2023-05-15_ChartTypes_022_PCA

June 5, 2023

[1]:
```python
import numpy as np
import scipy
import imageio

import matplotlib
import matplotlib.pyplot as plt
import matplotlib.cm as cm

matplotlib.rc('image', interpolation='nearest')
matplotlib.rc('figure',facecolor='white')
matplotlib.rc('image',cmap='viridis')
colors=plt.rcParams['axes.prop_cycle'].by_key()['color']
%matplotlib inline
```

1 PCA

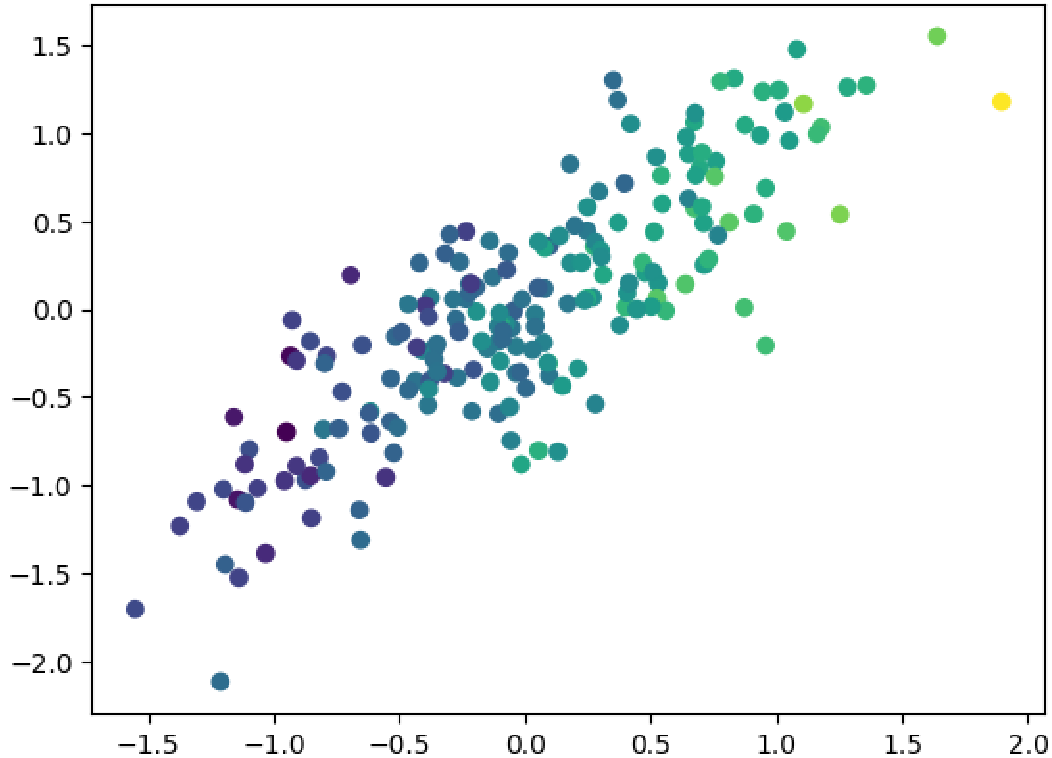
[2]:
```python
# do simple PCA on data matrix
# dataMat is assumed to be centered, matrix of shape (nSamples,dimSample)
def PCA(dataMat,keep=None):
    nSamples,dim=dataMat.shape
    if dim<nSamples:
        if keep is None:
            keep=dim
        A=dataMat.transpose().dot(dataMat)/nSamples
        eigData=np.linalg.eigh(A)
        eigval=(eigData[0][-keep::])[::-1]
        eigvec=((eigData[1][:,-keep::]).transpose())[::-1]
    else:
        if keep is None:
            keep=nSamples
        A=dataMat.dot(dataMat.transpose())/nSamples
        eigData=np.linalg.eigh(A)
        eigval=(eigData[0][-keep::])[::-1]
        eigvec=((eigData[1][:,-keep::]).transpose())[::-1]

        eigvec=np.einsum(eigvec,[0,1],dataMat,[1,2],[0,2])
```

```
        # renormalize
        normList=np.linalg.norm(eigvec,axis=1)
        eigvec=np.einsum(eigvec,[0,1],1/normList,[0],[0,1])
    return eigval,eigvec
```

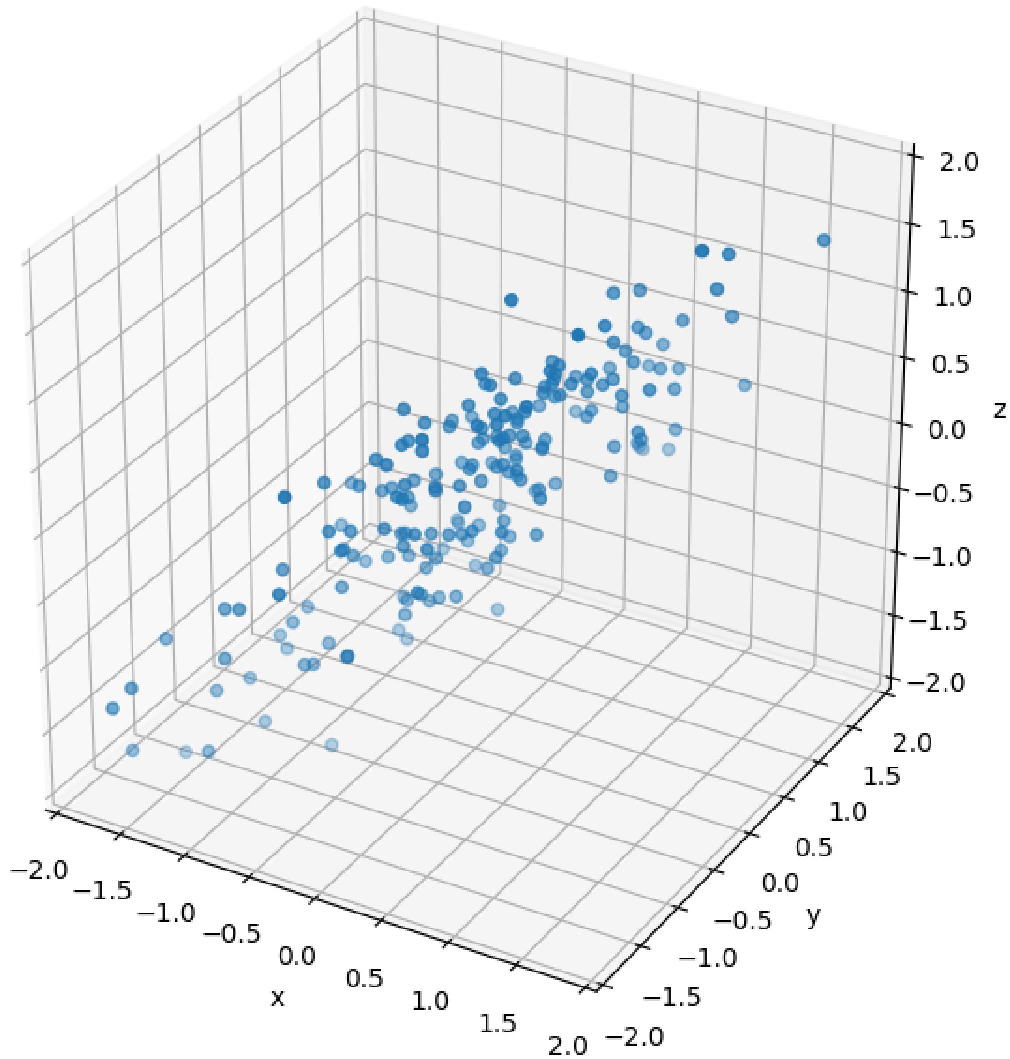
1.1 A toy example

[3]:
```
nPts=200
dim=3
data=np.random.normal(size=(nPts,dim))
# scale axes differently
scales=np.array([1.,0.5,0.2])
data=np.einsum(data,[0,1],scales,[1],[0,1])
# move directions a little
# each row of A contains a direction into which the corresponding dimension of
 ↪data should be pointed
A=np.array([[1.,1.,1.],[0.,1.,-1.],[1.,-0.5,-0.5]])
# normalize rows of A to unit length
norms=np.linalg.norm(A,axis=1)
A=np.einsum(A,[0,1],1/norms,[0],[0,1])
data=np.einsum(A,[0,1],data,[2,0],[2,1])
# center the data
dataMean=np.mean(data,axis=0)
data-=dataMean
```

