Machine Intelligence II SoSe 2016 Exercise 7

The Nebenhoerers: Danijar Hafner, Thomas Kellermeier, Patrick Kuhn, Jan Szynal

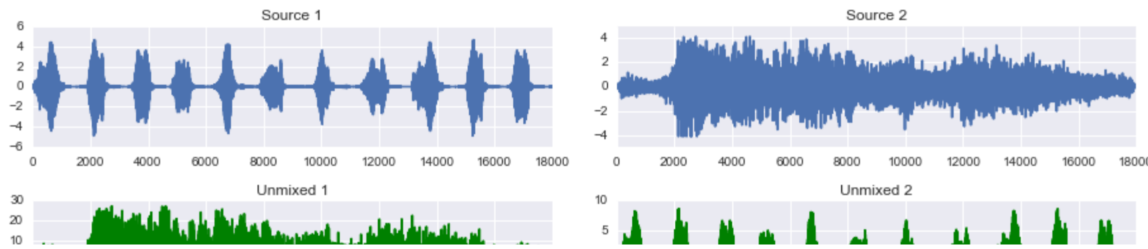
In [3]:
```
%matplotlib inline

import scipy.io
import scipy.io.wavfile
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import collections
```

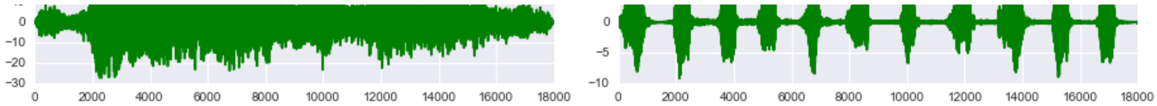
7.1 Natural Gradient

In [4]:
```
# From previous exercise
def sigmoid(y):
    return 1 / (1 + np.exp(-y))

def psi(y):
    return 1 - 2 * sigmoid(y)

def update_natural(W, x):
    n = x.shape[0]
    phee = psi(W.dot(x)).reshape(n, 1)
    delta_W = np.dot(phee.dot(np.dot(W, x).reshape(1, n)), W)
    delta_W = delta_W + W  # multiplied out delta function
    for i in range(n):  # Bell-Sejnowski solution
        delta_W[i, i] = 0
    return delta_W

def plot(ax, data, **kwargs):
    ax.plot(data, **kwargs)
    ax.set_title(kwargs['label'])
    scipy.io.wavfile.write(kwargs['label'] + '.wav', 8192, data),

def online_ica(X, X0, lambda_= 0.99, epsilon = 0.001, eta = 0.15):
    n = X.shape[0]  # Number of sources
    W = np.linalg.inv(np.random.RandomState(seed+1).rand(n, n))
    for i in range(n):  # Bell-Sejnowski solution
        W[i, i] = 1
    time = 0
    while eta > epsilon:
        example = X.T[time % X.shape[1]]
        eta = eta * lambda_
        W += eta * update_natural(W, example)
        time += 1
    print("Calculated unmixing matrix in {} steps".format(time))
    return W.dot(X0)

seed = 13  # seed for random states to get always the same result
sound1 = np.loadtxt('sounds/sound1.dat')
sound2 = np.loadtxt('sounds/sound2.dat')
```

In [5]:
```
# 7.1. (a) Online ICA with natural gradient decaying slowly to 0
sounds = np.concatenate([[sound1, sound2]], axis=1)
A = np.linalg.inv(np.random.RandomState(seed+4).rand(2,2))
X0 = A.dot(sounds)
X = X0[:,np.random.RandomState(seed+1).permutation(X0.shape[1])]
X -= X.mean(axis=1).reshape((2, 1))
unmixed_nat = online_ica(X, X0)

fig, ax = plt.subplots(2, 2, figsize=(13, 4))
plot(ax[0, 0], sound1, label='Source 1')
plot(ax[0, 1], sound2, label='Source 2')
plot(ax[1, 0], unmixed_nat[0,:], label='Unmixed 1', color='green')
plot(ax[1, 1], unmixed_nat[1,:], label='Unmixed 2', color='green')
fig.tight_layout()
```

```
Calculated unmixing matrix in 499 steps
```

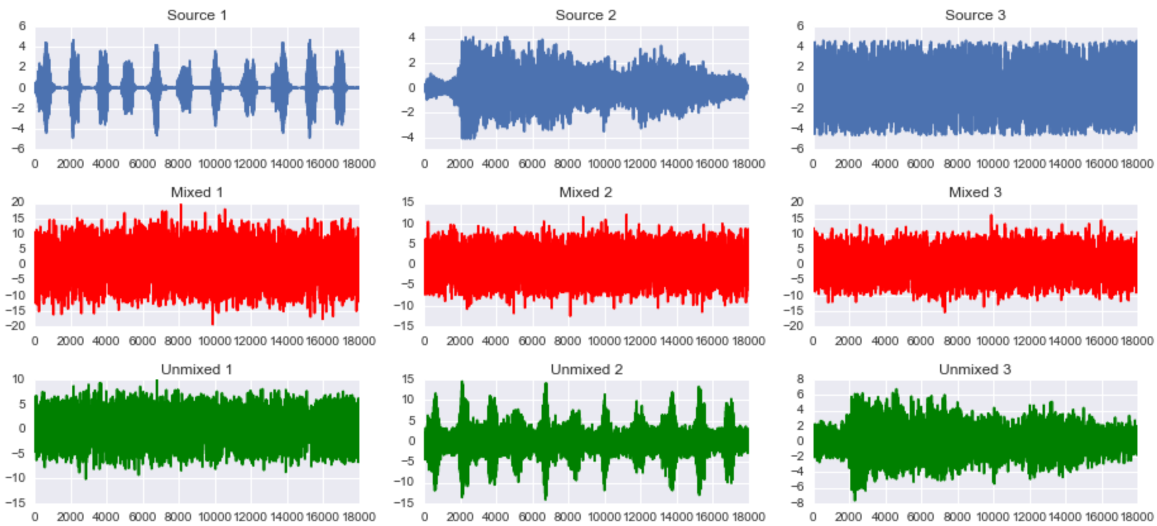




In [8]:
```
def laplace_rand(*shape):
    return np.random.RandomState(seed).laplace(size=np.array(shape).prod()).reshape(*shape)

def normally_rand(*shape):
    result = np.random.RandomState(seed).rand(*shape) * sound1.max()
    result[::2] *= -1
    return result

def generate_sample(sound, rand_gen=normally_rand):
    new_sound = rand_gen(*sound.shape)
    std = sound.std()
    std_rounded = int(std * 1000) / 1000
    std_n = 0
    steps = 0
    while int(std_n * 1000) / 1000 != std_rounded:
        pos = np.random.RandomState(steps).randint(new_sound.shape[0])
        if std_n > std:
            new_sound[pos] = new_sound.mean()
        else:
            new_sound[pos] += std
        std_n = new_sound.std()
        steps += 1
    print("Generating third sample took {} steps".format(steps))
    return new_sound
```

