Groupname: Gruppe 5

Arne Siebenmorgen	330529
Holger Prang	WM
Eugene Reese	319776
Jens Meiners	332697

In [3]:
```
%matplotlib inline
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
```

In [4]:
```
def pca(data, axis=0, center=True, normalize=False):
    mu = data.mean(axis=axis)
    if center:
        data = data - mu
    if normalize:
        data = (data - mu)/data.std(axis=axis)
    eigenvectors, eigenvalues, V = np.linalg.svd(data.T, full_matrices=False)
    return eigenvectors, eigenvalues

def project(data, eigenvectors, axis=0):
    projected_data = np.dot(data, eigenvectors)
    sigma = projected_data.std(axis=axis).mean()
    return projected_data, sigma
```

PCA as a tool for preprocessing and Kernel PCA

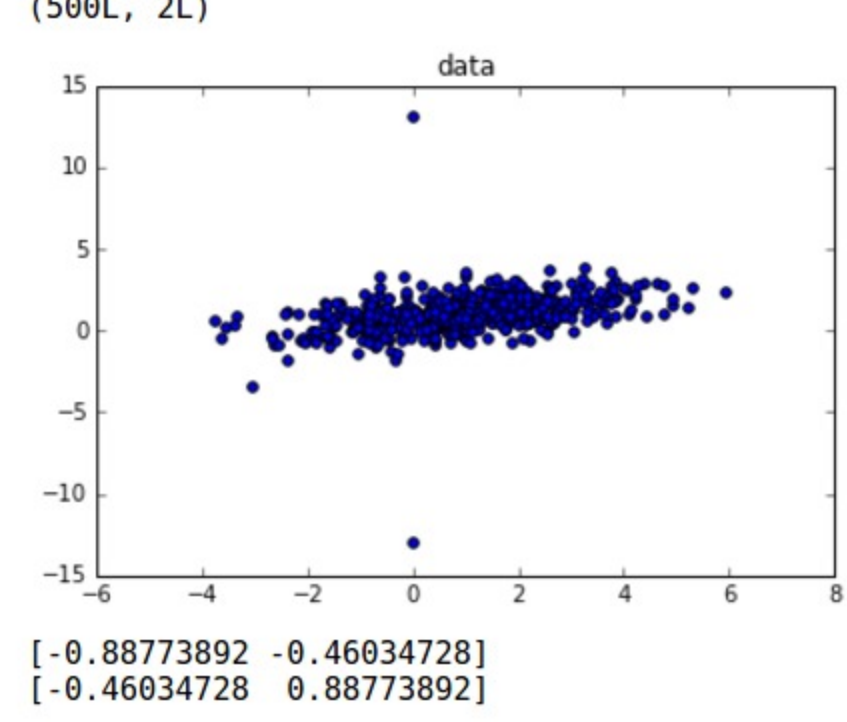
This problem sheet explores applications and extensions of PCA. The first two exercises deal with PCA as a method for preprocessing and the third one illustrates how to find nonlinear structure via kernel PCA.

4.1 Preprocessing (2 points)

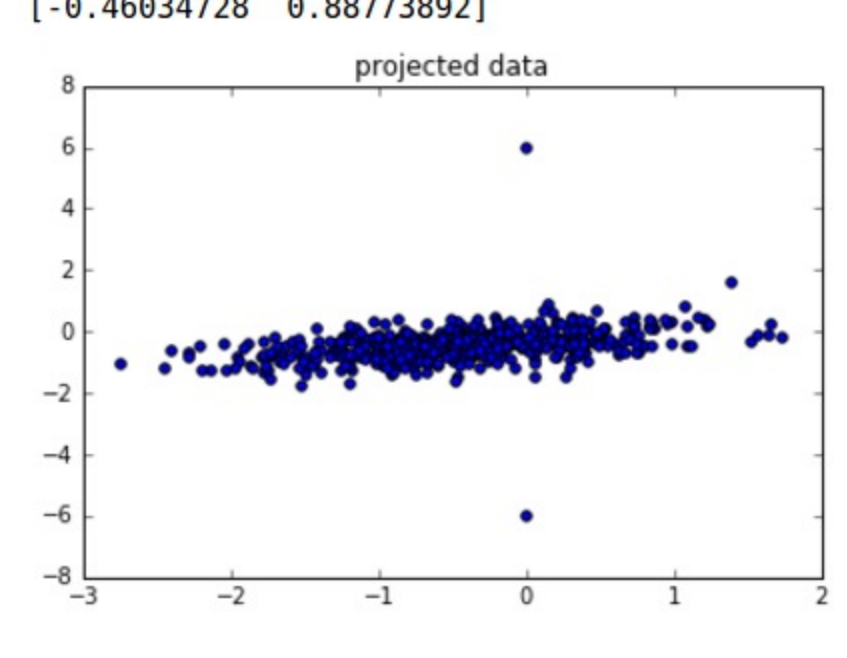
(a) Load the dataset pca2.csv. Compute the Principal Components PC1 and PC2 and plot the data in the coordinate system PC1 vs. PC2 – What do you observe?

In [10]:
```
from numpy import genfromtxt
#laden:
data = genfromtxt('pca2.csv',delimiter=',',skip_header=1)
print data.shape
plt.scatter(data[:,0], data[:,1])
plt.title("data")
plt.show()
#calc PCA
eigenvectors, eigenvalues = pca(data)
print eigenvectors[0]
print eigenvectors[1]
#project data
projected_data, sigma = project(data, eigenvectors[0,1])
plt.scatter(projected_data[:,0], projected_data[:,1])
plt.title("projected data")
plt.show()
```

```
(500L, 2L)
```




```
[-0.88773892 -0.46034728]
[-0.46034728  0.88773892]
```

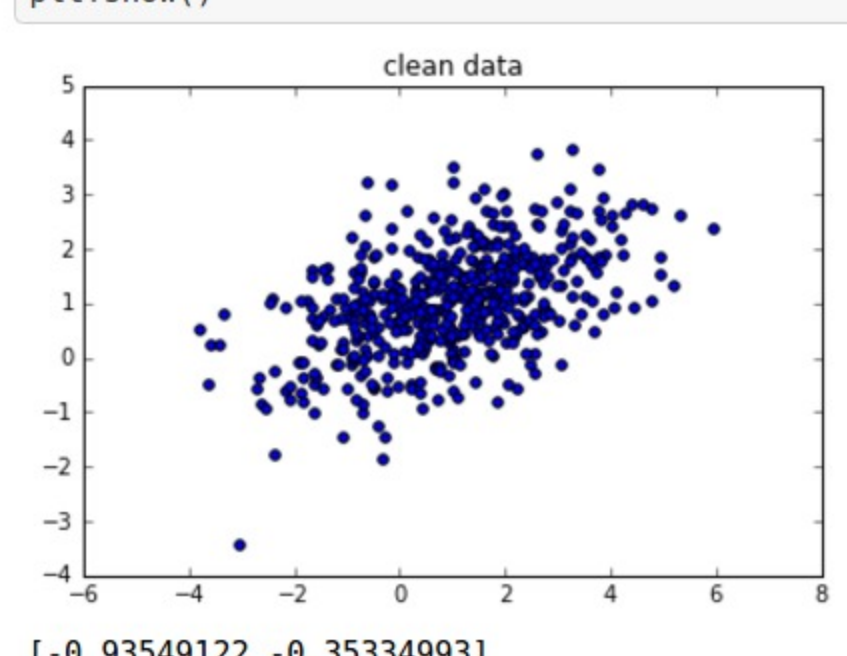


(b) Remove Observations 17 and 157 and redo the first two steps. What is the difference?