[4]:
```
# just visualize the first 2 dimensions, plug third dimension into color
# we see: color is definitely strongly correlated with position
# so the positions do not tell the full story of the data
%matplotlib inline
plt.scatter(data[:,0],data[:,1],c=data[:,2])
plt.show()
```



[7]:
```python
# visualize as 3d point cloud
%matplotlib widget
fig = plt.figure(figsize=(6,6))
ax = fig.add_subplot(111, projection='3d')
# aspect ratio
ax.set_xlim([-2,2])
ax.set_ylim([-2,2])
ax.set_zlim([-2,2])
ax.set_box_aspect((1.,1.,1.))

ax.scatter(data[:,0],data[:,1],data[:,2])
ax.set_xlabel("x")
ax.set_ylabel("y")
ax.set_zlabel("z")
plt.tight_layout()
plt.show()
```



[8]:
```
plt.close()
%matplotlib inline
```

[4]:
```
eigval,eigvec=PCA(data)
```

[5]:
```
# eigenvalues: correspond to variance along the principal directions
eigval
```

[5]:
```
array([0.93685017, 0.24951978, 0.03782157])
```

[6]:
```
# compare with predictions: scale factors in original Gaussians, squared
(scales)**2
```

[6]: array([1. , 0.25, 0.04])

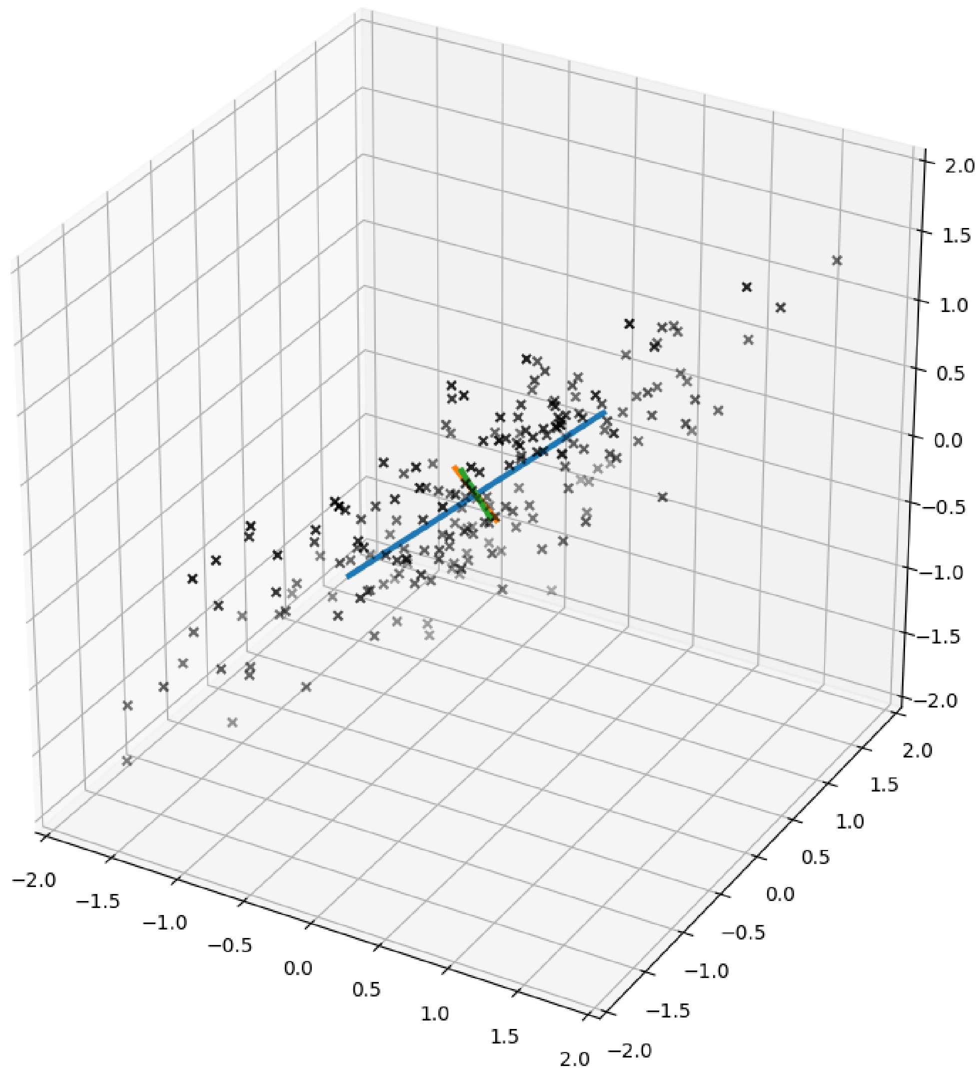
[7]:
```
# same for eigenvectors
eigvec
```

[7]:
```
array([[-0.61400857, -0.6193394 , -0.48929764],
       [ 0.08315523, -0.66722254,  0.74020219],
       [-0.7849068 ,  0.41380283,  0.46118167]])
```

[8]:
```
# compare with rows of A
A
```

[8]:
```
array([[ 0.57735027,  0.57735027,  0.57735027],
       [ 0.        ,  0.70710678, -0.70710678],
       [ 0.81649658, -0.40824829, -0.40824829]])
```

[9]:
```
# add eigenvectors with scale to plot
%matplotlib inline
fig = plt.figure(figsize=(8,8))
ax = fig.add_subplot(111, projection='3d')
# aspect ratio
ax.set_xlim([-2,2])
ax.set_ylim([-2,2])
ax.set_zlim([-2,2])
ax.set_box_aspect((1.,1.,1.))
# this step is required such that eigenvectors actually appear perpendicular to␣
 ↪each other

ax.scatter(data[:,0],data[:,1],data[:,2],marker="x",color="k")
for i,(val,vec) in enumerate(zip(eigval,eigvec)):
    # let x be the (unit) eigenvector, scaled by sqrt of eigenvalue (=std␣
 ↪deviation)
    x=vec*(val**0.5)
    # add line from -x to x into plot
    ax.plot([-x[0],x[0]],[-x[1],x[1]],[-x[2],x[2]],color=colors[i],lw=3)
plt.tight_layout()
plt.show()
```



[10]:
```
# projecting the points onto the eigenbasis
# (position of each point along the three colored lines)
coef=np.einsum(eigvec,[0,1],data,[2,1],[2,0])
print(coef.shape)
```

```
(200, 3)
```

[11]:
```
# compare covariance matrix of naive data, and of PCA coefficients
# by construction: the latter is diagonal
covAfter=np.cov(coef.transpose())
covBefore=np.cov(data.transpose())
print(covBefore)
print()
```

```
print(covAfter)
```

```
[[0.38012569 0.33179565 0.28455095]
 [0.33179565 0.47931376 0.16873304]
 [0.28455095 0.16873304 0.37090379]]

[[ 9.41557963e-01  1.45268061e-16 -7.75646453e-17]
 [ 1.45268061e-16  2.50773650e-01 -1.80758074e-18]
 [-7.75646453e-17 -1.80758074e-18  3.80116235e-02]]
```