In [9]:
```
# 7.1 (b)
sound3 = generate_sample(sound1)
sounds = np.concatenate([[sound1, sound2, sound3]], axis=1)
A = np.linalg.inv(np.random.RandomState(seed).rand(3,3))
X0 = A.dot(sounds)
X = X0[:,np.random.RandomState(seed+1).permutation(X0.shape[1])]
X -= X.mean(axis=1).reshape((3, 1))
unmixed3_nat = online_ica(X, X0, lambda_ = 0.999, eta = 0.06, epsilon=0.00001)
# The result might be too loud
for result in unmixed3_nat:
    if result.max() > 15:
        result /= result.max() / 10

fig, ax = plt.subplots(3, 3, figsize=(13, 6))
plot(ax[0, 0], sound1, label='Source 1')
plot(ax[0, 1], sound2, label='Source 2')
plot(ax[0, 2], sound3, label='Source 3')
plot(ax[1, 0], X[0, :], label='Mixed 1', color='red')
plot(ax[1, 1], X[1, :], label='Mixed 2', color='red')
plot(ax[1, 2], X[2, :], label='Mixed 3', color='red')
plot(ax[2, 0], unmixed3_nat[0,:], label='Unmixed 1', color='green')
plot(ax[2, 1], unmixed3_nat[1,:], label='Unmixed 2', color='green')
plot(ax[2, 2], unmixed3_nat[2,:], label='Unmixed 3', color='green')
fig.tight_layout()
```

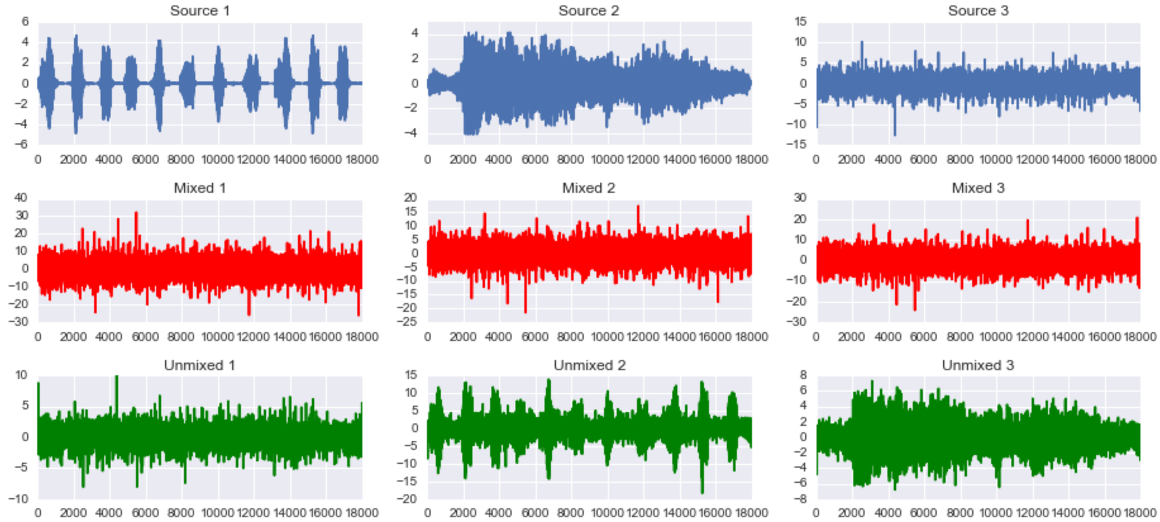
```
Generating third sample took 34926 steps
Calculated unmixing matrix in 8696 steps
```



In [11]:
```
# 7.1 (c)
sound3 = generate_sample(sound1, rand_gen=laplace_rand)
sounds = np.concatenate([[sound1, sound2, sound3]], axis=1)
A = np.linalg.inv(np.random.RandomState(seed).rand(3,3))
X0 = A.dot(sounds)
X = X0[:,np.random.RandomState(seed+1).permutation(X0.shape[1])]
```

```
X -= X.mean(axis=1).reshape((3, 1))
unmixed3_nat = online_ica(X, X0, lambda_ = 0.9999, eta = 0.05, epsilon=0.00001)
# The result might be too loud
for result in unmixed3_nat:
    if result.max() > 15:
        result /= result.max() / 10

fig, ax = plt.subplots(3, 3, figsize=(13, 6))
plot(ax[0, 0], sound1, label='Source 1')
plot(ax[0, 1], sound2, label='Source 2')
plot(ax[0, 2], sound3, label='Source 3')
plot(ax[1, 0], X[0, :], label='Mixed 1', color='red')
plot(ax[1, 1], X[1, :], label='Mixed 2', color='red')
plot(ax[1, 2], X[2, :], label='Mixed 3', color='red')
plot(ax[2, 0], unmixed3_nat[0,:], label='Unmixed 1', color='green')
plot(ax[2, 1], unmixed3_nat[1,:], label='Unmixed 2', color='green')
plot(ax[2, 2], unmixed3_nat[2,:], label='Unmixed 3', color='green')
fig.tight_layout()
```

```
Generating third sample took 12538 steps
Calculated unmixing matrix in 85168 steps
```



7.3 Kurtosis of Toy Data

In [15]:
```
mat = scipy.io.loadmat('distrib.mat')
uniform = mat['uniform']
normal = mat['normal']
laplacian = mat['laplacian']

A = np.array([[4,3],[2,1]])
```

In [16]:
```
def plot_dataset(data, title='', xlabel='Source 1', ylabel='Source 2', zoom=1):
    df = pd.DataFrame(data.T, columns=[xlabel, ylabel])
    g = sns.jointplot(x=xlabel, y=ylabel, data=df, xlim=[-40/zoom,40/zoom], ylim=
[-40/zoom,40/zoom],size=7)
    zoomf = lambda a, b: zoom
    g = g.annotate(zoomf,template="{stat}: {val:.1f}", stat="zoom", loc="lower left", fontsize=16)
    sns.plt.suptitle(title, fontsize=20, y=1.08)
```

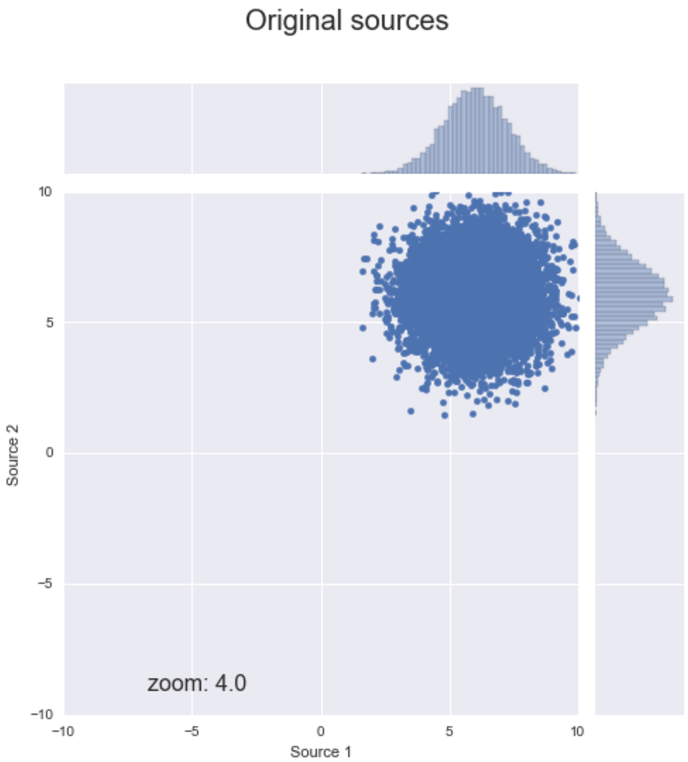
In [17]:
```
def plot_kurtosis(angs, kurts, title='Kurtosis'):
    plt.figure()
    sns.set_style('darkgrid')
    plt.plot(angs,kurts[0,:],label="First dimension")
    plt.plot(angs,kurts[1,:],label="Second dimension")
    plt.xlim(0,6.28)
    #plt.ylim(-0.1,0.05)
    plt.legend()
    plt.suptitle('Kurtosis values by angle', fontsize=20)
```