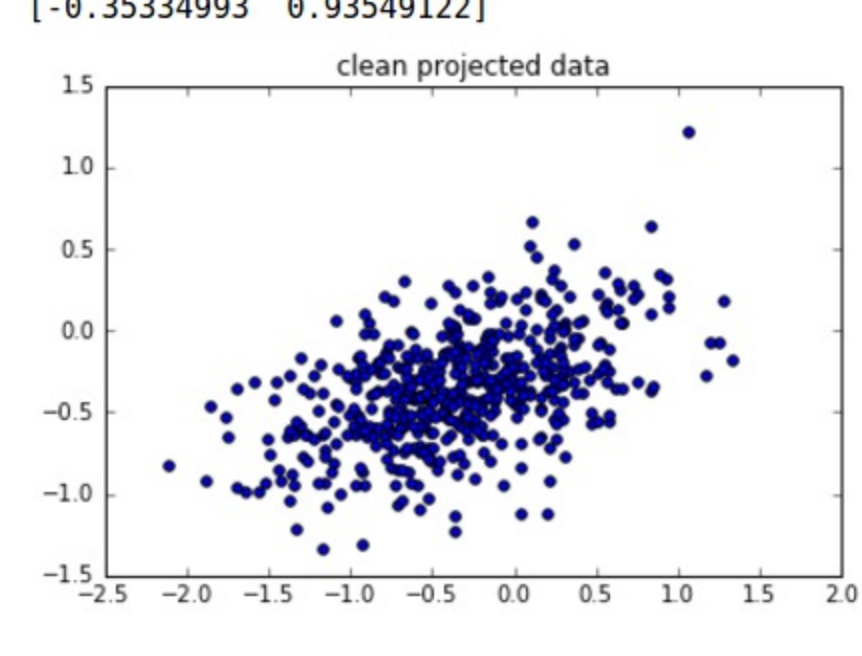
In [18]:
```
delete = [16,156]
clean_data = np.delete(data, (delete), axis=0)
print clean_data.shape
```

```
(498L, 2L)
```

In [19]:
```
plt.scatter(clean_data[:,0], clean_data[:,1])
plt.title("clean data")
plt.show()
#calc PCA
clean_eigenvectors, clean_eigenvalues = pca(clean_data)
print clean_eigenvectors[0]
print clean_eigenvectors[1]
#project data
clean_projected_data, clean_sigma = project(clean_data, clean_eigenvectors[0,1])

plt.scatter(clean_projected_data[:,0], clean_projected_data[:,1])
plt.title("clean projected data")
plt.show()
```




```
[-0.93549122 -0.35334993]
[-0.35334993  0.93549122]
```

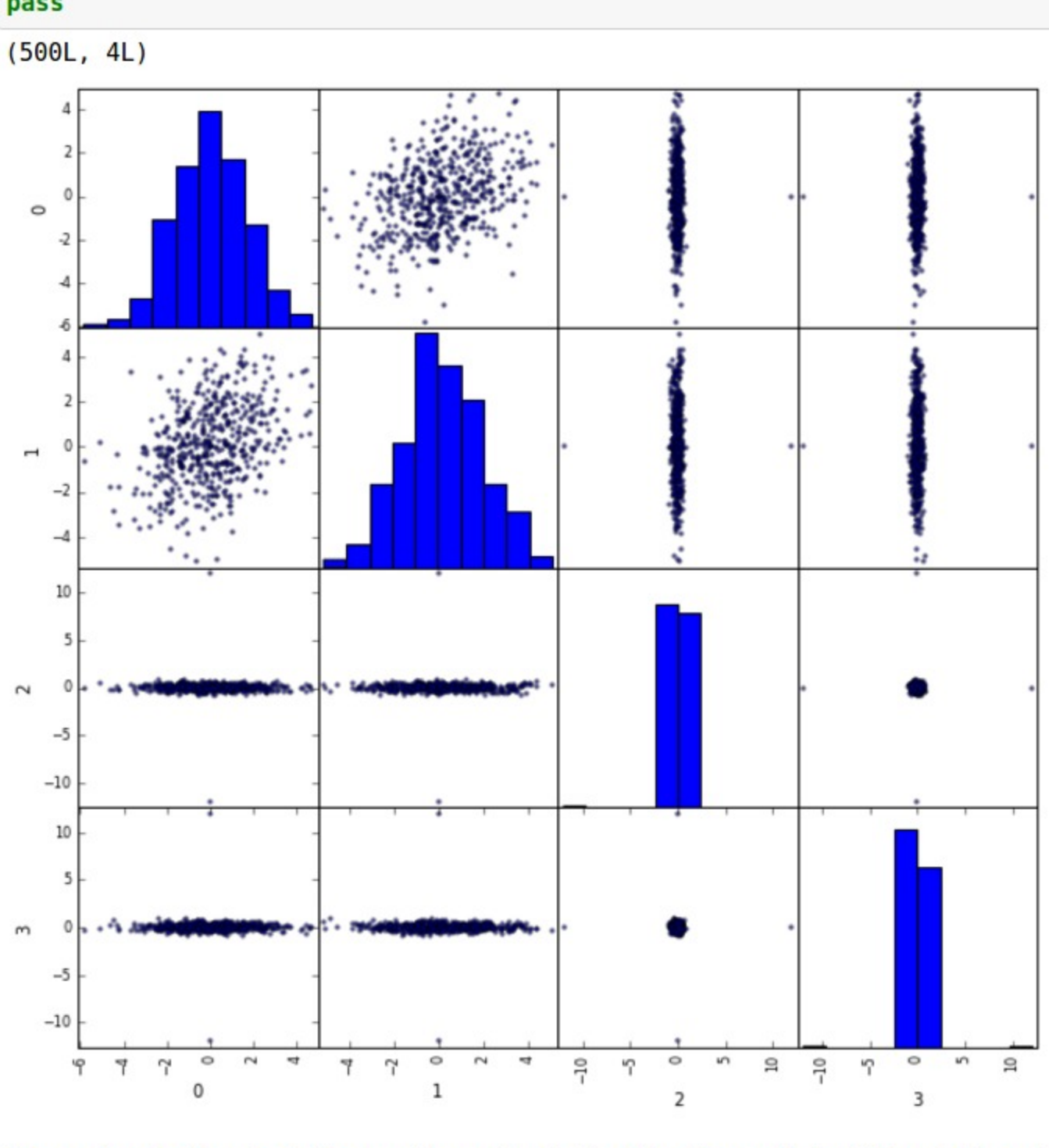


4.2 Whitening (3 points)

(a) Load the dataset pca4.csv and check for outliers in the individual variables.

