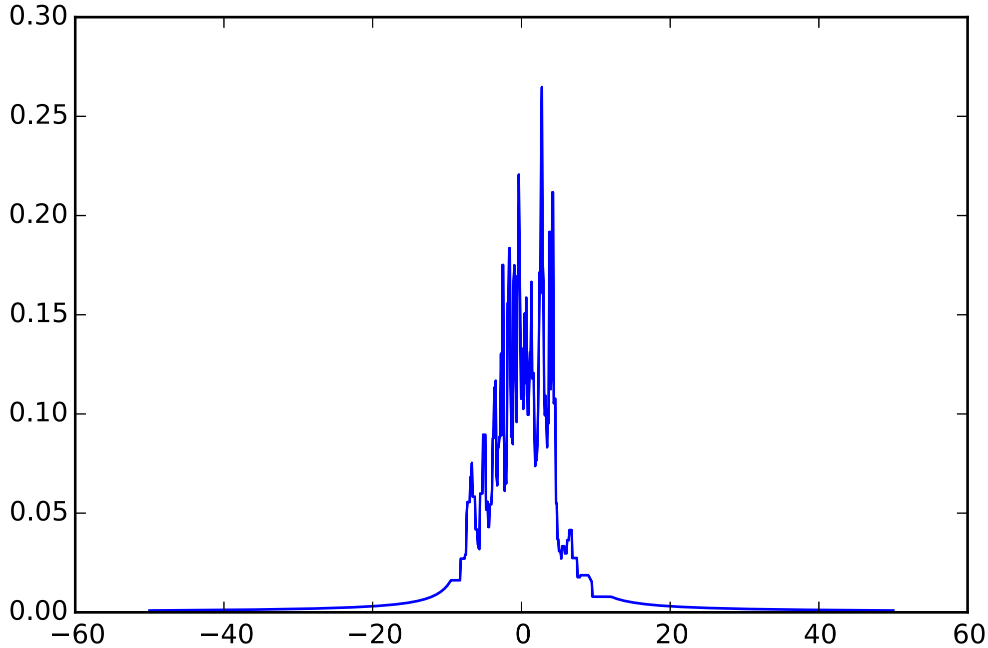


4.3 kNN density estimation for x_3

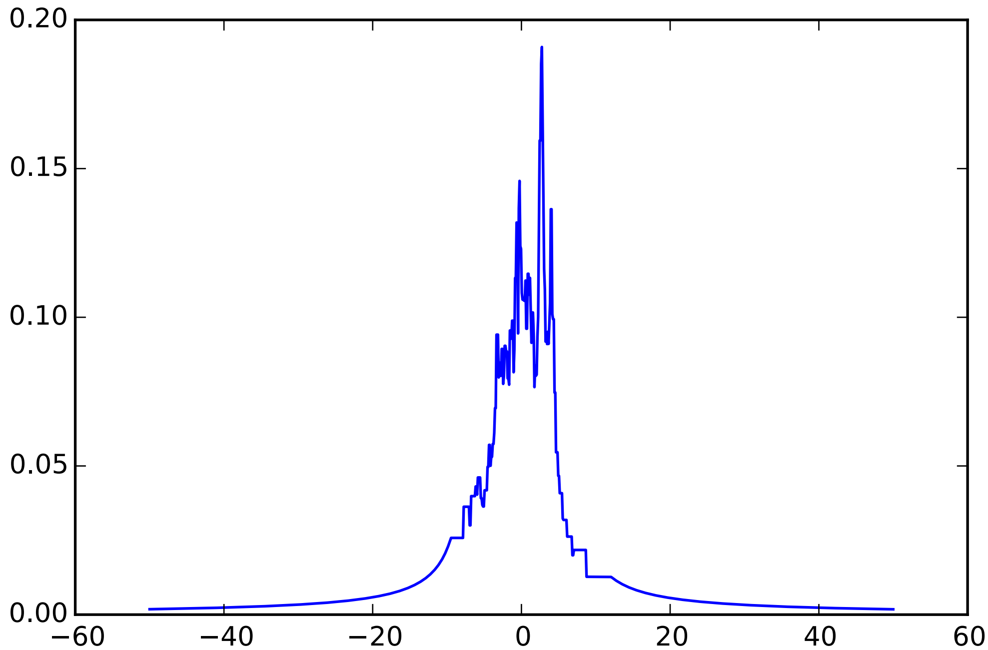
In [30]:
```
dlim = 50
for K in (1,5,10,15):
    pl.figure()
    prange, p = pdfKNN(X[:,2], K, dlim)
    pl.plot(prange, p)

    filename = './figures/knn_x3_{}.eps'.format(K)
    pl.savefig(filename, format='eps', dpi=1000)
```

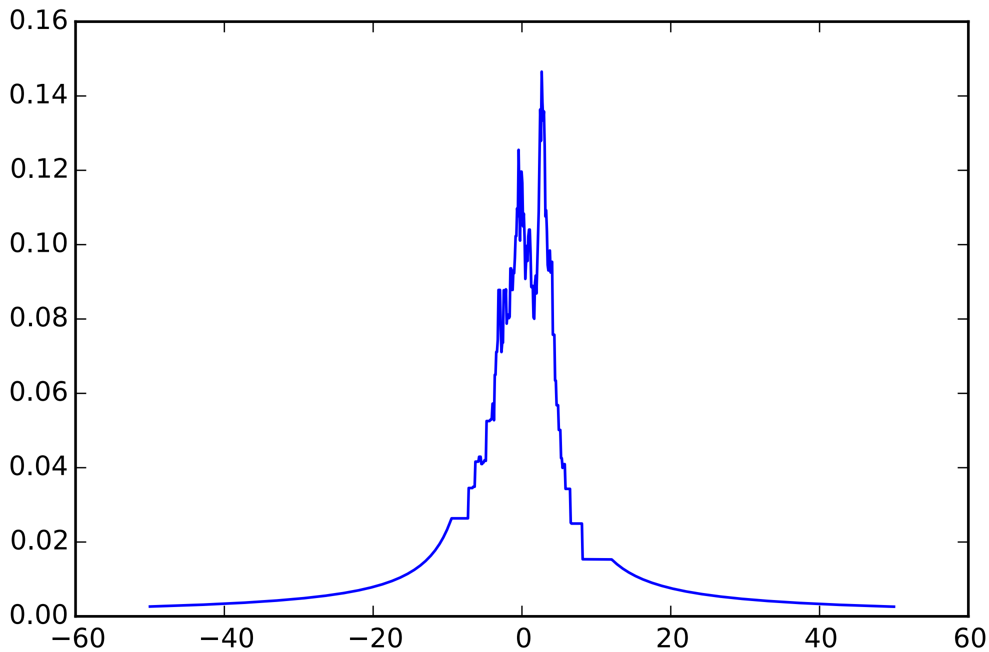


9
8
7
6
5
4
3
2
1
0
−60
−40
−20
0
20
40
60



0.30
0.25
0.20
0.15
0.10
0.05
0.00
−60
−40
−20
0
20
40
60





In [31]:
```
print X[:,0].max()
print X[:,1].max()
print X[:,2].max()
```

```
0.9995736
3.5783969
12.033746
```