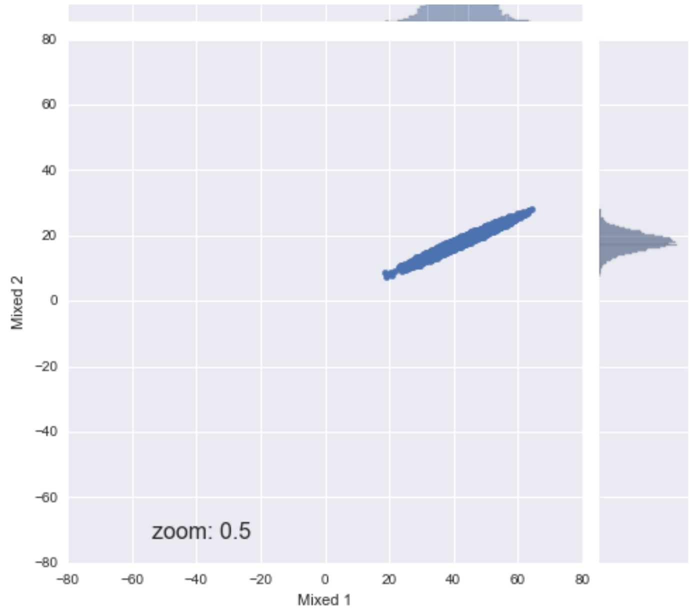
In [22]:
```
def procedure(s):
    #Plot the original sources
    plot_dataset(s, title='Original sources', zoom=4)

    #7.3a - Apply mixing matrix A and plot mixed data
    x = np.dot(A,s)
    plot_dataset(x, title='After mixing', xlabel='Mixed 1', ylabel='Mixed 2', zoom=0.5)

    #7.3b - Center to mean 0 and plot centered data
    x = x - np.mean(x,axis=1).reshape(2,1)
    plot_dataset(x, title='After centering', xlabel='Centered 1', ylabel='Centered 2')

    #7.3c - Decorrelate by PCA and project onto the principal components
```

```
    # (consult: 1. eig vs eigh 2. should we sort eigvals? 3. eigvecs transposed?)
    covmat = np.cov(x)
    eigvals, eigvecs = np.linalg.eig(covmat)
    p = np.dot(eigvecs.T, x)
    plot_dataset(p, title='Projected onto PCs', xlabel='PC 1', ylabel='PC 2')

    #7.3d - Whiten data (scale to unit variance)
    p = p / np.std(p,axis=1).reshape(2,1)
    plot_dataset(p, title='Sphered projection', xlabel='PC 1', ylabel='PC 2', zoom=9)


    #7.3e - Calculate and plot kurtoses - for both dimensions, for rotations by 100 angle values i
n range [0,2pi]
    angs = np.pi*np.arange(0,100,1) / 50
    kurts = np.zeros([2,100])

    for i,a in enumerate(angs):
        Ro = np.array([[np.cos(a),-np.sin(a)],[np.sin(a),np.cos(a)]])
        po = np.dot(Ro,p)
        kurt = (np.sum(po**4, axis=1).reshape(2,1) / p.shape[1]) - 3
        kurts[:,i] = kurt.T

    #7.3f - Find the angles for which the kurtosis value is the largest and the smallest, rotate b
y them
    ang_kurtmax = angs[np.argmax(kurts[0,:])]
    ang_kurtmin = angs[np.argmin(kurts[0,:])]

    a = ang_kurtmax
    Ro = np.array([[np.cos(a),-np.sin(a)],[np.sin(a),np.cos(a)]])
    po = np.dot(Ro,p)
    plot_dataset(po, title='Rotated for maximal kurtosis', xlabel='PC 1', ylabel='PC 2', zoom=9)

    a = ang_kurtmin
    Ro = np.array([[np.cos(a),-np.sin(a)],[np.sin(a),np.cos(a)]])
    po = np.dot(Ro,p)
    plot_dataset(po, title='Rotated for minimal kurtosis', xlabel='PC 1', ylabel='PC 2', zoom=9)

    plot_kurtosis(angs,kurts)

    #this line causes an interesting error
    #plot_dataset(po, title='Rotated for maximal kurtosis', xlabel='a', ylabel='a')
```

