Ex 1

a) $$\Lambda(s(x),\lambda_1,\lambda_2,\lambda_3) = -\int e^{s(x)}\log e^{s(x)}dx + \lambda_1\left(\int e^{s(x)}dx - 1\right) + \lambda_2\left(\int x e^{s(x)}dx\right) + \lambda_3\left(\int x^2 e^{s(x)}dx - \sigma^2\right)$$

b) $$\frac{\partial \Lambda}{\partial s(x)} = -e^{s(x)}\left(1+s(x)\right) + \lambda_1 e^{s(x)} + \lambda_2 x e^{s(x)} + \lambda_3 e^{s(x)} x^2 = 0$$

$$\Rightarrow \quad +\underbrace{e^{s(x)}}_{>0}\Big(\underbrace{-1 - s(x) + \lambda_1 + \lambda_2 x + \lambda_3 x^2}_{=0}\Big) = 0$$

$$\Rightarrow \quad s(x) = -1 + \lambda_1 + \lambda_2 x + \lambda_3 x^2$$

i) $$\frac{\partial \Lambda}{\partial \lambda_1} = \int e^{-1+\lambda_1+\lambda_2 x+\lambda_3 x^2}dx - 1 = 0$$

$$\Rightarrow \frac{\sqrt{\pi}}{\sqrt{-\lambda_3}} \cdot e^{\lambda_1 - \frac{\lambda_2^2}{4\lambda_3} - 1} = 1 \qquad (\text{assume } \lambda_3 < 0)$$ 1.1

ii) $$\frac{\partial \Lambda}{\partial \lambda_2} = \int x e^{-1+\lambda_1+\lambda_2 x+\lambda_3 x^2}dx = 0$$ 1.2

$$\Rightarrow \frac{\sqrt{\pi}\,\lambda_2}{2\sqrt{-\lambda_3^3}} \cdot e^{\lambda_1 - \frac{\lambda_2^2}{4\lambda_3} - 1} = 0$$

iii) $$\frac{\partial \Lambda}{\partial \lambda_3} = \int x^2 e^{-1+\lambda_1+\lambda_2 x+\lambda_3 x^2} - \sigma^2 = 0$$

$$\Rightarrow \frac{\sqrt{\pi}}{4\sqrt{-\lambda_3^3}} \cdot e^{\lambda_1 - \frac{\lambda_2^2}{4\lambda_3} - 1} = \sigma^2$$

From (i) we obtain $e^{\lambda_1 - \frac{\lambda_2^2}{4\lambda_3} - 1} = \frac{\sqrt{-\lambda_3}}{\sqrt{\pi}}$

From (ii) we get $\lambda_2 = 0$

From (iii) and (i) we have: $\sqrt{\pi}\,(-2\lambda_3)\cdot\frac{\sqrt{-\lambda_3}}{\sqrt{\pi}} = \sigma^2 4\,(-\lambda_3)^{\frac{3}{2}}$

$$-2\lambda_3\sqrt{-\lambda_3} = \sigma^2 4\cdot(-\lambda_3)^{\frac{3}{2}}$$

$$2\,(-\lambda_3)^{\frac{3}{2}} = \sigma^2 4\cdot(-\lambda_3)^{\frac{3}{2}}$$

$$\lambda_3 = -\frac{1}{2\sigma^2}$$

Insert into ①:

$$e^{\lambda_1 - 1} = \frac{1}{\sqrt{2\pi}\,\sigma}$$

$$\lambda_1 - 1 = -\log(\sqrt{2\pi}\,\sigma)$$

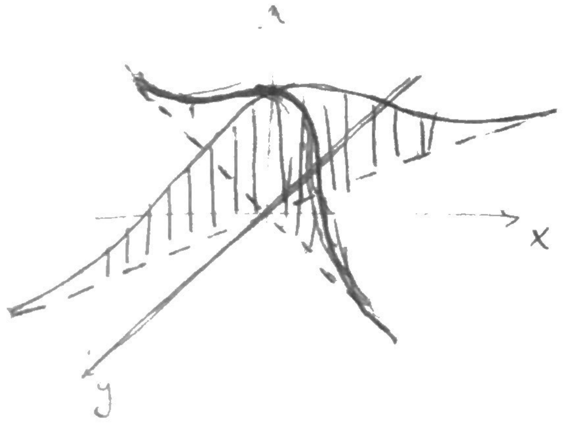
$$\lambda_1 = 1 - \log(\sqrt{2\pi}\,\sigma)$$