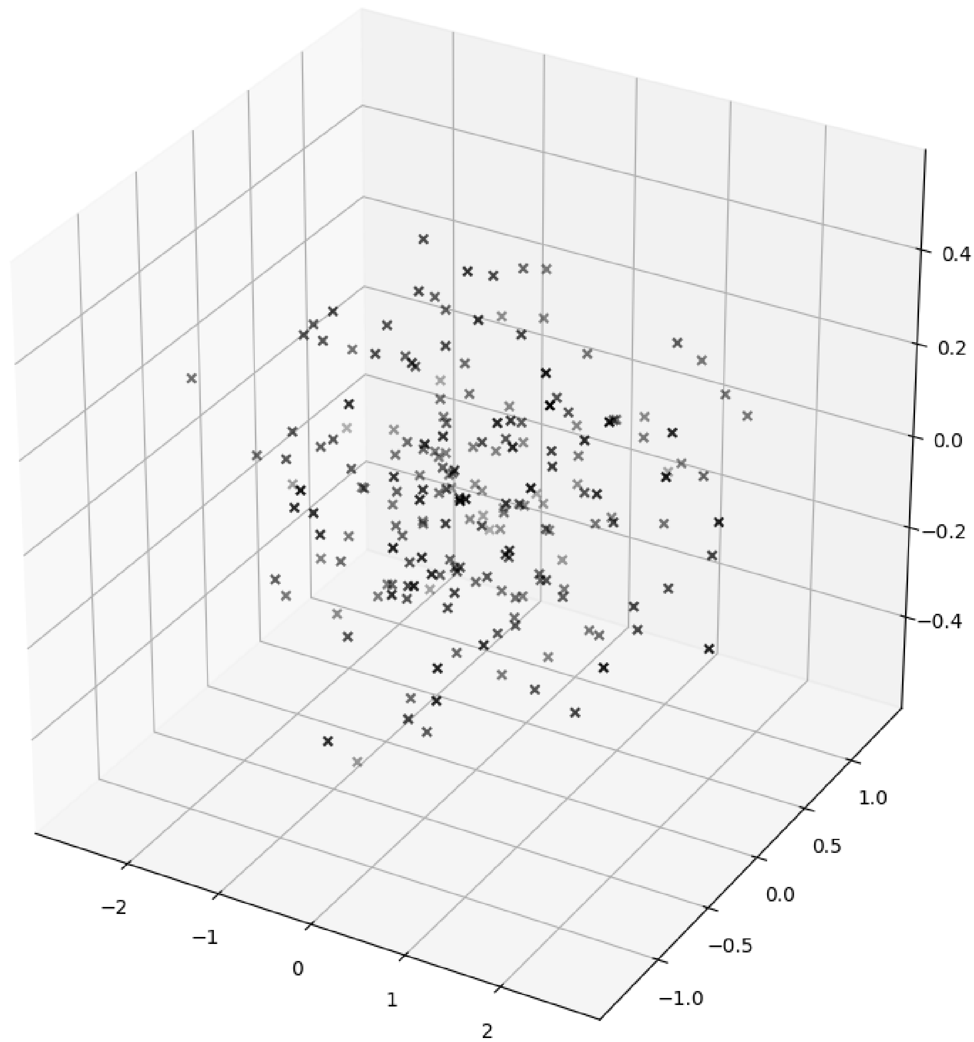
[12]:
```
# compare empirical variance of each coefficient with eigenvalues
np.var(coef,axis=0)
```

[12]:
```
array([0.93685017, 0.24951978, 0.03782157])
```

[13]:
```
eigval
```

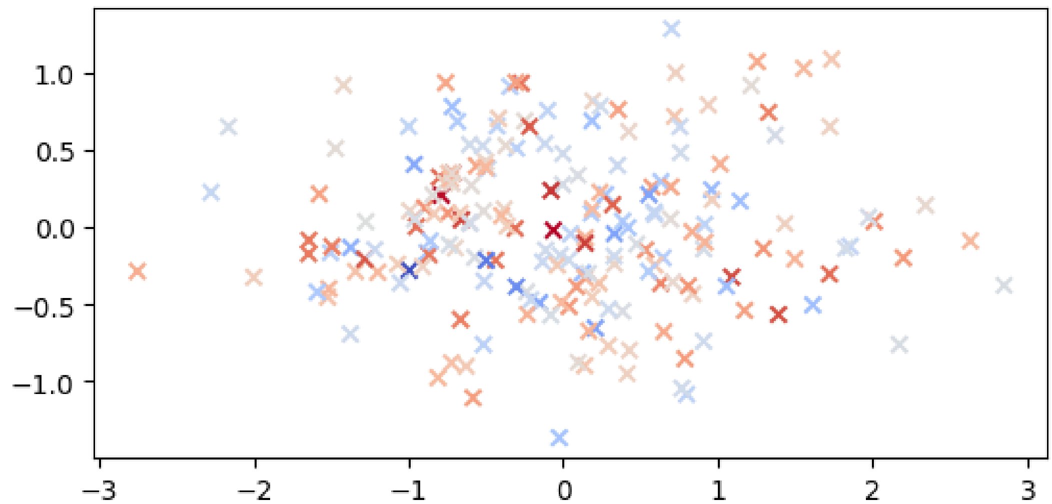
[13]:
```
array([0.93685017, 0.24951978, 0.03782157])
```

[14]:
```
# plot transformed coordinates
%matplotlib inline
fig = plt.figure(figsize=(8,8))
ax = fig.add_subplot(111, projection='3d')
# set lim of each axis to -3 to +3 sigma of the given axis
ax.set_xlim([-3*eigval[0]**0.5,3*eigval[0]**0.5])
ax.set_ylim([-3*eigval[1]**0.5,3*eigval[1]**0.5])
ax.set_zlim([-3*eigval[2]**0.5,3*eigval[2]**0.5])
# set uniform aspect ratio = plot will look like cube
ax.set_box_aspect((1.,1.,1.))

ax.scatter(coef[:,0],coef[:,1],coef[:,2],marker="x",color="k")
plt.tight_layout()
plt.show()
```



[15]:
```
plt.close()
%matplotlib inline
```

[16]:
```
%matplotlib inline
fig=plt.figure()
fig.add_subplot(aspect=1.)
plt.scatter(coef[:,0],coef[:,1],marker="x",c=coef[:,2],cmap="coolwarm")
plt.show()
# this is the "best" 2d representation of the original point cloud
# in the sense that it is the 2d projection with the least loss of variance
# note: now color is uncorrelated with position
```

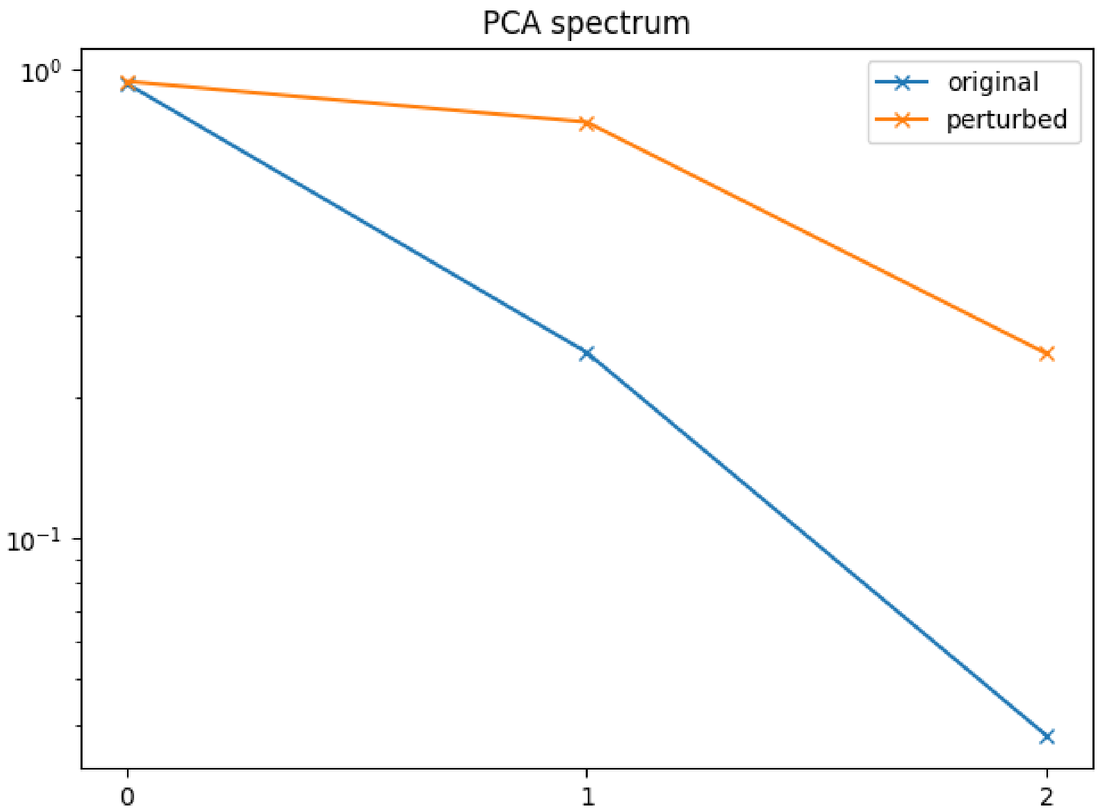


1.2 Careful: PCA is relatively susceptible to outliers

[17]:
```
# disrupt one of the points
dataNew=data.copy()
dataNew[0]=np.array([10.,-5.,-5.])
dataMean=np.mean(dataNew,axis=0)
dataNew-=dataMean
```