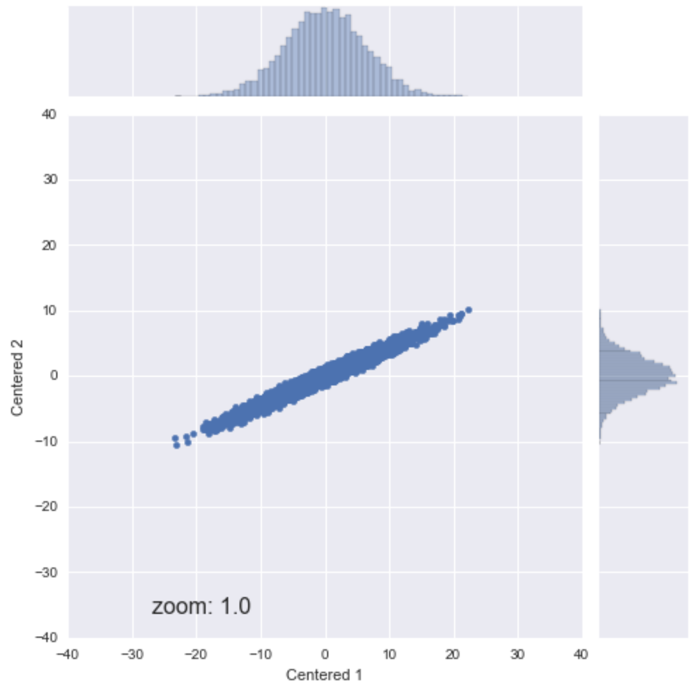
7.3 (I). For a normal distribution

In [23]:
```
procedure(normal)
```

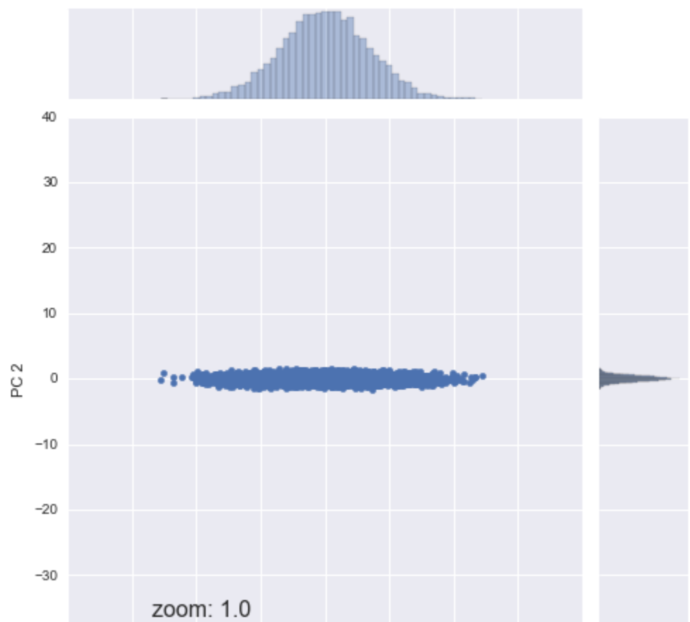


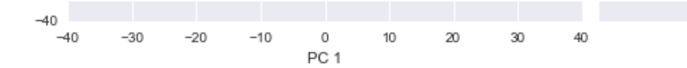


After centering

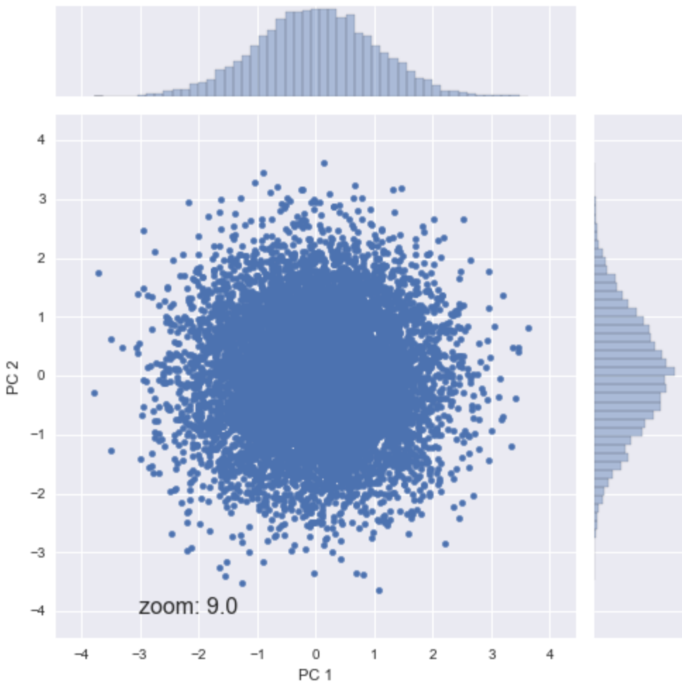


Projected onto PCs

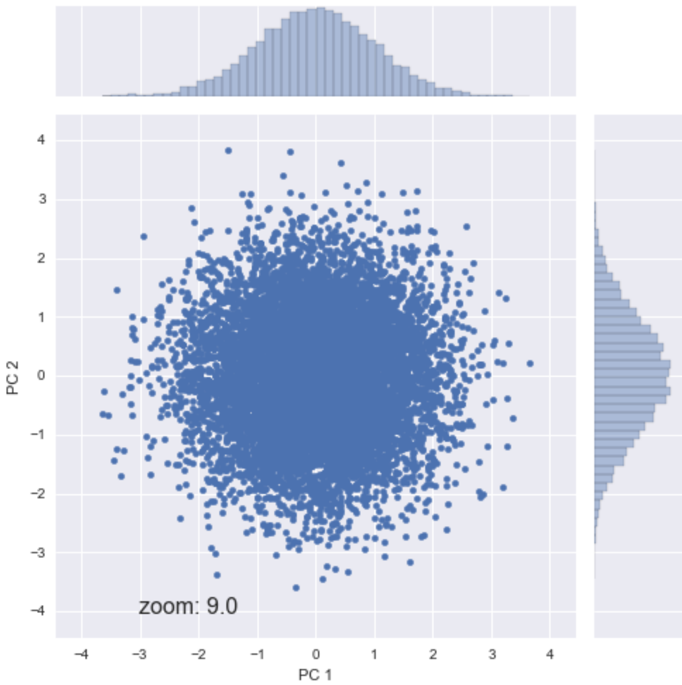




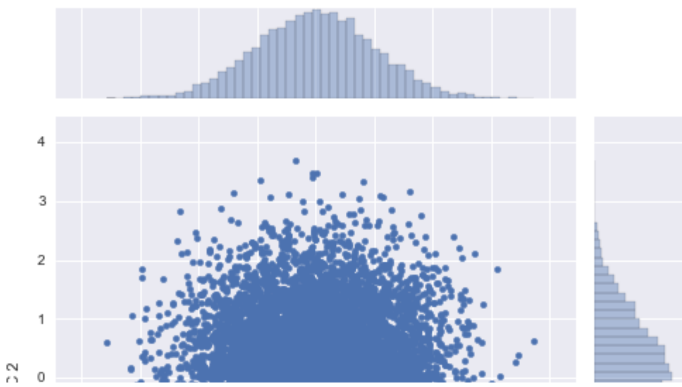
Sphered projection

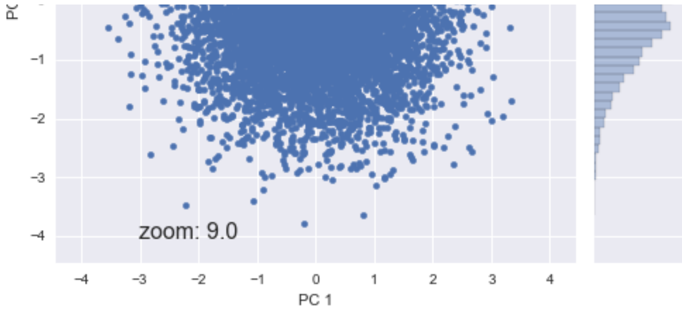


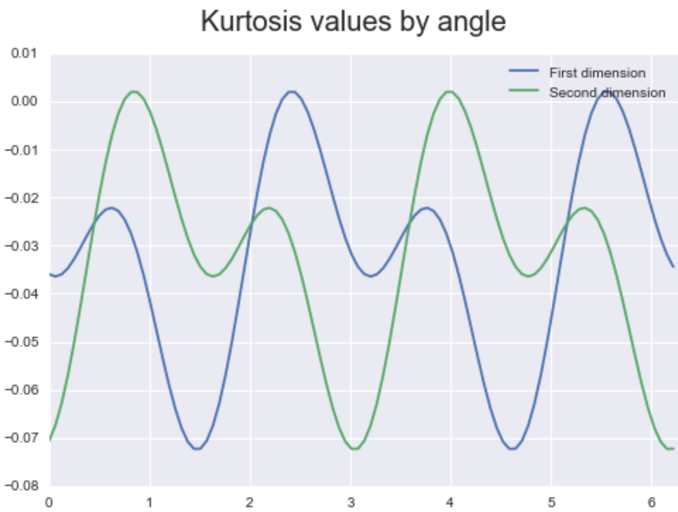
Rotated for maximal kurtosis



Rotated for minimal kurtosis

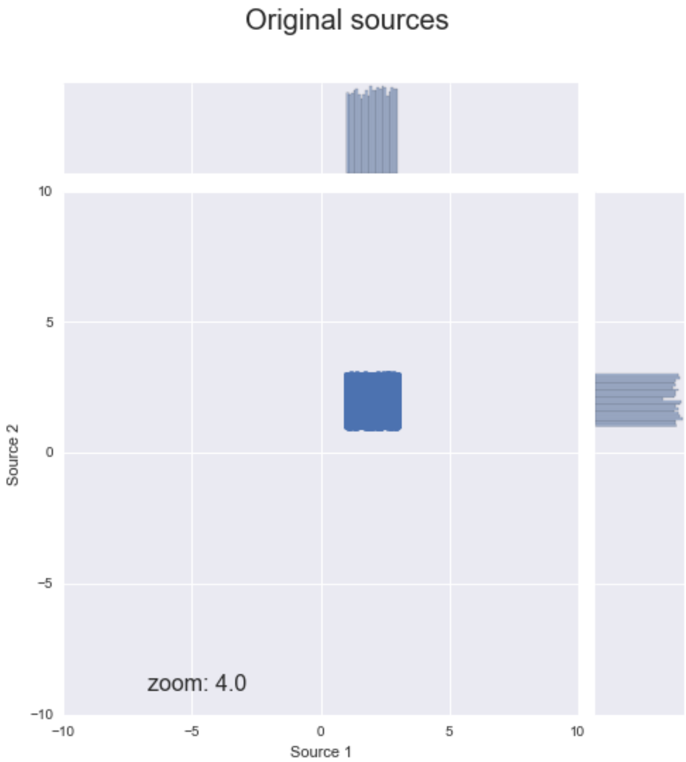


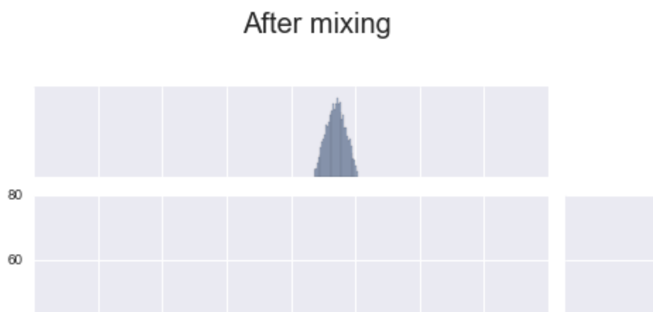


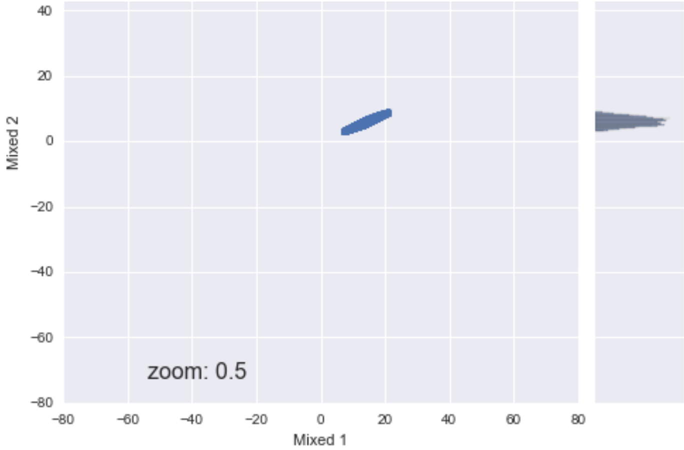


7.3 (I). For a uniform distribution

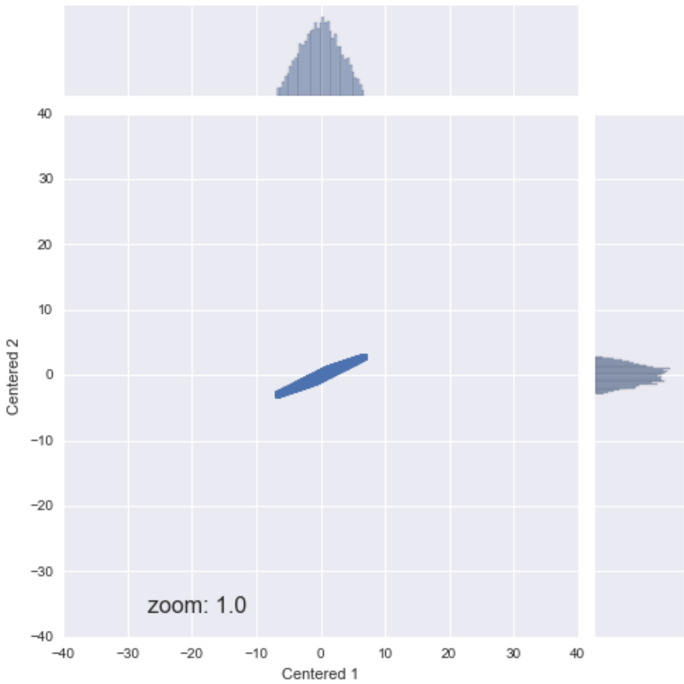
In [24]:
```
procedure(uniform)
```



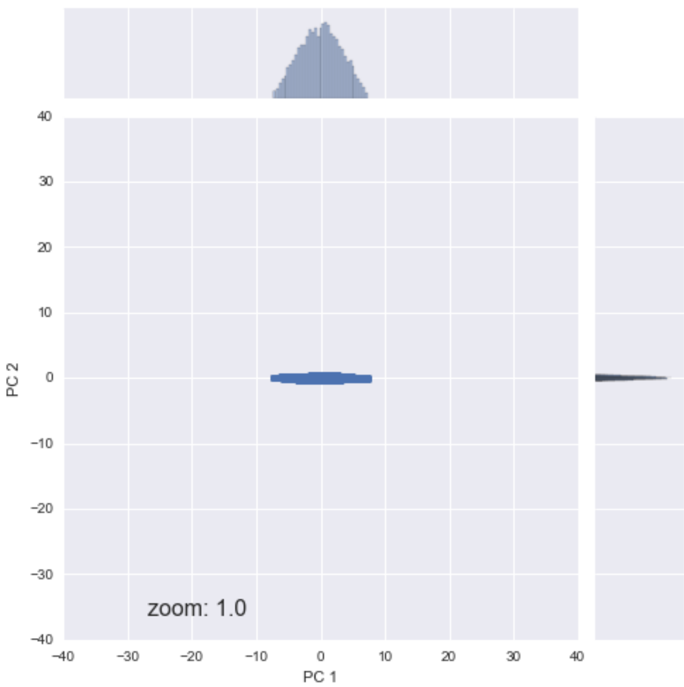




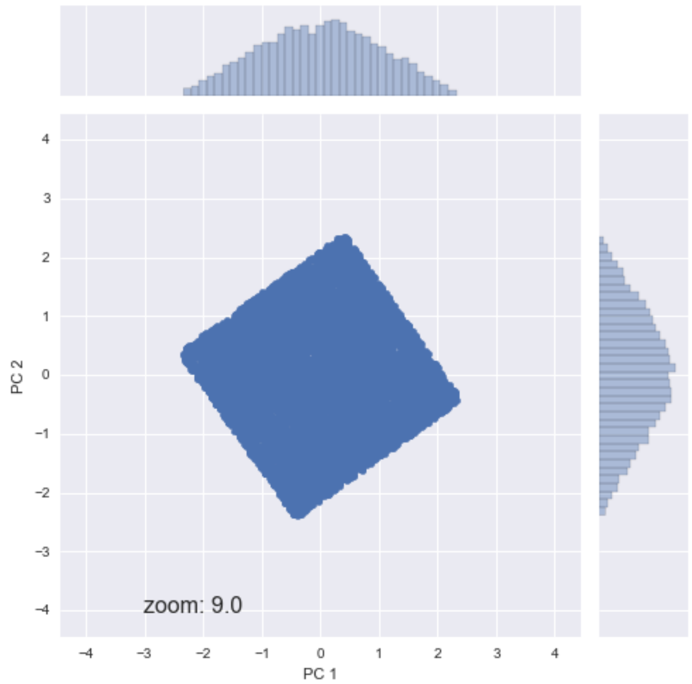