We obtain:

$$P(x) = e^{-1+1-\log(\sqrt{2\pi}\,\sigma) - \frac{1}{2\sigma^2}x^2} = \frac{1}{\sqrt{2\pi}\,\sigma}\,e^{-\frac{1}{2\sigma^2}x^2}$$

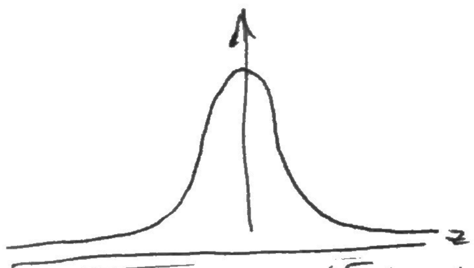
Ex 2

a) $P(x) = \mathcal{N}(0,1)$ $\quad p(y|x) = \frac{1}{2}\delta(y-x) + \frac{1}{2}\delta(y+x)$



$\underline{\theta = 0}$ $\quad z = \langle \binom{1}{0}, \binom{x}{y} \rangle = x$

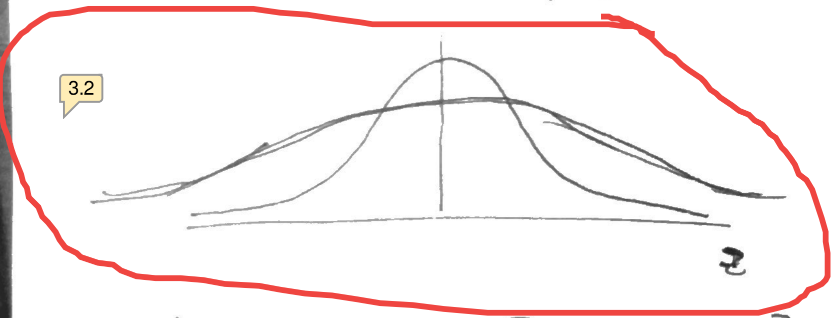
$\Rightarrow P(z) = P(x)$



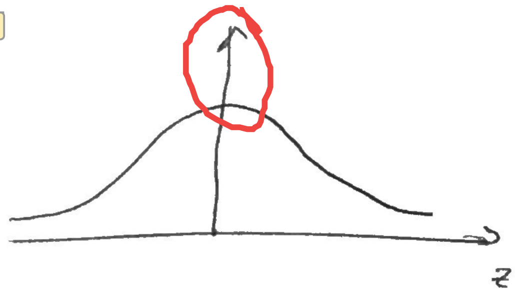
$\theta = \frac{\pi}{8}$. $\quad z = \langle \begin{pmatrix} \frac{\sqrt{2+\sqrt{2}}}{2} \\ \frac{\sqrt{2-\sqrt{2}}}{2} \end{pmatrix}, \begin{pmatrix} x \\ y \end{pmatrix} \rangle$

$\theta = \frac{\pi}{4}$, $\quad z = \langle \begin{pmatrix} \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \end{pmatrix}, \begin{pmatrix} x \\ y \end{pmatrix} \rangle$

3.2



3.1



b) $Var(z) = \mathbb{E}[(z - \mathbb{E}[z])^2]$

$$\mathbb{E}[z] = \int_{x,y} z \; P(x,y) \, dx\,dy =$$

$$= \int_{x,y} (x\cos\theta + y\sin\theta) \left[\frac{1}{2}\delta(y-x) + \frac{1}{2}\delta(y+x) \right] P(x) \, dy\,dx$$

$$= \int \left[\frac{1}{2}(x\cos\theta + x\sin\theta) + \frac{1}{2}(x\cos\theta + x\sin\theta) \right] P(x) \, dx$$

$$= \int_x x\cos\theta \, P(x) \, dx = \cos\theta \underbrace{\int x \, P(x) \, dx}_{E[x] = 0} = 0$$

$$Var[Z] = \mathbb{E}[Z^2] = \int Z^2 P(x,y)\,dx\,dy$$

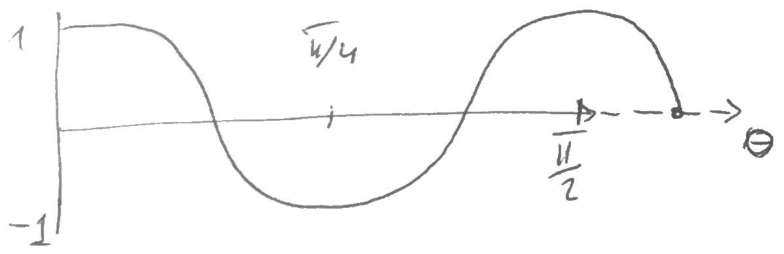
$$\int_x x^2[\cos^2\Theta + \sin^2\Theta]P(x)\,dx = \int x^2 P(x)\,dx = 1$$