In [21]:
```
import pandas as pd
from pandas.tools.plotting import scatter_matrix
from numpy import genfromtxt
#laden:
data = genfromtxt('pca4.csv',delimiter=',',skip_header=1)
print data.shape

df = pd.DataFrame(data[:,:])
# make a scatter plot
scatter_matrix(df, figsize=(10, 10))
pass
```

```
(500L, 4L)
```



From a visual outlier check, there are four outliers in the data. Two are in the third variable and two in the fourth. All of them are around a numerical value of 10 (-10). Therefore all datapoints that have observations greater than abs(8) in the third and fourth variable should be discarded.

(b) Do PCA on a reasonable subset of this data. Use a scree plot to determine how many PCs represent the data well.

In [43]:
```
def outlier(data, m=6):
    outliers = []
    for n in range(0, data.shape[0]):
        for m in range(0, data.shape[1]):
            if(abs(data[n,m])>8):
                outliers.append(n)
    return outliers
```

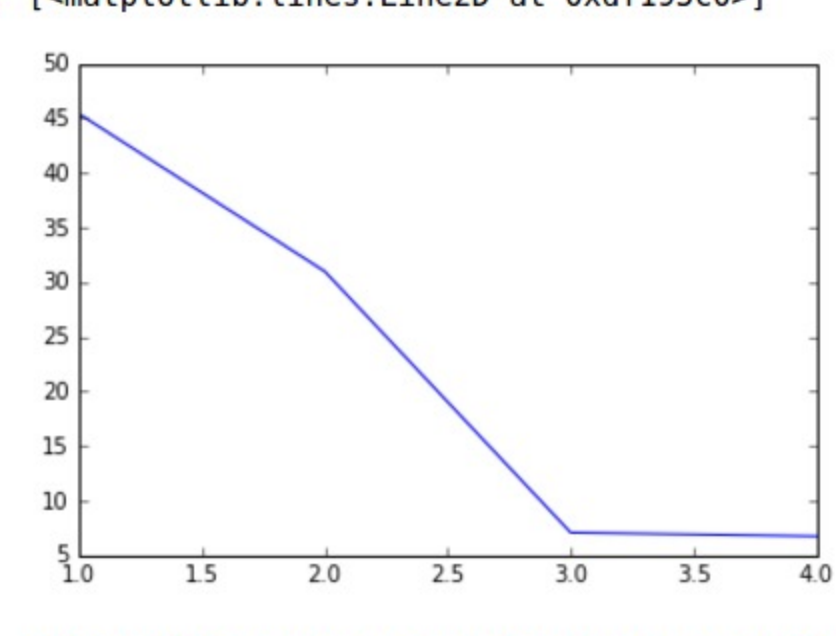
In [44]:
```
outliers = outlier(data)

print outliers

clean_data = np.delete(data, (outliers), axis=0)
print clean_data.shape
```

```
[99, 111, 199, 211]
(496L, 4L)
```

In [48]:
```
clean_eigenvectors, clean_eigenvalues = pca(clean_data)
print clean_eigenvalues
plt.plot([1,2,3,4], clean_eigenvalues)
```

```
[ 45.41447949  30.98736096   7.08142664   6.74163331]
```

Out[48]:
```
[<matplotlib.lines.Line2D at 0xdf195c0>]
```



The scree plot indicates that three Principal Components represent the data well.

(c) "Whiten" the data, i.e. create a set of 4 *uncorrelated* variables with *mean 0* and *standard deviation equal to 1*. This can be done e.g. using the transformation

$$Z = \tilde{X} E D^{-1/2}$$

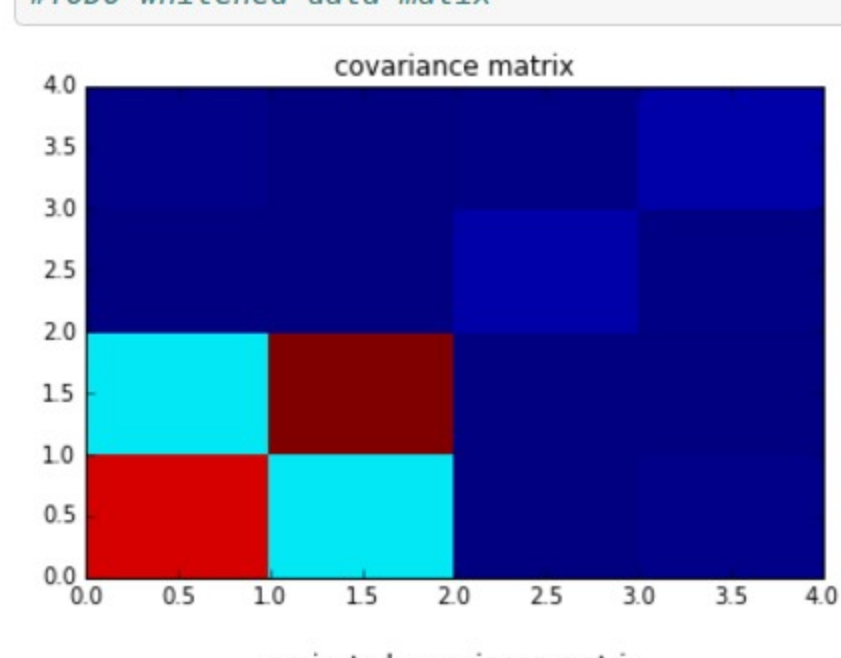
The new variables z_i form the columns of Z, E is a matrix containing the normalized eigenvectors of the covariance matrix Σ of the centered data $\tilde{X}$ and D is a diagonal matrix containing the corresponding eigenvalues.

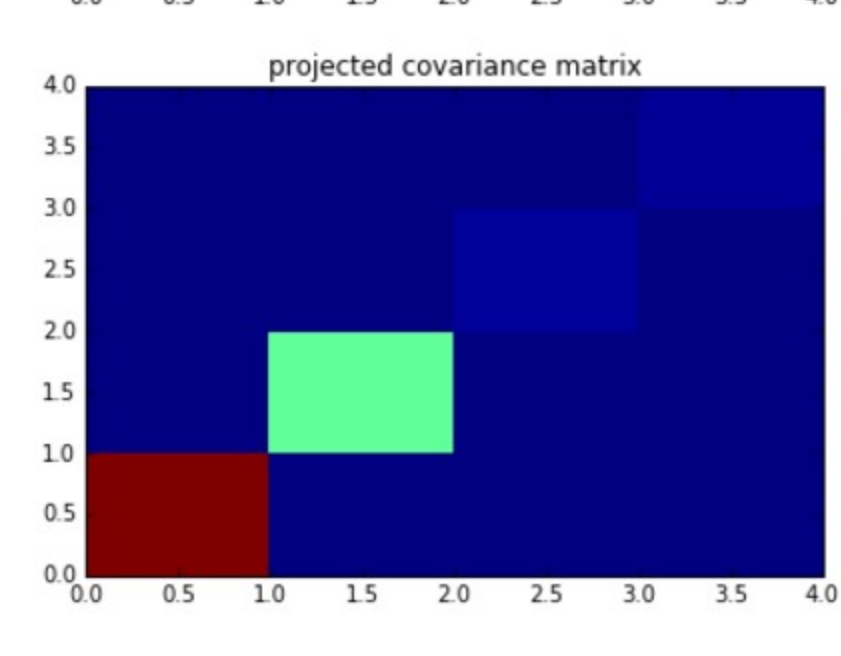
In [53]:
```
#center data
X = clean_data - clean_data.mean(axis=0)
print X.shape
#normalized eigenvectors
E = clean_eigenvectors / clean_eigenvectors.sum(axis=1, keepdims=True)
print E.shape
#eigenvalue matrix
D = np.diag(-np.sqrt(clean_eigenvalues))
print D.shape

#TODO whats the problem here?
Z = X*E*D
```

```
(496L, 4L)
(4L, 4L)
(4L, 4L)

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-53-01bc03d3832f> in <module>()
      8 D = np.diag(-np.sqrt(clean_eigenvalues))
      9 print D.shape
---> 10 Z = X*E*D

ValueError: operands could not be broadcast together with shapes (496,4) (4,4)
```

(d) Make 3 heat plots of the (i) 4x4 covariance matrix Σ, (ii) the covariance matrix of the data projected onto PC1-PC4, and (iii) of the whitened variables.