After centering



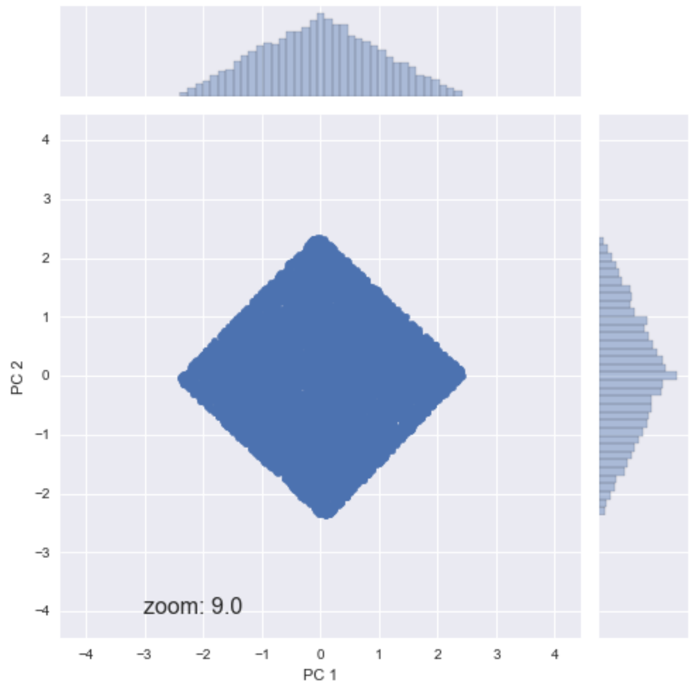
Projected onto PCs



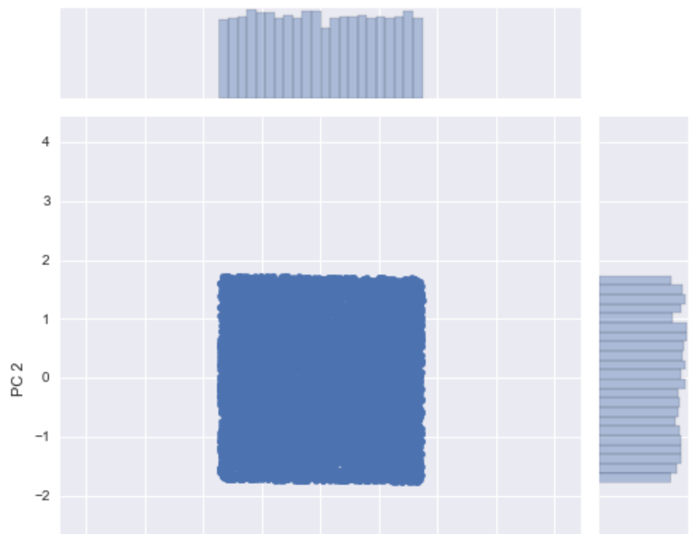
Sphered projection

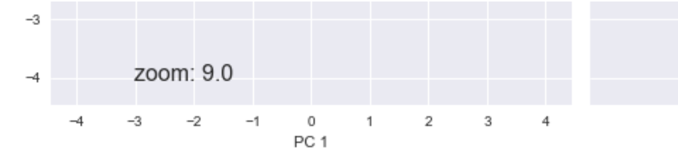


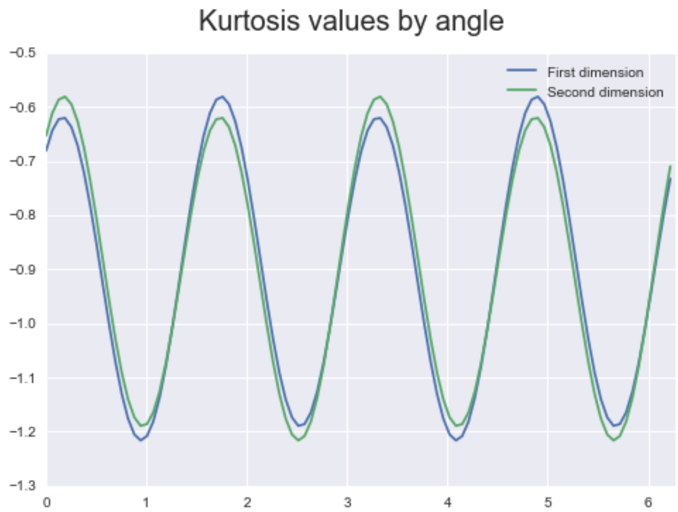
Rotated for maximal kurtosis



Rotated for minimal kurtosis

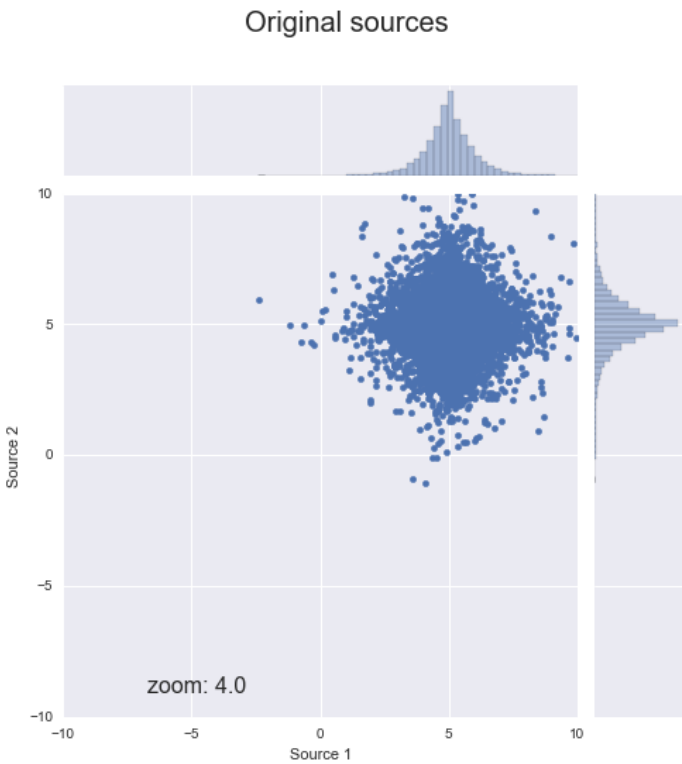


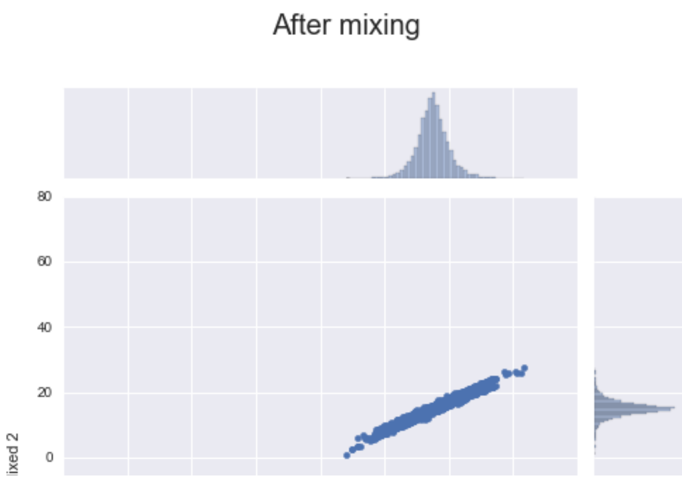


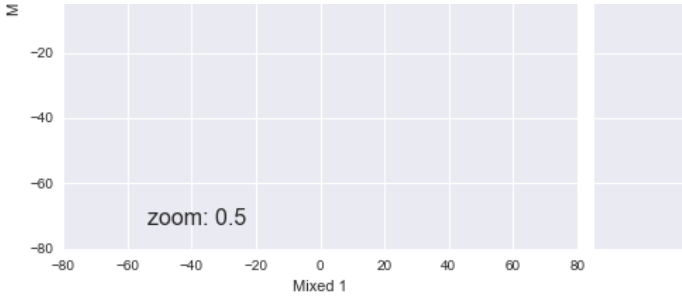


7.3 (l). For a laplacian distribution

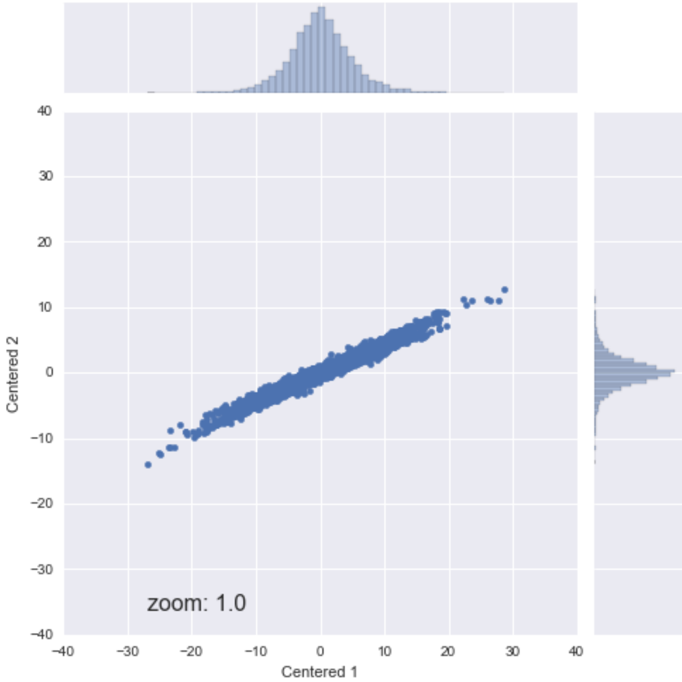
In [25]:
```
procedure(laplacian)
```



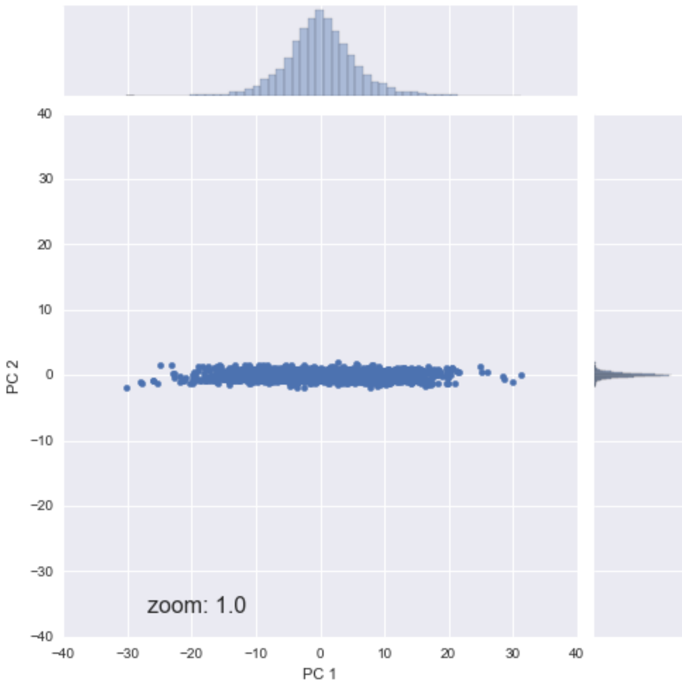




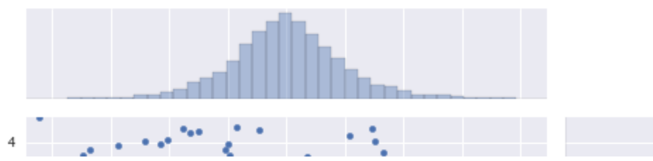
After centering

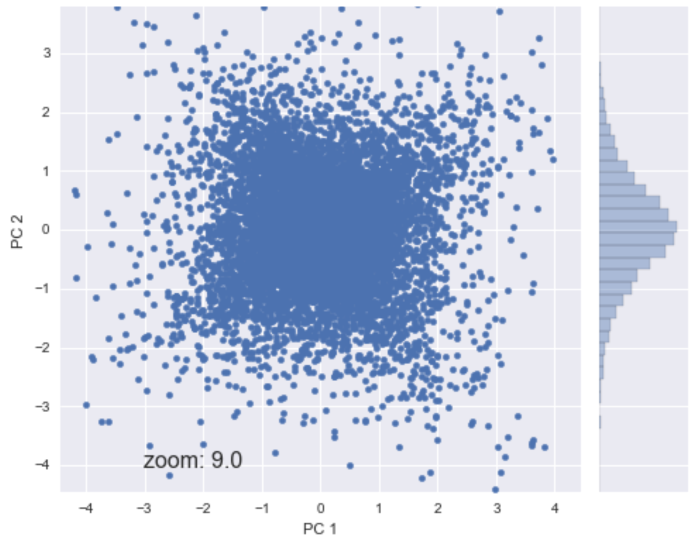


Projected onto PCs

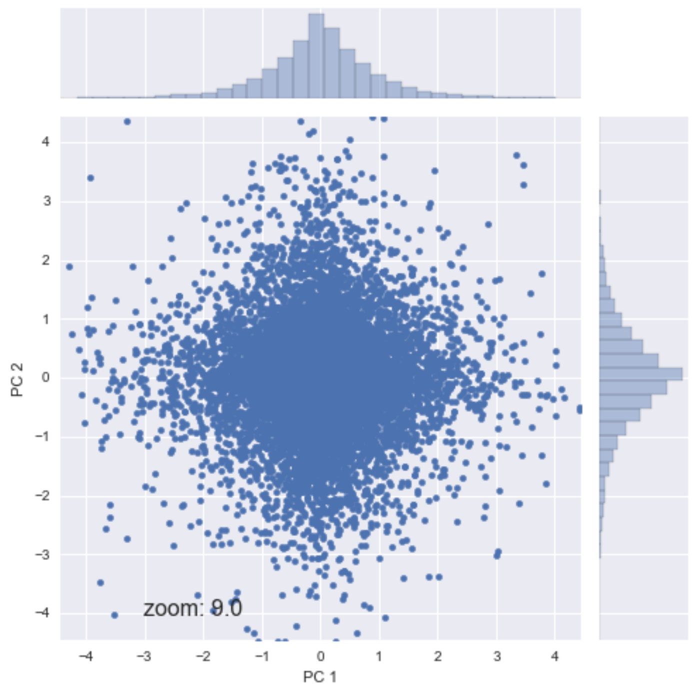


Sphered projection

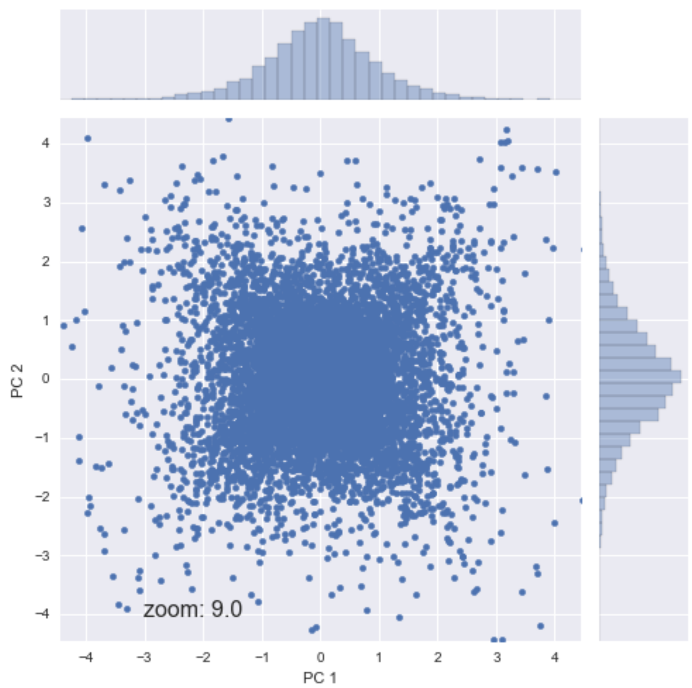


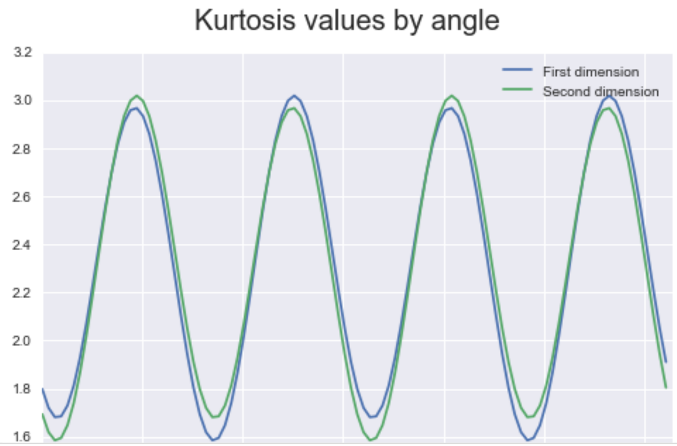