c) $kurt[Z(\theta)] = \mathbb{E}[Z(\Theta)^4]$

$$\int x^4 [\underbrace{\cos^4\Theta + \sin^4\Theta}_{} + \underbrace{6\cos^2\Theta\sin^2\Theta}_{}]P(x)\,dx$$

$$\underset{=1}{(\cos^2\theta + \sin^2\theta)^2} + \underbrace{4\cos^2\theta\sin^2\theta}_{=\sin(2\theta)}$$

$$= \sin^2 2\theta \underbrace{\int x^4 P(x)\,dx}_{\mathbb{E}[X^4]} = \frac{1-\cos 4\theta}{2}\mathbb{E}[X]$$



$\theta \in \left\{\frac{\pi}{2}, \frac{3\pi}{4}, \frac{5\pi}{4}, \frac{7\pi}{4}\right\}$

sheet03

May 9, 2019

1 Independent Component Analysis

In this exercise, you will implement the FastICA algorithm, and apply it to model independent components of a distribution of image patches. The description of the fastICA method is given in the paper *"A. Hyvärinen and E. Oja. 2000. Independent component analysis: algorithms and applications"* linked from ISIS, and we frequently refer to sections and equations in that paper.

Three methods are provided for your convenience:

- `utils.load()` extracts a dataset of image patches from an collection of images (contained in the folder `images/` that can be extracted from the `images.zip` file). The method returns a list of RGB image patches of size 12×12, presented as a matrix of size $\#patches \times 432$. (Note that $12 \cdot 12 \cdot 3 = 432$).
- `utils.scatterplot(...)` produces a scatter plot from a two-dimensional data set. Each point in the scatter plot represents one image patch.
- `utils.render(...)` takes a matrix of size $\#patches \times 432$ as input and renders these patches in the IPython notebook.