In [60]:
```
cov = np.cov(clean_data.T)
plt.pcolor(cov)
plt.title('covariance matrix')
plt.show()

cov_projected = np.cov( project(clean_data, clean_eigenvectors)[0].T)
plt.pcolor(cov_projected)
plt.title('projected covariance matrix')
plt.show()

#TODO whitened data matix
```

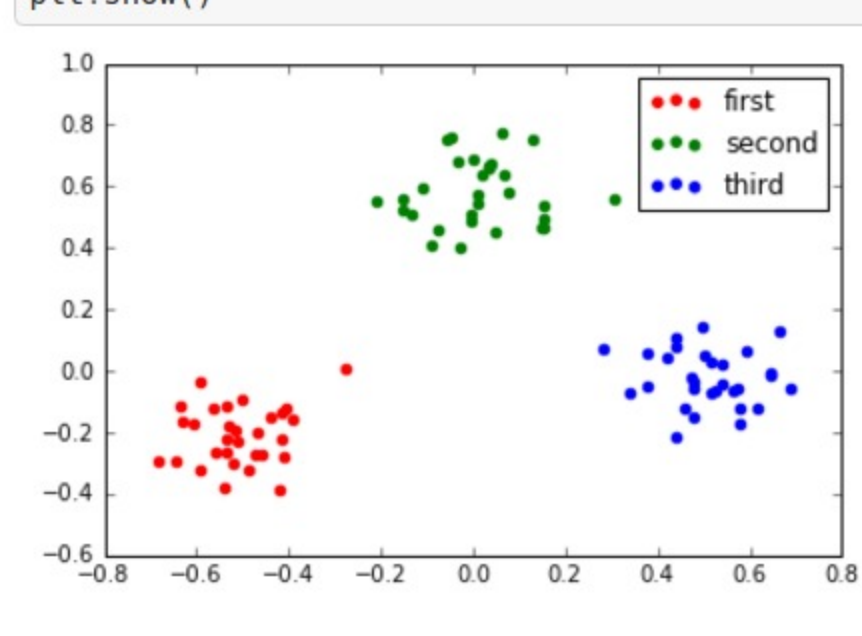




4.3 Kernel PCA: Toy Data (5 points)

(a) Create a toy dataset of 2-dimensional data points $x^{(\alpha)} = (x_1^{(\alpha)}, x_2^{(\alpha)})$, $\alpha = 1, \dots, 90$. The points represent iid samples of 30 points from 3 different distributions with uncorrelated, normally distributed (sd=0.1) coordinate values differing only in their mean value. The first sample ($\alpha = 1, \dots, 30$) should be centered on $\langle x^{(\alpha)} \rangle_1 = (-0.5, -0.2)$, the second ($\alpha = 31, \dots, 60$) on $\langle x^{(\alpha)} \rangle_2 = (0, 0.6)$, and the third ($\alpha = 61, \dots, 90$) on $\langle x^{(\alpha)} \rangle_3 = (0.5, 0)$.

In [3]:
```
# prepare
sd = 0.1
data = np.random.normal(0.0, sd, (90, 2))
data[:30] = data[:30] + np.array([-0.5, -0.2])
data[30:60] = data[30:60] + np.array([0, 0.6])
data[60:] = data[60:] + np.array([0.5, 0])

# plot to check correctness
plt.scatter(data[:30,0], data[:30,1], color='r', label='first')
plt.scatter(data[30:60,0], data[30:60,1], color='g', label='second')
plt.scatter(data[60:,0], data[60:,1], color='b', label='third')
plt.legend()
plt.show()
```



(b) Apply a Kernel PCA using the RBF kernel (see below) with a suitable parameter value for the width σ of the kernel and calculate the coefficients for the representation of the eigenvectors (PCs) in the space spanned by the transformed data points.

In [4]:
```
from scipy.spatial.distance import pdist, squareform
from scipy.linalg import eigh
```

Remark 2:

RBF kernel: $k(x^{(\alpha)}, x^{(\beta)}) = \exp\left(-\frac{\|x^{(\alpha)} - x^{(\beta)}\|^2}{2\sigma^2}\right)$

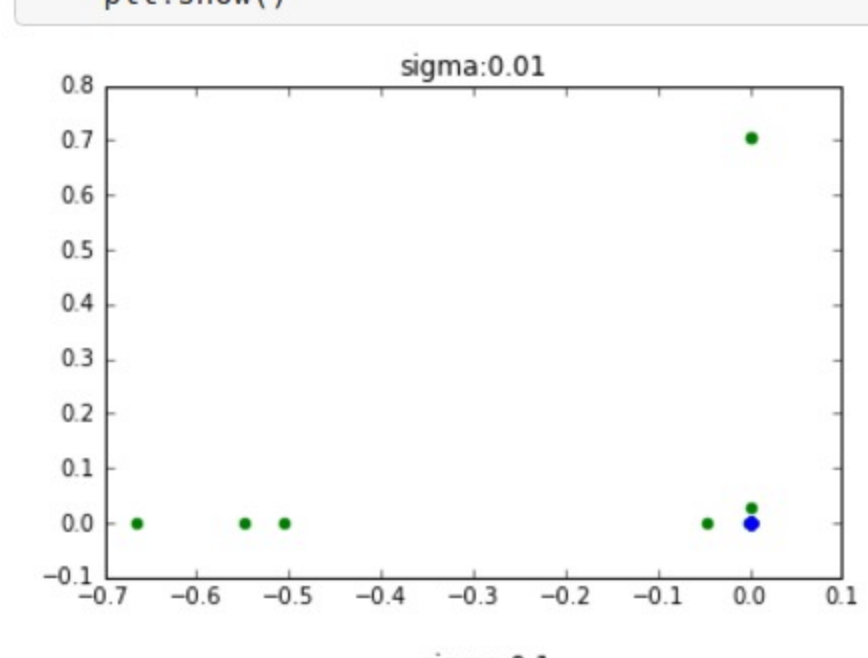
In [5]:
```
def rbf(x, sigma):
    dists = squareform(pdist(x, 'euclidean'))
    return np.exp(- np.power(dists, 2) / (2* sigma**2))
```