Rotated for maximal kurtosis



Rotated for minimal kurtosis




Kurtosis values by angle
First dimension
Second dimension
3.2
3.0
2.8
2.6
2.4
2.2
2.0
1.8
1.6

Uniform
Using the definition of central moments directly:
$\int_a^b (x-\mu)^k * \frac{1}{b-a} dx$
$\frac{1}{b-a} * \frac{1}{k+1}((b-\mu)^{k+1} - (a-\mu)^{k+1}))$
Since the mean of the uniform distribution is $\frac{a+b}{2}$
$\frac{1}{b-a} * \frac{1}{k+1}((\frac{b-a}{2})^{k+1} - (-\frac{b-a}{2})^{k+1}))$
$\frac{1}{b-a} * \frac{1}{k+1}((\frac{b-a}{2})^{k+1} - (-1)^{k+1}(\frac{b-a}{2})^{k+1}))$

This means that uneven central moments are 0.
For even central moments:
$\frac{1}{b-a} * \frac{1}{k+1}(2 * (\frac{b-a}{2})^{k+1})$
$\frac{1}{b-a} * \frac{1}{k+1}(2 * \frac{(b-a)^{k+1}}{2^{k+1}})$
$\frac{1}{k+1}(\frac{(b-a)}{2})^k$
So the first and third moment are 0. The second central moment is $\frac{1}{12} * (a-b)^2$ and the forth central moment is $\frac{1}{80} * (a-b)^4$