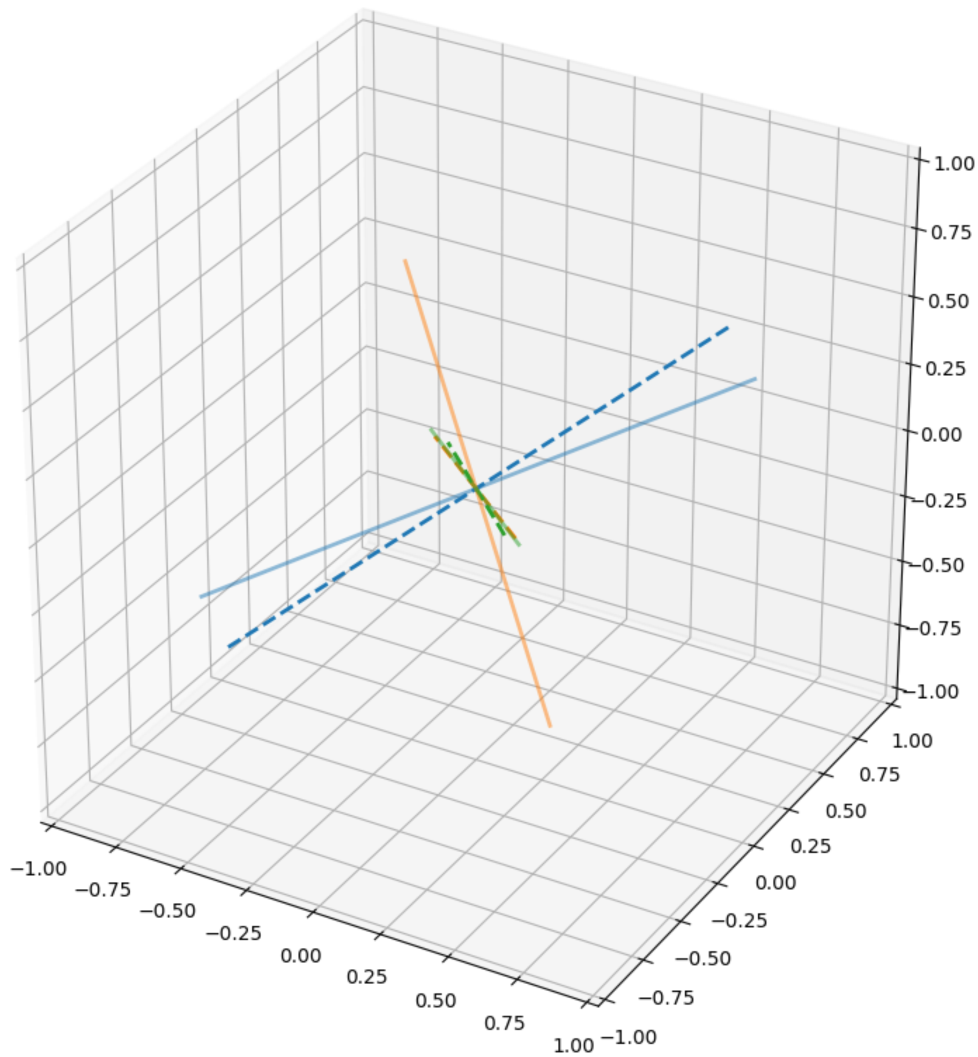
[18]:
```
eigvalNew,eigvecNew=PCA(dataNew)
```

[19]:
```
plt.title("PCA spectrum")
plt.plot(eigval,marker="x",label="original")
plt.plot(eigvalNew,marker="x",label="perturbed")
plt.yscale("log")
plt.xticks([0,1,2])
plt.legend()
plt.tight_layout()
plt.show()
```



[20]:
```
# add eigenvectors with scale to plot
%matplotlib widget
fig = plt.figure(figsize=(8,8))
ax = fig.add_subplot(111, projection='3d')
# aspect ratio
scale=1.
ax.set_xlim([-scale,scale])
ax.set_ylim([-scale,scale])
ax.set_zlim([-scale,scale])
ax.set_box_aspect((1.,1.,1.))

#ax.scatter(data[:,0],data[:,1],data[:,2],marker="x",color="k")
for i,(val,vec) in enumerate(zip(eigval,eigvec)):
    x=vec*(val**0.5)
    ax.
 ↪plot([-x[0],x[0]],[-x[1],x[1]],[-x[2],x[2]],color=colors[i],lw=2,ls="dashed")
for i,(val,vec) in enumerate(zip(eigvalNew,eigvecNew)):
    x=vec*(val**0.5)
    ax.plot([-x[0],x[0]],[-x[1],x[1]],[-x[2],x[2]],color=colors[i],lw=2,alpha=0.
 ↪5)
plt.tight_layout()
```

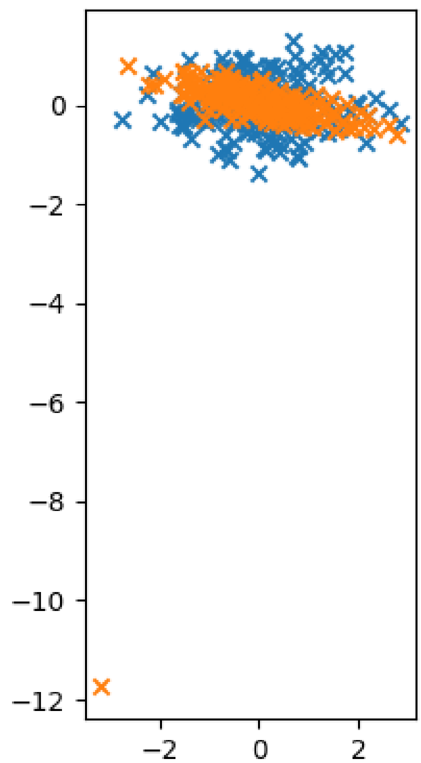
```
plt.show()
```



[21]:
```
plt.close()
%matplotlib inline
```

[22]:
```
coefNew=np.einsum(eigvecNew,[0,1],dataNew,[2,1],[2,0])
```

[23]:
```
%matplotlib inline
fig=plt.figure()
fig.add_subplot(aspect=1.)
plt.scatter(coef[:,0],coef[:,1],marker="x")
plt.scatter(coefNew[:,0],coefNew[:,1],marker="x")
plt.show()
```