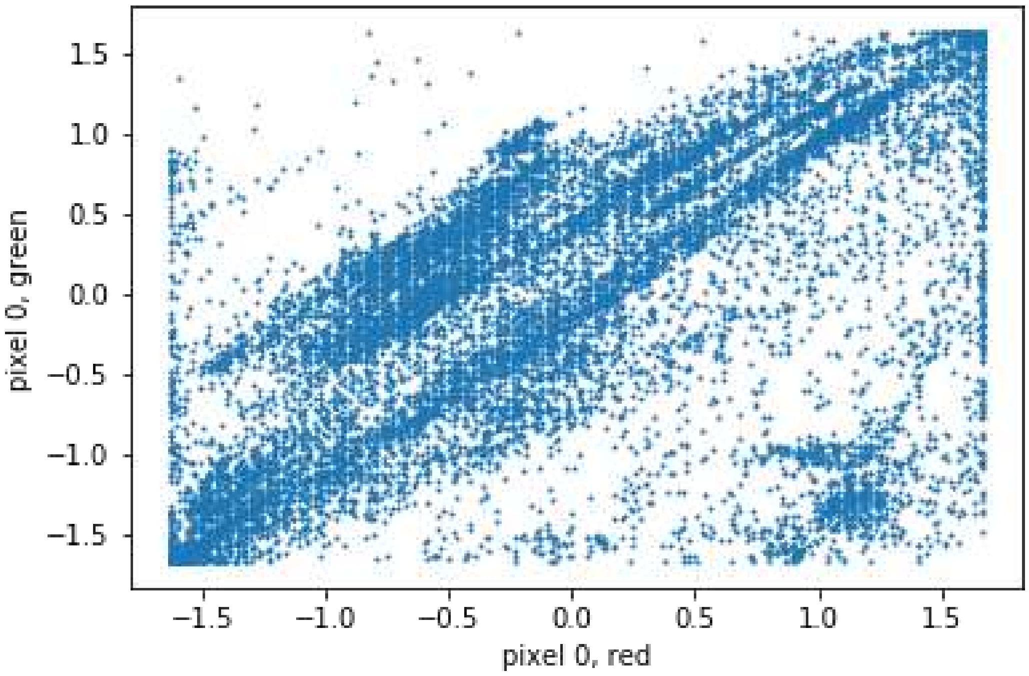
1.1 Demo code

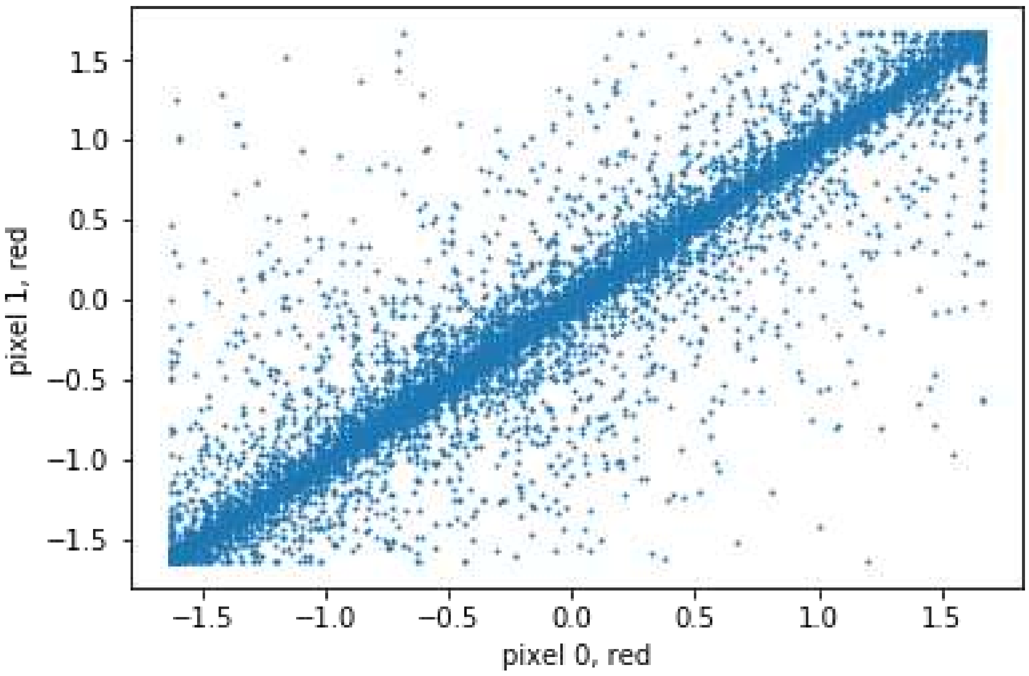
A demo code that makes use of these three methods is given below. The code performs basic analysis such as loading the data, plotting correlations between neighboring pixels, or different color channels of the same pixel, and rendering some image patches.

```
In [1]: import utils
        %matplotlib inline

        # Load the dataset of image patches
        X = utils.load()
        print(X.shape)

        # Plot the red vs. green channel of the first pixel
        utils.scatterplot(X[:,0],X[:,1],xlabel='pixel 0, red',ylabel='pixel 0, green')

        # Plot the red channel of the first and second pixel
        utils.scatterplot(X[:,0],X[:,3],xlabel='pixel 0, red',ylabel='pixel 1, red')
```

```
    # Visualize 500 image patches from the image
    utils.render(X[:500])
```

```
(14840, 432)
```