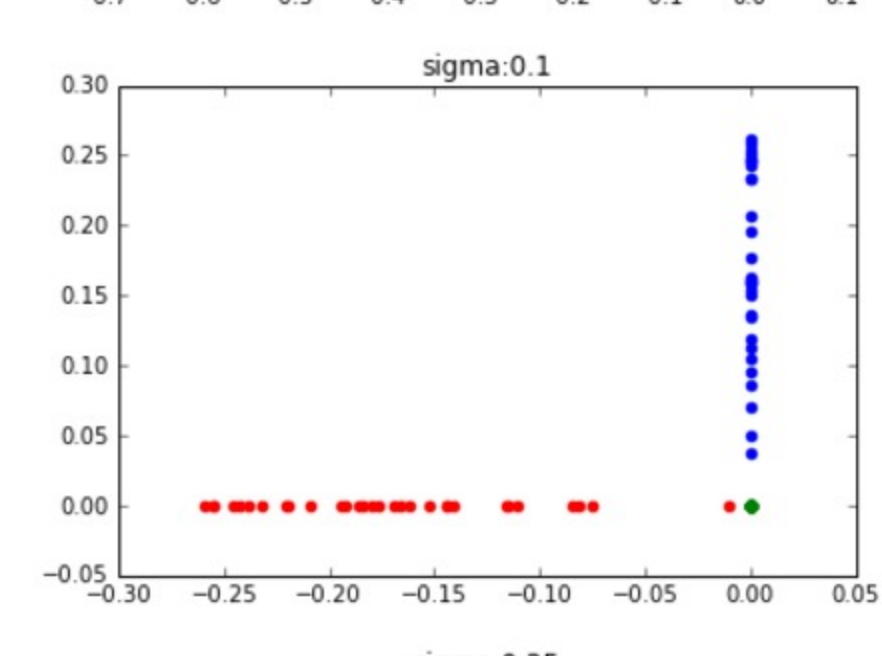
Remark 1: Ensure to center the kernel matrix of the data points before projecting them onto the PCs. Furthermore, since most test points will differ from the sampled data points, you have to ensure that also when calculating the PC projections of these points centered feature vectors are considered.

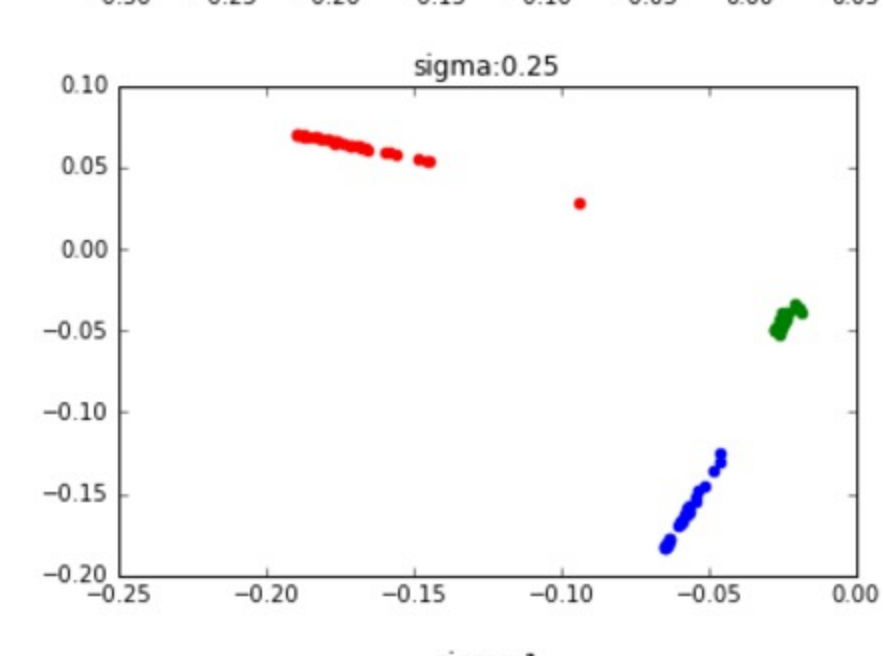
In [12]:
```
def center_kernelmatrix(x):
    N = x.shape[0]
    one_n = np.ones((N,N)) / N
    x = x - one_n.dot(x) - x.dot(one_n) + one_n.dot(x).dot(one_n)
```

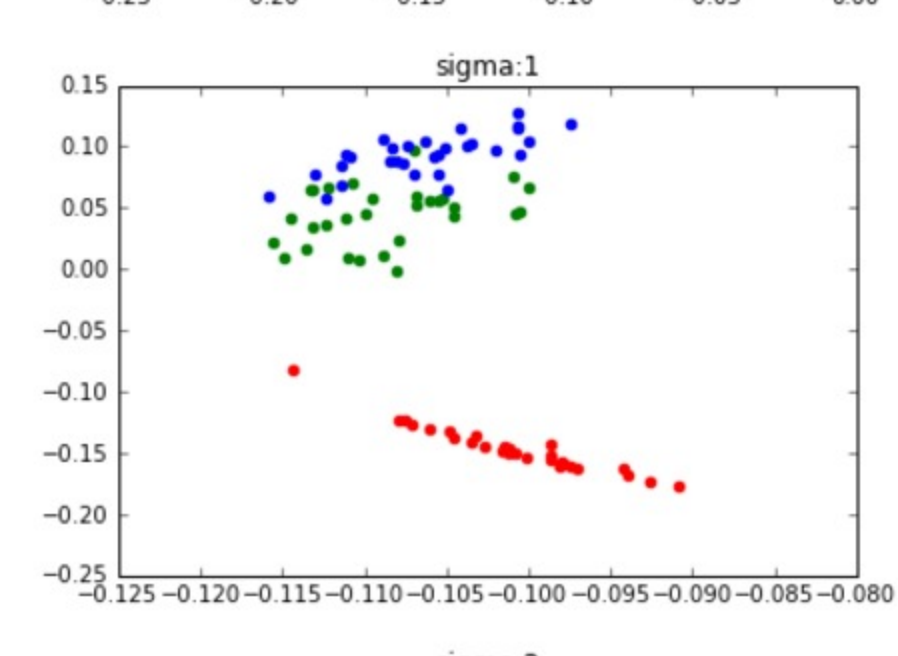
In [13]:
```
def kernel_pca(data, center=True):
    if center:
        center_kernelmatrix(data)
    eigenvalues, eigenvectors = eigh(data)
    return eigenvectors[:,::-1], eigenvalues[::-1]
```