1.3 Higher-dimensional example: discretized functions

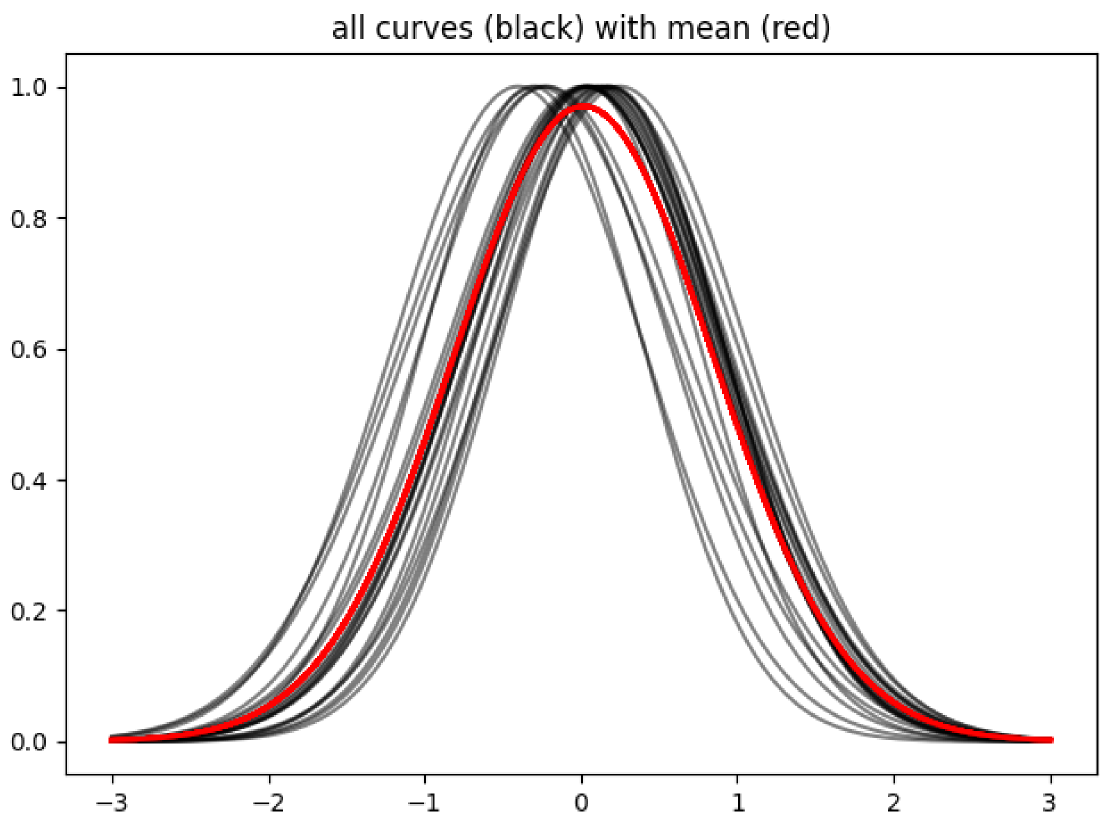
- each sample is a discrete function on nGrid gridpoints
- each sample generated as Gaussian bell curve, with random mean and variance
- we can interpret them as abstract vectors in $\mathbb{R}^{\mathrm{nGrid}}$, but also visualize them easily as functions

1.3.1 data creation

[26]:
```
nPts=100
nGrid=200
x=np.linspace(-3,3,num=nGrid)
listMean=.2*np.random.normal(size=nPts)
listVar=0.3*np.random.random(size=nPts)+0.5
listY=np.array([np.exp(-0.5*(x-mean)**2/var) for mean,var in␣
  ↪zip(listMean,listVar)])
mean=np.mean(listY,axis=0)
data=listY-np.reshape(mean,(1,-1))
```

1.3.2 processing

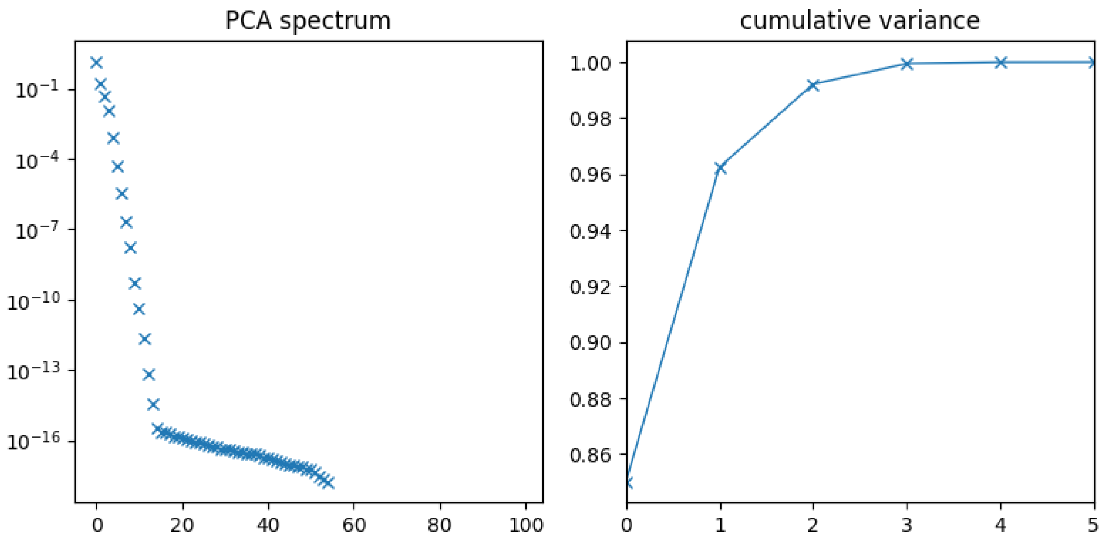
[27]:
```
# visualize all functions/vectors and the mean, as simple line plots
%matplotlib inline
plt.title("all curves (black) with mean (red)")
for y in listY[:20]:
    plt.plot(x,y,c="k",alpha=0.5)
    plt.plot(x,mean,c="r",lw=2,zorder=2)
plt.tight_layout()
plt.show()
```



[28]:
```
# perform PCA, interpret functions as abstract vectors
eigval,eigvec=PCA(data)
```

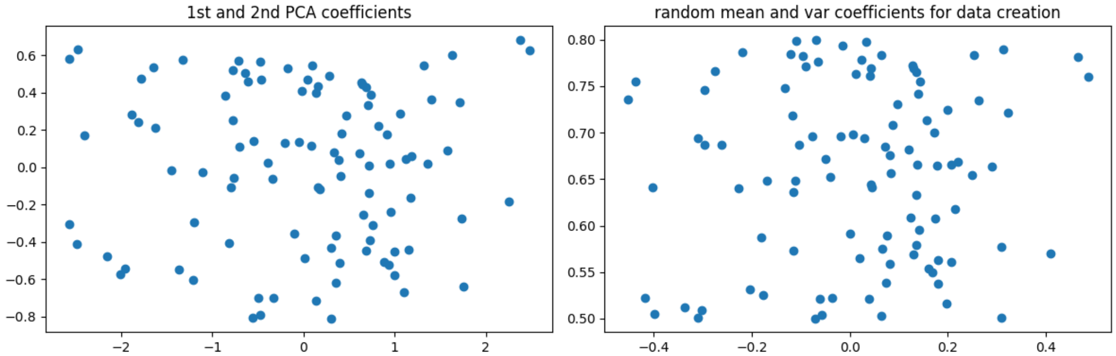
[29]:
```
# plot PCA spectrum, note: decays very quickly
fig=plt.figure(figsize=(8,4))
fig.add_subplot(1,2,1)
plt.title("PCA spectrum")
plt.plot(eigval,lw=0,marker="x")
plt.yscale("log")
```

```
fig.add_subplot(1,2,2)
plt.title("cumulative variance")
plt.plot(np.cumsum(eigval)/np.sum(eigval),lw=1,marker="x")
plt.xlim([0,5])

plt.tight_layout()
plt.show()
```



[30]:
```
# projection of samples onto eigenbasis
coef=np.einsum(eigvec,[0,1],data,[2,1],[2,0])
```