1.2 Whitening (10 P)

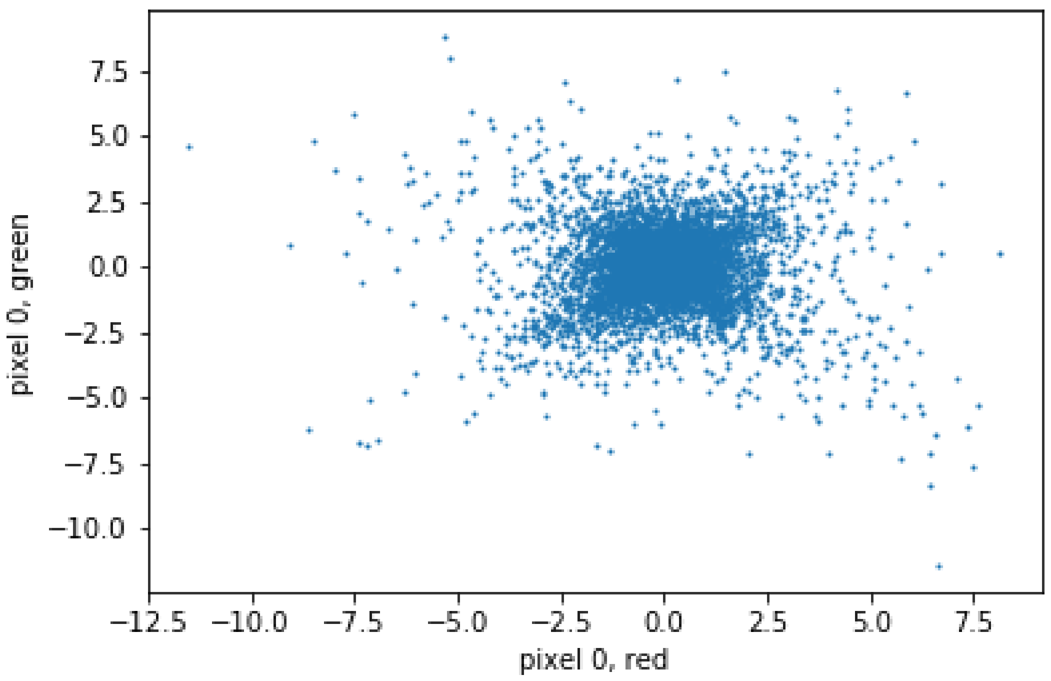
Independent component analysis applies whitening to the data as a preprocessing step. The whitened data matrix $\tilde{X}$ is obtained by linear projection of X, such data such that $\mathrm{E}[\tilde{x}\tilde{x}^\top] = I$, where $\tilde{x}$ is a row of the whitened matrix $\tilde{X}$. See Section 5.2 of the paper for a complete description of the whitening procedure.

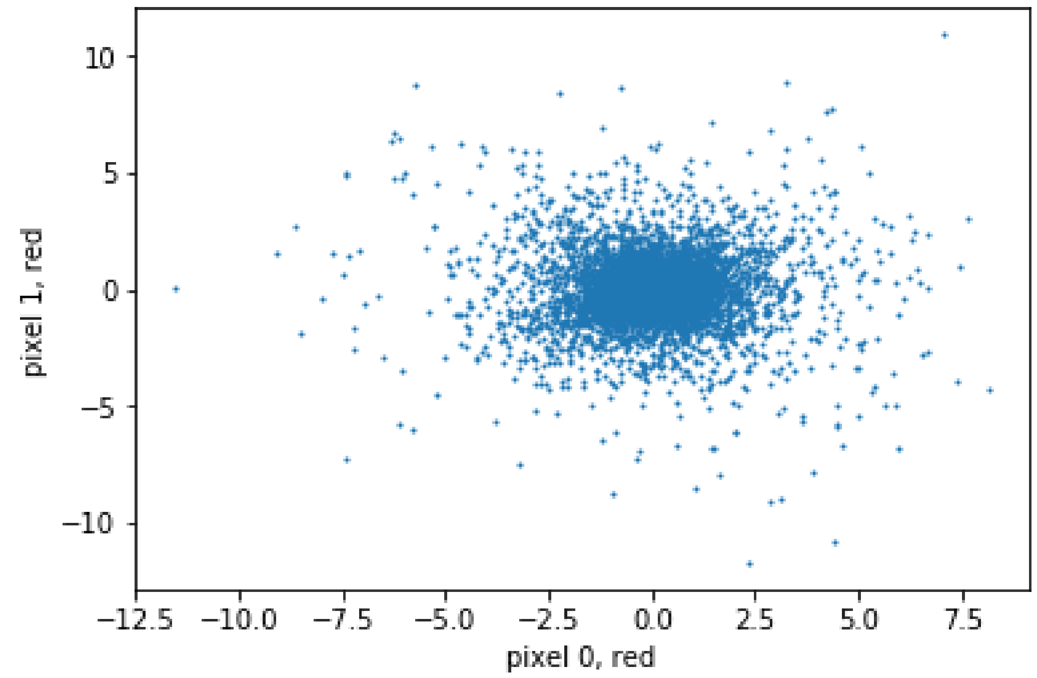
Tasks:

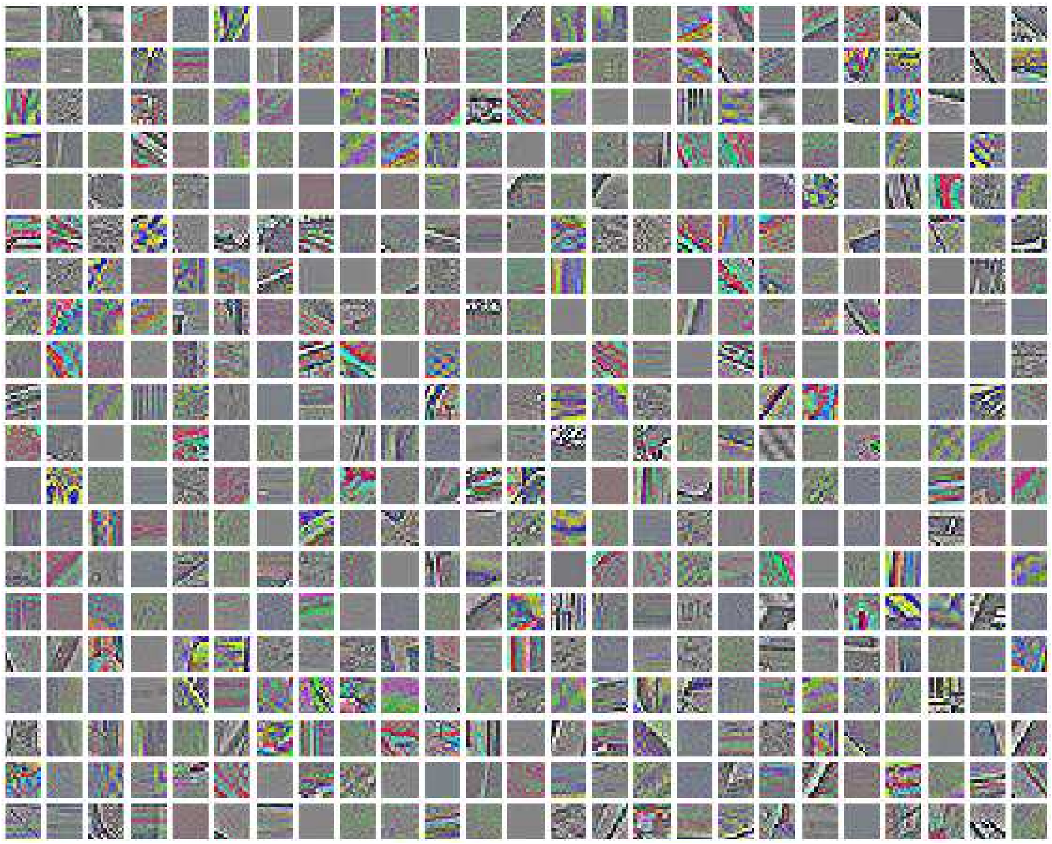
- **Implement a function that returns a whitened version of the data given as input.**
- **Add to this function a test that makes sure that $\mathrm{E}[\tilde{x}\tilde{x}^\top] \approx I$ (up to numerical accuracy).**
- **Reproduce the scatter plots of the demo code, but this time, using the whitened data.**
- **Render 500 whitened image patches.**

```
In [5]: import numpy as np
        from numpy import linalg as lin
        import utils
        %matplotlib inline
```

```
def whiten(X):

    #first center
    Xc = X - X.mean(axis = 0)
    #compute covariance
    cov = np.cov(Xc.T)
    #apply eigendecomposition                      8.1
    E,V = lin.eigh(cov)
    D = np.diag(1. / np.sqrt(E+10**(-6)))
    # whitening matrix
    W = np.dot(np.dot(V, D), V.T)
    Xw = np.dot(W, Xc.T).T

    testX = np.dot(Xw.T, Xw)
    delta = np.abs(np.eye(Xw.shape[1])-np.dot(Xw.T,Xw)/Xw.shape[0])
    assert np.all(delta<0.1)

    return Xw

Xw = whiten(X)

utils.scatterplot(Xw[:,0],Xw[:,1],xlabel='pixel 0, red',ylabel='pixel 0, green')

# Plot the red channel of the first and second pixel
utils.scatterplot(Xw[:,0],Xw[:,3],xlabel='pixel 0, red',ylabel='pixel 1, red')

# Visualize 500 image patches from the image
utils.render(Xw[:500])
```


pixel 0, green
pixel 0, red
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−12.5
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5


pixel 1, red
pixel 0, red
10
5
0
−5
−10
−12.5
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5



1.3 Implementing FastICA (20 P)

We now would like to learn 100 independent components of the distribution of whitened image patches. For this, we follow the procedure described in the Chapter 6 of the paper. Implementation details specific to this exercise are given below:

- **Nonquadratic function G**: In this exercise, we will make use of the nonquadratic function $G(x) = \frac{1}{a} \log \cosh(ax)$, proposed in Section 4.3.2 of the paper, with $a = 1.5$. This function admits as a derivative the function $g(x) = \tanh(ax)$, and as a double derivative the function $g'(x) = a \cdot (1 - \tanh^2(ax))$.
- **Number of iterations**: The FastICA procedure will be run for 64 iterations. Note that the training procedure can take a relatively long time (up to 5 minutes depending on the system). Therefore, during the developement phase, it is advised to run the algorithm for a fraction of the total number of iterations.
- **Objective function**: The objective function that is maximized by the ICA training algorithm is given in Equation 25 of the paper. Note that since we learn 100 independent components, the objective function is in fact the *sum* of the objective functions of each independent components.