Gaussian
Using relation between central moments and raw moments:
$\mu_n = E[X - E[x])^n] = \sum_{j=0}^n \binom{n}{j}(-1)^n \mu_j^{'} \mu^{n-j}$

This allows to express the first four central moments as:
$\mu_1 = \mu_1^{'}$

$\mu_2 = \mu_2^{'} - \mu^2$

$\mu_3 = \mu_3^{'} - 3\mu\mu_2^{'} + 2\mu^3$

$\mu_4 = \mu_4^{'} - 4\mu\mu_3^{'} + 6\mu^2\mu_2^{'} - 3\mu^4$

So in the following first the raw moments of the distributions are retrieved using the moment generating function. $M_X(t) := \mathbf{E}[e^{tX}]$

The moment generating function of the normal distribution $f(t) = e^{t\mu + \frac{1}{2}\mu^2 * t^2}$
$f^{'}(t) = (\mu + \sigma^2 * t) * f(t)$
$f^{'}(0) = \mu * f(0) = \mu$

$f^{''}(t) = (\mu + \sigma^2 * t) * f^{'}(t) + \sigma^2 * f(t)$
$f^{''}(0) = \mu^2 + \sigma^2$

$f^{'''}(t) = 2\sigma^2 * f^{'}(t) + f^{''}(t)(\mu + \sigma^2 * t)$
$f^{'''}(0) = 2\sigma^2\mu + (\mu^2 + \sigma^2) * \mu = 3\sigma^2\mu + \mu^3$

$f^{''''}(t) = 2\sigma^2 * f^{''}(t) + f^{'''}(t) * (\mu + \sigma^2 * t) + \sigma^2 * f^{''}(t)$
$f^{''''}(t) = 3\sigma^2 * f^{''}(t) + f^{'''}(t) * (\mu + \sigma^2 * t)$
$f^{''''}(0) = 3 * \sigma^4 + 6 * \sigma^2 * \mu^2 + \mu^4$

Inserting it back into the formula for the moments:
$\mu_2 = \mu^2 + \sigma^2 - \mu^2 = \sigma^2$
$\mu_3 = 3\sigma^2\mu + \mu^3 - 3\mu(\mu^2 + \sigma^2) + 2\mu^3$
$\mu_3 = 3\sigma^2\mu + 3\mu^3 - 3\mu^3 - 3\sigma^2\mu = 0$
$\mu_4 = 3\sigma^4 + 6\sigma^2 * \mu^2 + \mu^4 - 4\mu(3\sigma^2\mu + \mu^3) + 6\mu^2(\mu^2 + \sigma^2) - 3\mu^4$
$\mu_4 = 3\sigma^4 + 6\sigma^2\mu^2 + \mu^4 - 12\mu^2\sigma^2 - 4\mu^4 + 6\mu^4 + 6\sigma^2\mu^2 - 3\mu^4$
$\mu_4 = 3\sigma^4 + \mu^4 - 4\mu^4 + 6\mu^4 - 3\mu^4$
$\mu_4 = 3\sigma^4$