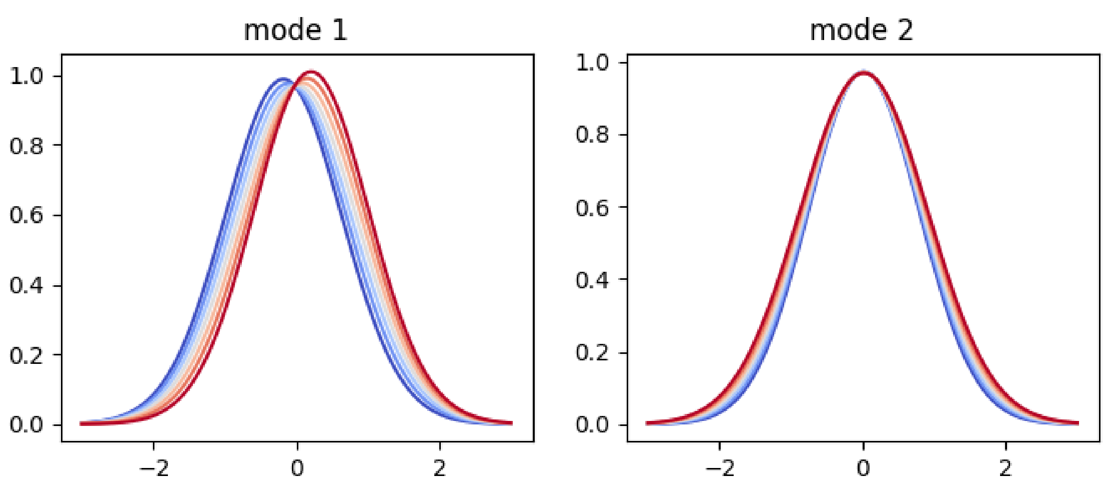
[31]:
```
fig=plt.figure(figsize=(12,4))
fig.add_subplot(1,2,1)
plt.title("1st and 2nd PCA coefficients")
plt.scatter(coef[:,0],coef[:,1])

fig.add_subplot(1,2,2)
plt.title("random mean and var coefficients for data creation")
plt.scatter(listMean,listVar)

plt.tight_layout()
plt.show()
```



visualize eigenmodes by projecting back to original sample space

- each eigenvector v_i can again be interpreted as function
- add this function to mean m, with different scales t, visualize collection of curves $m + t \cdot v_i$ where t should vary approximately according to the standard deviation $\sqrt{\lambda_i}$ along v_i
- in this way we can try and interpret 'meaning' of eigenvectors

[33]:
```
fig=plt.figure(figsize=(2*4,3))
for i in range(2):
    fig.add_subplot(1,2,i+1)
    plt.title("mode {:d}".format(i+1))
    scale=eigval[i]**0.5
    vList=np.linspace(-1,1,num=7)
    dataRe=np.einsum(eigvec[i],[0],vList*scale,[1],[1,0])
    for v,y in zip(vList,dataRe):
        plt.plot(x,y+mean,c=cm.coolwarm(0.5*v+0.5))
plt.show()
```



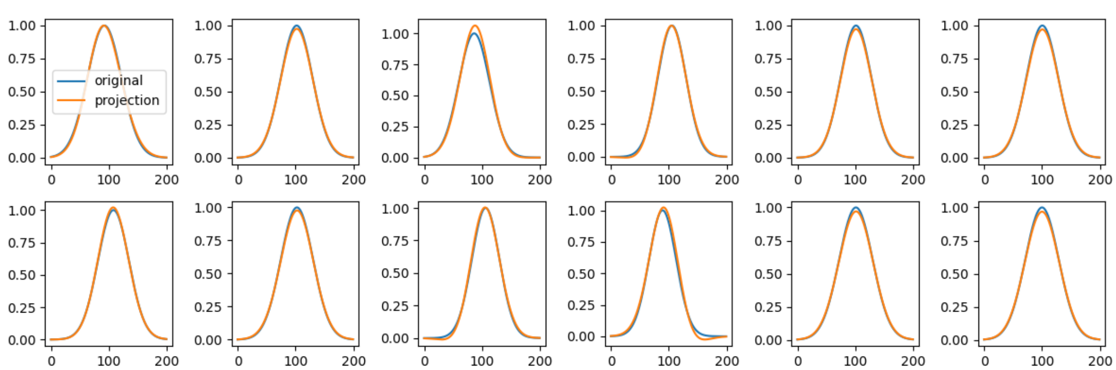
Visualize projections of samples onto eigenbasis

- the eigenvectors $(v_i)_{i=1}^n$ obtained by the PCA decomposition provide a full orthonormal basis of the sample space, each sample x can be written as

$$x = \sum_{i=1}^{n} v_i (v_i^\top x)$$

- sorting this sum from largest to smallest eigenvalue, and truncating after a few entries, we can project samples to the lower-dimensional reduced space. this gives a good qualitative feeling of how well a given basis can represent the samples
- careful: in the above formula we ignored the 'centering' of the data. x represents the centered sample. so for good visualization we need to add the mean again afterwards

[34]:
```python
# how many basis vectors to use
keep=2
# the values v_i^\top x are already stored in the coef array
# now combine the first eigenvectors and the coef array
# np.einsum is really powerful, indices can sometimes be a bit tricky
# dont forget to add mean in the end
dataRed=np.einsum(eigvec[:keep,:],[0,1],coef[:,:keep],[2,0],[2,1])+mean
# for a few of them plot original sample and projection for comparison
fig=plt.figure(figsize=(12,4))
for i in range(12):
    fig.add_subplot(2,6,i+1)
    plt.plot(data[i,:]+mean,label="original")
    plt.plot(dataRed[i,:],label="projection")
    if i==0:
        plt.legend()
plt.tight_layout()
plt.show()
```



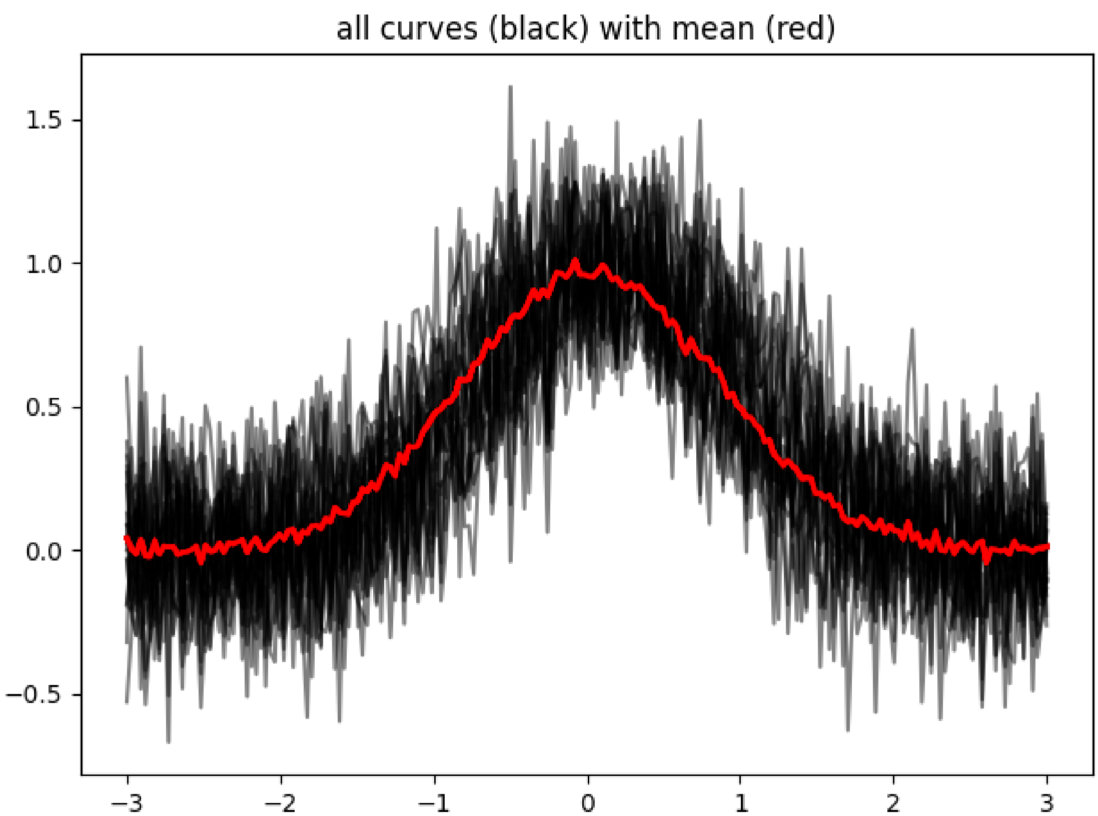
1.3.3 Retry with other data

Add random noise

- this still works, if noise has variance below data; otherwise we are in trouble