- **Finding multiple independent components**: Conceptually, finding multiple independent components as described in the paper is equivalent to running multiple instances of FastICA (one per independent component), under the constraint that the components learned by these instances are decorrelated. In order to keep the learning procedure computationally affordable, the code must be parallelized, in particular, make use of numpy matrix multiplications instead of loops whenever it is possible.

- **Weight decorrelation**: To decorrelate outputs, we use the inverse square root method given in Equation 45.

Tasks:

- **Implement the FastICA method described in the paper, and run it for 64 iterations.**

- **Print the value of the objective function at each iteration.**

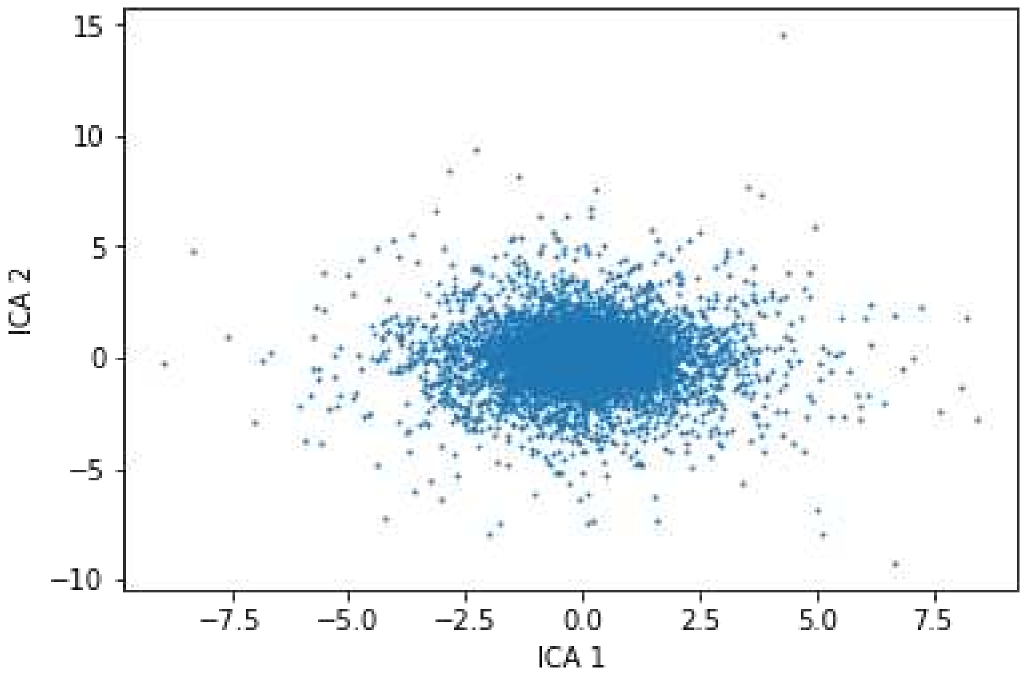
- **Create a scatter plot of the projection of the whitened data on two distinct independent components after 0, 1, 3, 7, 15, 31, 63 iterations.**

- **Visualize the learned independent components using the function `render(...)`.**