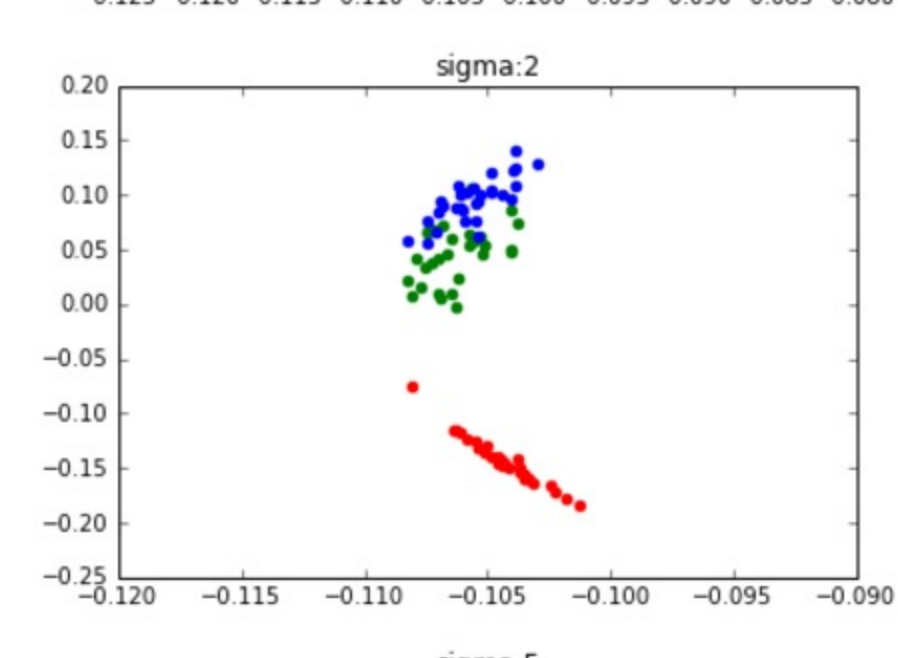
In [19]:
```
sigmas = [0.01, 0.1, 0.25, 1, 2, 5]
for sigma in sigmas:
    rbf_vecs, rbf_vals = kernel_pca(rbf(data, sigma))
    plt.scatter(rbf_vecs[:30,0], rbf_vecs[:30,1], color='r')
    plt.scatter(rbf_vecs[30:60,0], rbf_vecs[30:60,1], color='g')
    plt.scatter(rbf_vecs[60:,0], rbf_vecs[60:,1], color='b')
    plt.title('sigma:'+str(sigma))
    plt.show()
```

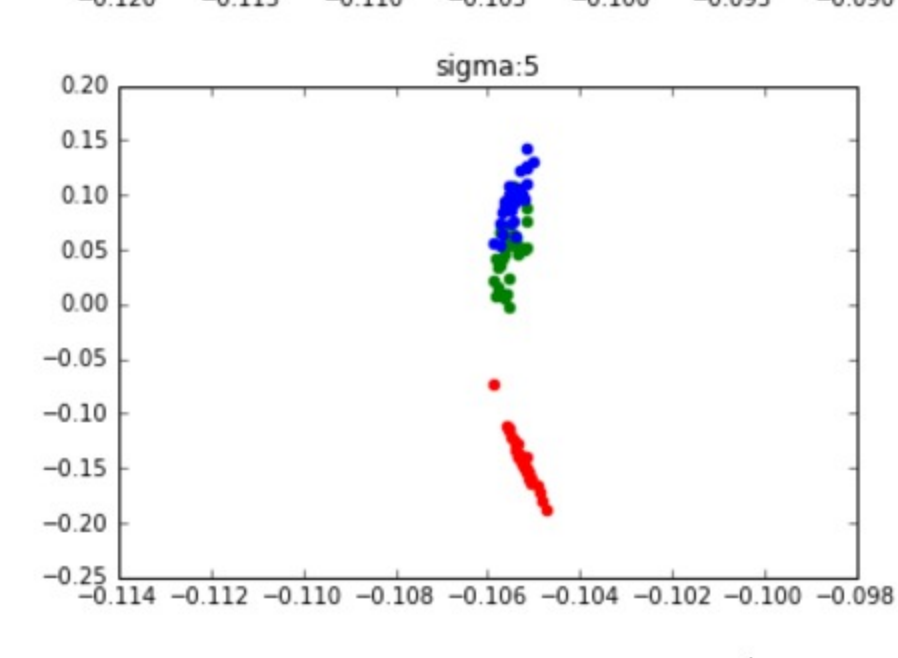












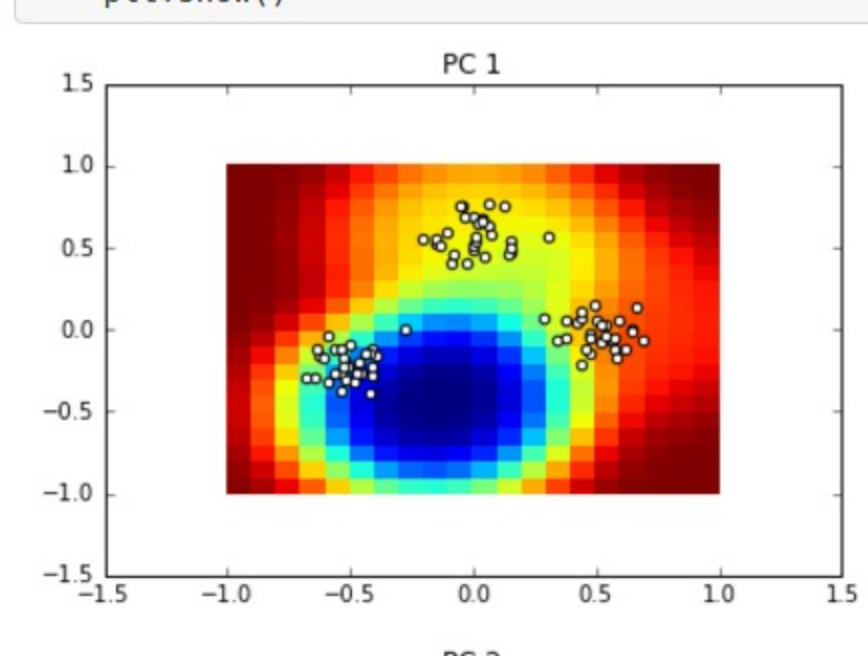
The datapoints are easiely separable using $\sigma = 0.2$ and projecting onto the second PC. There may be other σ for separation via PC one, but this result seems to be reasonable good.

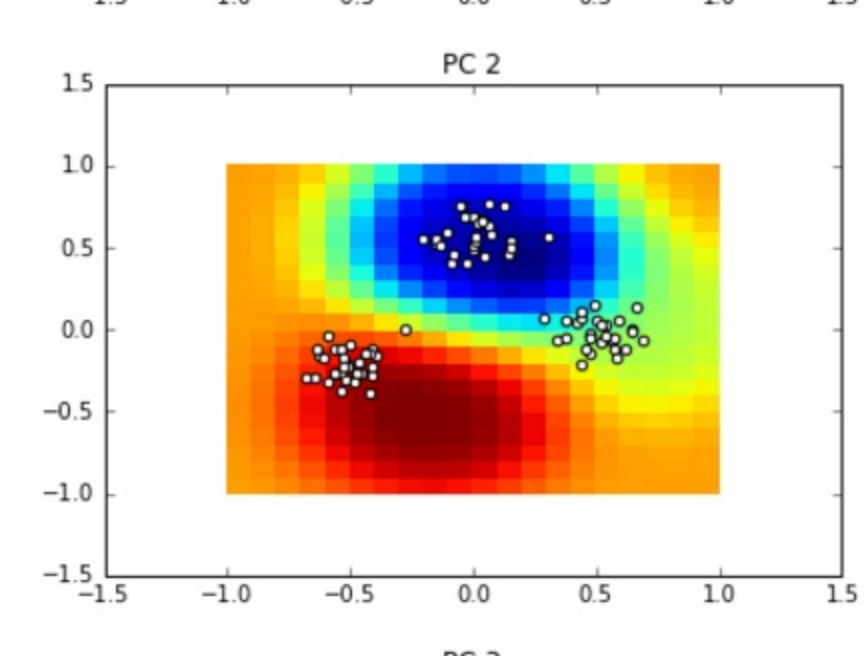
(c) Visualize the first 8 PCs in the 2-dimensional input space in the following way: Use equally spaced "test" gridpoints (in a rectangle $[a, b] \times [c, d] \subset \mathbb{R}^2$ containing all sampled data points) and determine their PC values by projecting onto the first 8 eigenvectors in feature space. For example, plot contour lines indicating points that yield the same projection onto the respective PC. You may also use a heat map or pseudo color plot (e.g. pcolor) to distinguish the different regions. Plot the 90 data points in the same plot (e.g, using small gray circles). How do you interpret the results?