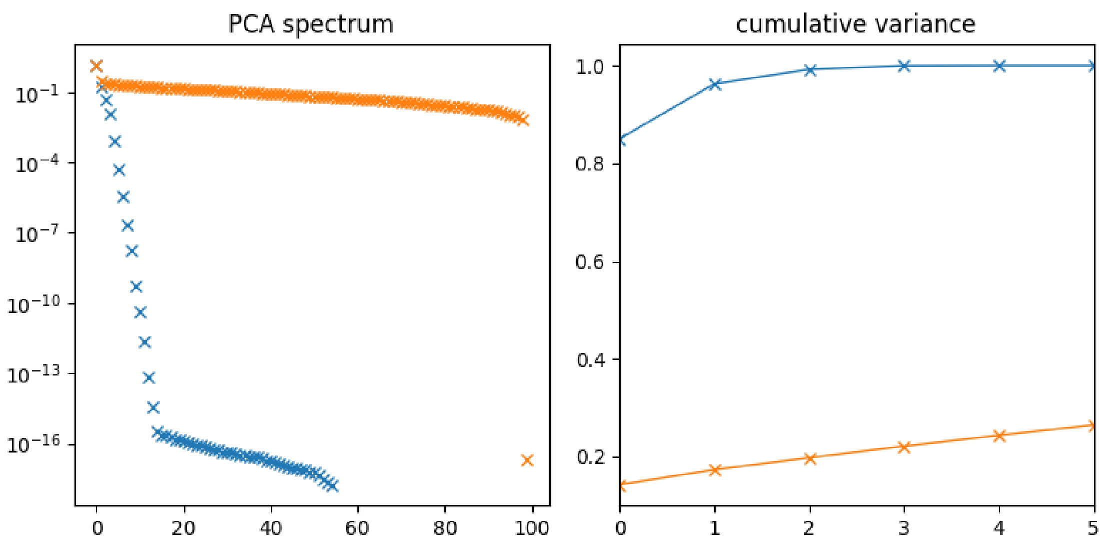
[42]:
```
nPts=100
nGrid=200
x=np.linspace(-3,3,num=nGrid)
listMean=.2*np.random.normal(size=nPts)
listVar=0.3*np.random.random(size=nPts)+0.5
listY=np.array([np.exp(-0.5*(x-mean)**2/var) for mean,var in zip(listMean,listVar)])
listY2=listY+0.2*np.random.normal(size=listY.shape)
mean2=np.mean(listY2,axis=0)
data2=listY2-np.reshape(mean2,(1,-1))
```

[43]:
```
# visualize all functions/vectors and the mean, as simple line plots
%matplotlib inline
plt.title("all curves (black) with mean (red)")
for y in listY2[:20]:
    plt.plot(x,y,c="k",alpha=0.5)
    plt.plot(x,mean2,c="r",lw=2,zorder=2)
plt.tight_layout()
plt.show()
```



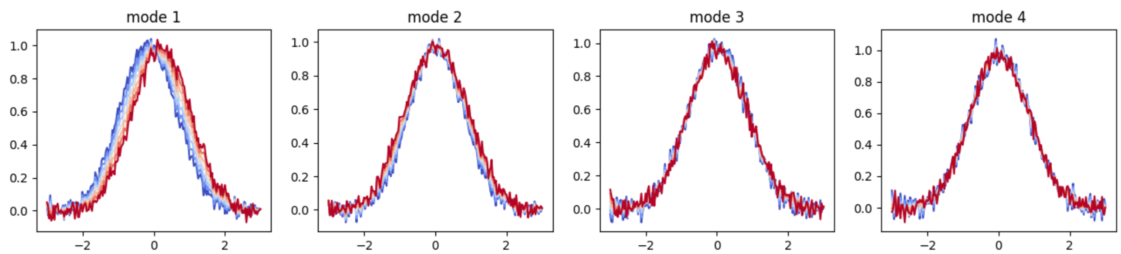
[44]:
```
# PCA and projected coefficients
eigval2,eigvec2=PCA(data2)
coef2=np.einsum(eigvec2,[0,1],data2,[2,1],[2,0])
```

[45]:
```
# plot spectrum
fig=plt.figure(figsize=(8,4))
fig.add_subplot(1,2,1)
plt.title("PCA spectrum")
plt.plot(eigval,lw=0,marker="x")
plt.plot(eigval2,lw=0,marker="x")
plt.yscale("log")

fig.add_subplot(1,2,2)
plt.title("cumulative variance")
plt.plot(np.cumsum(eigval)/np.sum(eigval),lw=1,marker="x")
plt.plot(np.cumsum(eigval2)/np.sum(eigval2),lw=1,marker="x")
plt.xlim([0,5])

plt.tight_layout()
plt.show()
```

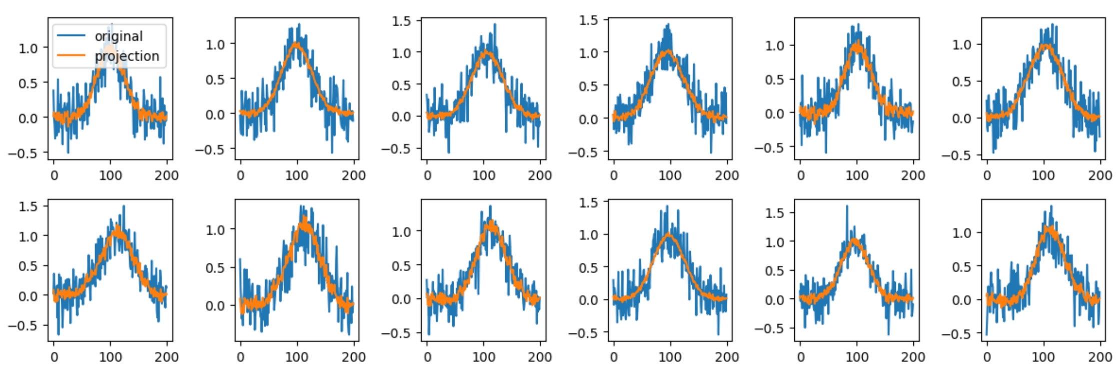


[46]:
```
# shooting along modes
nModes=4
fig=plt.figure(figsize=(nModes*4,3))
for i in range(nModes):
    fig.add_subplot(1,nModes,i+1)
    plt.title("mode {:d}".format(i+1))
    scale=eigval2[i]**0.5
    vList=np.linspace(-1,1,num=7)
```

```
    dataRe=np.einsum(eigvec2[i],[0],vList*scale,[1],[1,0])
    for v,y in zip(vList,dataRe):
        plt.plot(x,y+mean2,c=cm.coolwarm(0.5*v+0.5))
plt.show()
```



[47]:
```
# projection onto basis
keep=2
dataRed2=np.einsum(eigvec2[:keep,:],[0,1],coef2[:,:keep],[2,0],[2,1])+mean2
# for a few of them plot original sample and projection for comparison
fig=plt.figure(figsize=(12,4))
for i in range(12):
    fig.add_subplot(2,6,i+1)
    plt.plot(data2[i,:]+mean2,label="original")
    plt.plot(dataRed2[i,:],label="projection")
    if i==0:
        plt.legend()
plt.tight_layout()
plt.show()
```

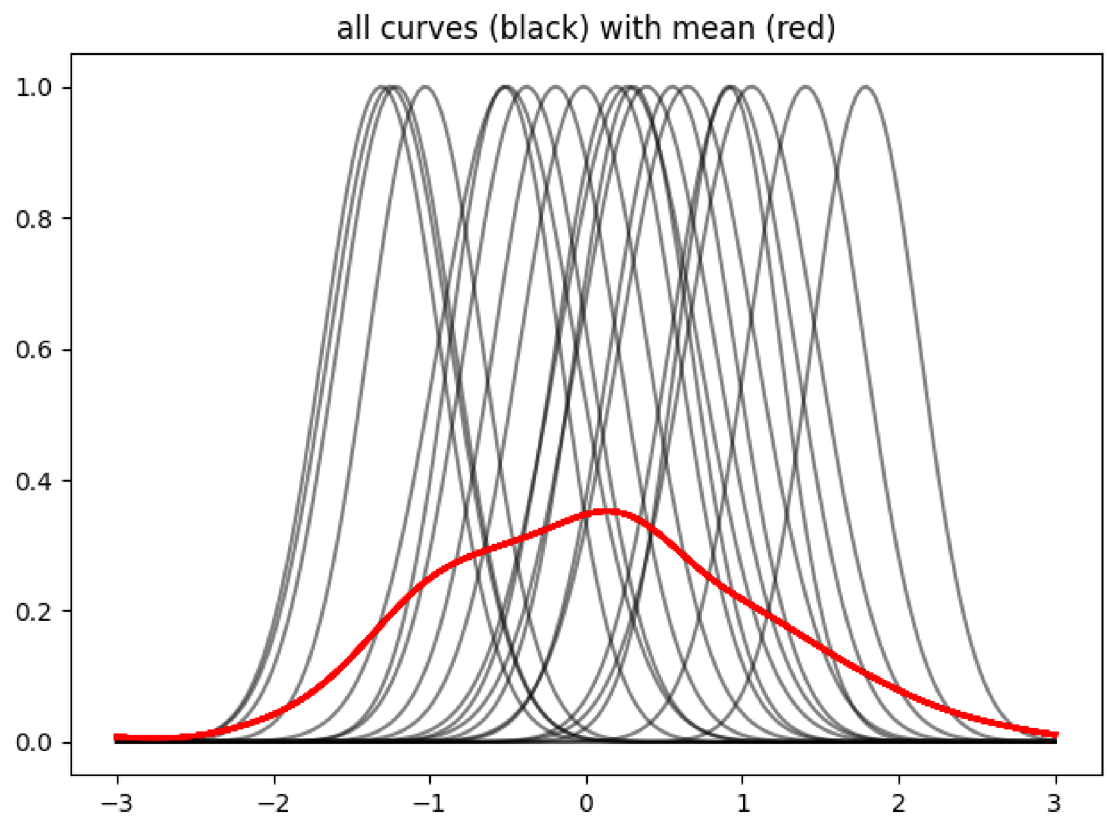


Much less overlap (lower variance, higher fluctuation in mean)

- in this case the standard Euclidean metric on the data samples is no longer informative
- good metrics are subject of ongoing research

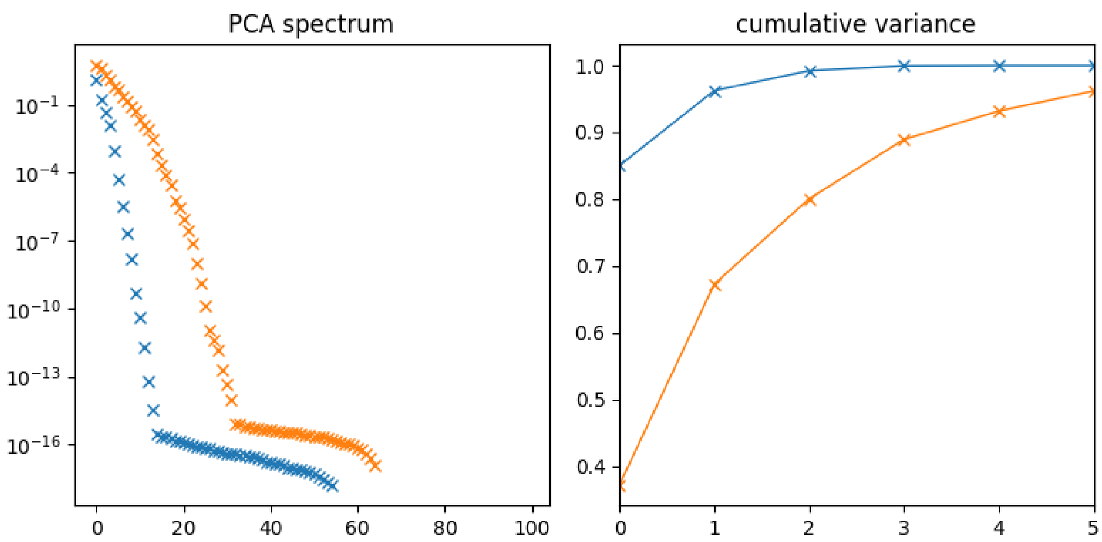
[49]:
```
nPts=100
nGrid=200
x=np.linspace(-3,3,num=nGrid)
listMean=1.*np.random.normal(size=nPts)
listVar=0.1*np.random.random(size=nPts)+0.1
listY2=np.array([np.exp(-0.5*(x-mean)**2/var) for mean,var in␣
 ↪zip(listMean,listVar)])
mean2=np.mean(listY2,axis=0)
data2=listY2-np.reshape(mean2,(1,-1))
```

[50]:
```
# visualize all functions/vectors and the mean, as simple line plots
%matplotlib inline
plt.title("all curves (black) with mean (red)")
for y in listY2[:20]:
    plt.plot(x,y,c="k",alpha=0.5)
    plt.plot(x,mean2,c="r",lw=2,zorder=2)
plt.tight_layout()
plt.show()
```



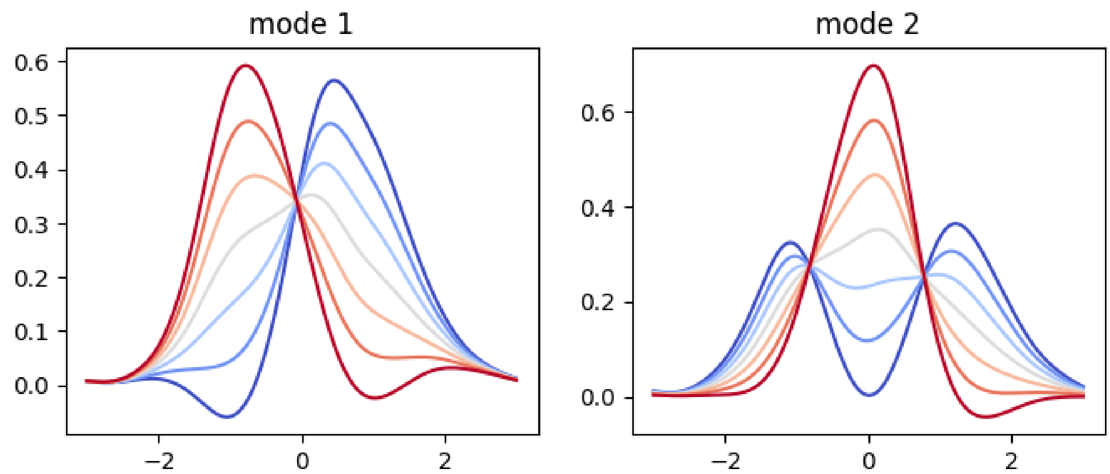
[51]:
```
# PCA and projected coefficients
eigval2,eigvec2=PCA(data2)
coef2=np.einsum(eigvec2,[0,1],data2,[2,1],[2,0])
```

[52]:
```
# plot spectrum
fig=plt.figure(figsize=(8,4))
fig.add_subplot(1,2,1)
plt.title("PCA spectrum")
plt.plot(eigval,lw=0,marker="x")
plt.plot(eigval2,lw=0,marker="x")
plt.yscale("log")

fig.add_subplot(1,2,2)
plt.title("cumulative variance")
plt.plot(np.cumsum(eigval)/np.sum(eigval),lw=1,marker="x")
plt.plot(np.cumsum(eigval2)/np.sum(eigval2),lw=1,marker="x")
plt.xlim([0,5])

plt.tight_layout()
plt.show()
```



[53]:
```
# shooting along modes
fig=plt.figure(figsize=(2*4,3))
for i in range(2):
    fig.add_subplot(1,2,i+1)
    plt.title("mode {:d}".format(i+1))
    scale=eigval2[i]**0.5
    vList=np.linspace(-1,1,num=7)
    dataRe=np.einsum(eigvec2[i],[0],vList*scale,[1],[1,0])
```

```
    for v,y in zip(vList,dataRe):
        plt.plot(x,y+mean2,c=cm.coolwarm(0.5*v+0.5))
plt.show()
```



[ ]:
```
# projection onto basis
keep=2
dataRed2=np.einsum(eigvec2[:keep,:],[0,1],coef2[:,:keep],[2,0],[2,1])+mean2
# for a few of them plot original sample and projection for comparison
fig=plt.figure(figsize=(12,4))
for i in range(12):
    fig.add_subplot(2,6,i+1)
    plt.plot(data2[i,:]+mean2,label="original")
    plt.plot(dataRed2[i,:],label="projection")
    if i==0:
        plt.legend()
plt.tight_layout()
plt.show()
```

[ ]:
```

```