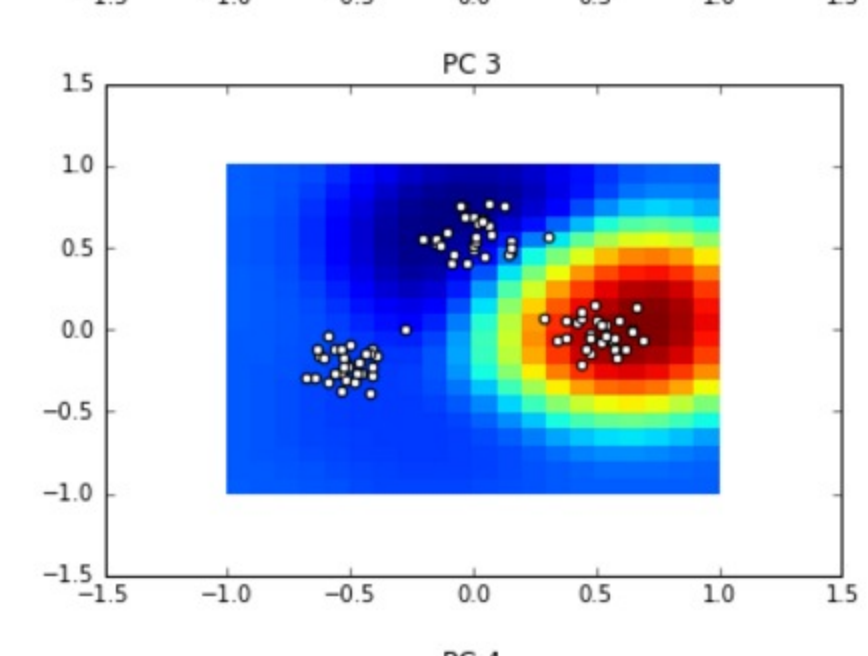
In [15]:
```
def project_kernel(pt, data, sigma, vecs, vals):
    dists = np.array([np.sum((pt-row)**2) for row in data])
    kern = np.exp(- np.power(dists, 2) / (2* sigma**2))
    return kern.dot(vecs / vals)
```

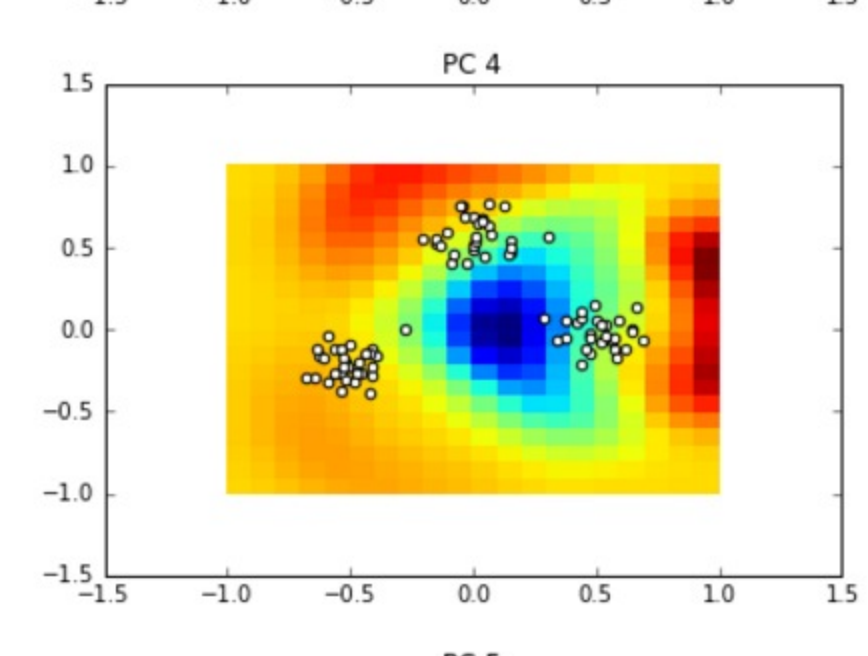
In [20]:
```
sigma = 0.25
rbf_vecs, rbf_vals = kernel_pca(rbf(data, sigma))
result = None
for i in np.arange(-1,1,0.1):
    for j in np.arange(-1,1,0.1):
        pt = np.array([i,j])
        proj = project_kernel(pt, data, sigma, rbf_vecs[:,:8], rbf_vals[:8])
        if result is None:
            result = proj
        else:
            result = np.vstack((result, proj))
```

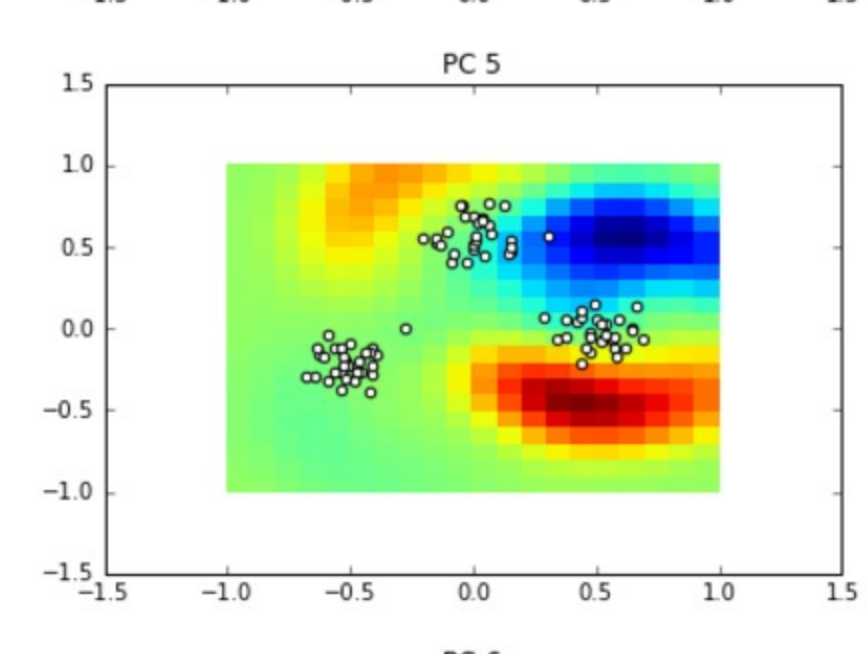
In [31]:
```
reresult = np.reshape(result, (20,20,8))
y, x = np.mgrid[slice(-1, 1.1, 0.1),
                slice(-1, 1.1, 0.1)]
for i in range(8):
    plt.pcolor(x, y, reresult[:,:,i])
    plt.scatter(data[:,0], data[:,1], color='w', edgecolors='k')
    plt.title('PC '+str(i+1))
    plt.show()
```

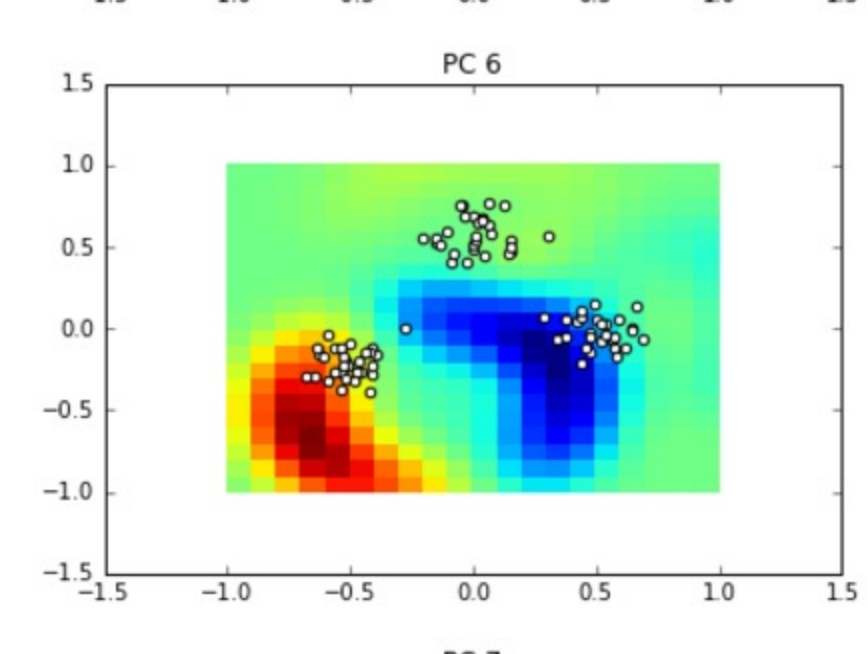


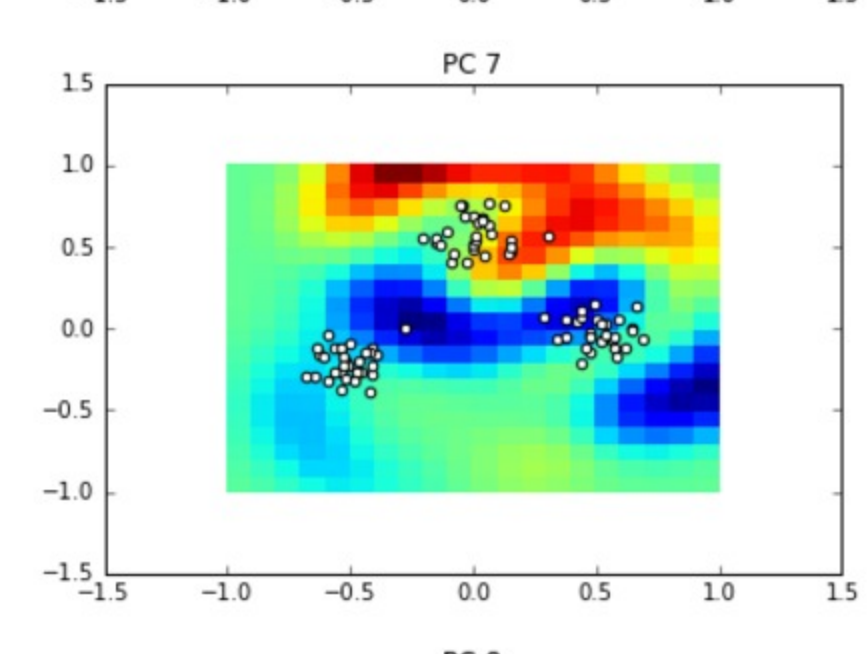


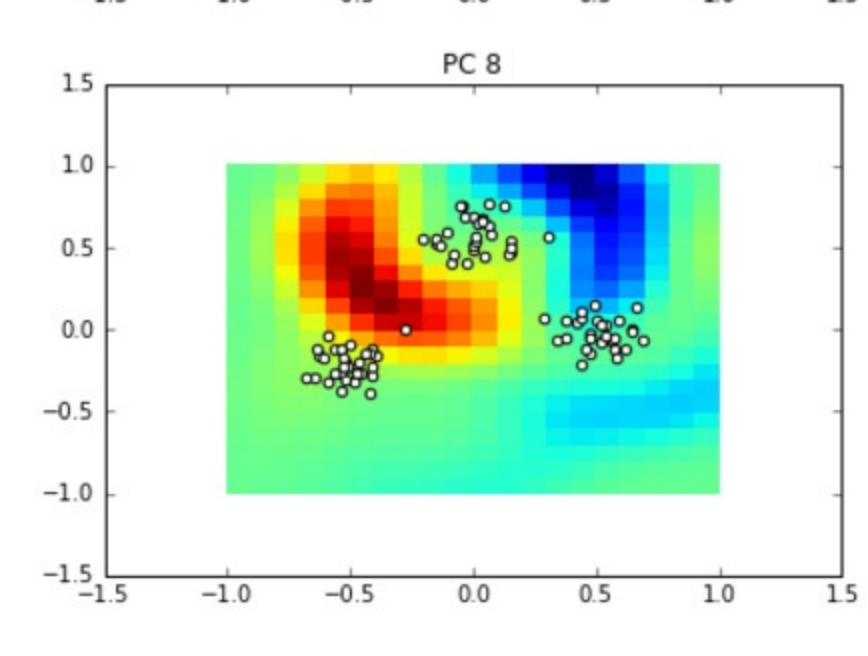












Desc:
We plotted 8 plots, one for every PC. This greatly improved readability for us and therefor better interpretations. We are excited to see the results of other groups for this plot!

Interpretation:
Using the pseudo color plot, we can nicely see the radial basis for every PC. The result alignes nicely with the previous plots where we saw that we can map onto the second PC best. Here, the projection of all test points onto the second PC also makes clear, that two radial basis align with two the clusters of out sample data points. the third cluster does not have a basis on its own, but we could have guessed that from the previous plots with lower σ values. To classify the different clusters, a projection of new test points would hopefully yield high and low projection values for the two clusters (or classes) backend by the radial bases, and intermediate values for the third cluster. Sadly, there may be intermediate values in between the two radial basis that could be classified as the third cluster. Also, any outlier will most probably be classified as cluster three. This is a negative aspect of the radial basis kernels, but we think we reached a reasonably good result never the less. For the other PC's, the basis seem to try to separated the clusters of just focus on one of them. This probably makes them good for reconstruction of the original data, but is not that useful for classification