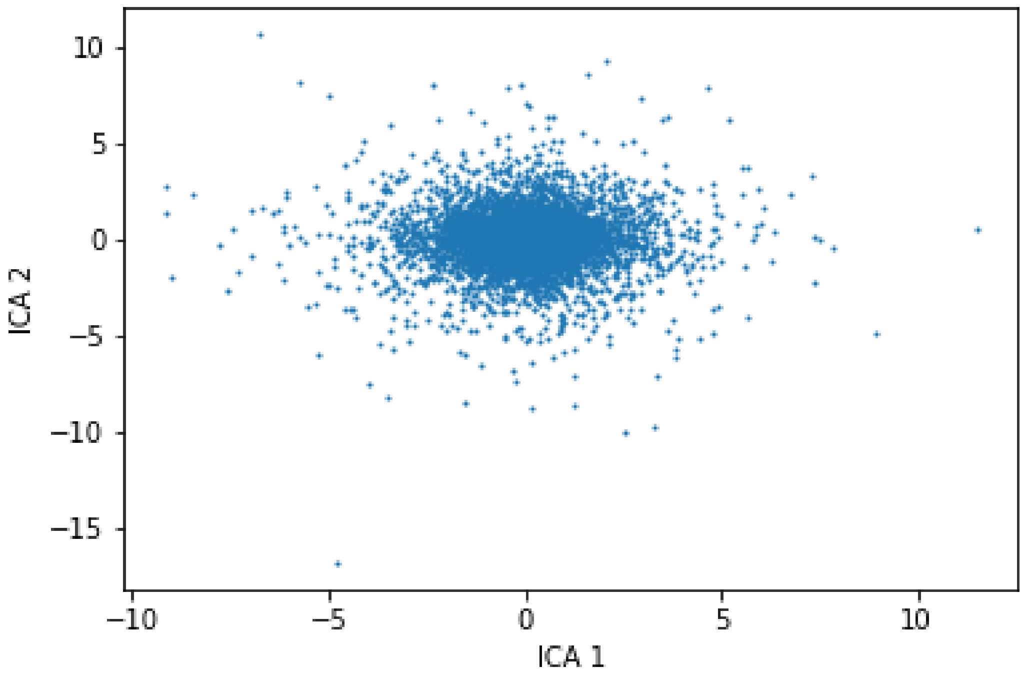
In [7]:
```
import random as rnd

iterations = 64
components = 100

def G(x):
    G = (1/1.5)*np.log(np.cosh(1.5*x))
    return G #,Gdx,Gd2x

def g(x):
    g = np.tanh(x*1.5)
    return g

def gdx(x):
    Gdx = (1-(np.tanh(1.5*x))**2)*1.5
    return Gdx

def obj(y,Gv):
    EGy = np.mean(G(y),axis=1)
    J = (EGy-Gv)**2
    return np.sum(J)

#initialize random gaussian  variable
def EGv():
    sumG=0
    for i in range(99999):
        sumG += G(rnd.normalvariate(0,1))
    return sumG/99999
```

11.1

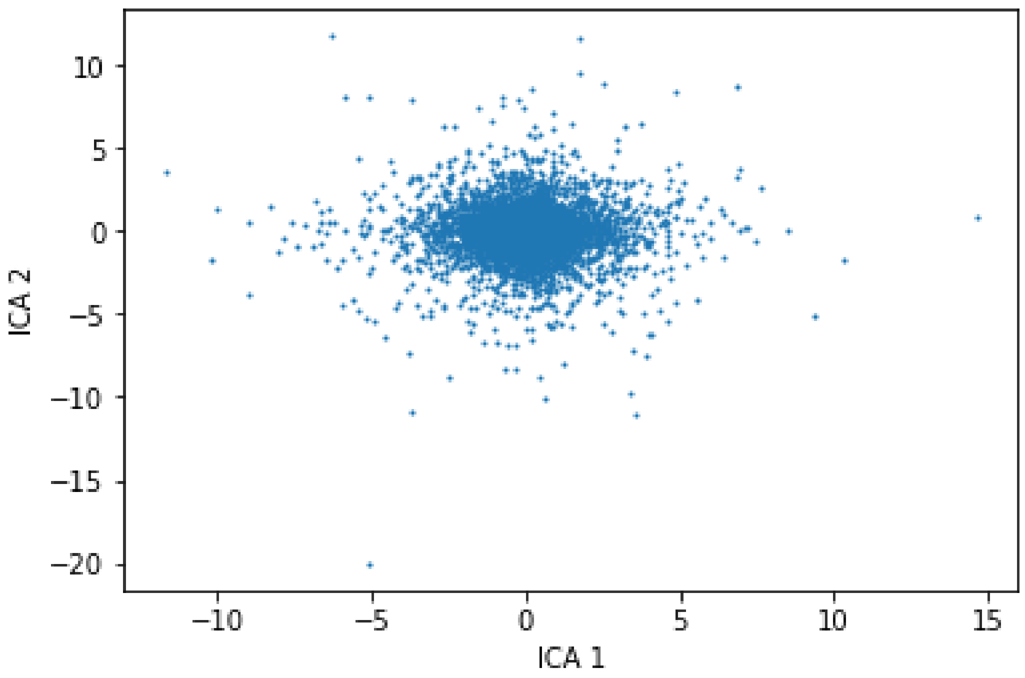
```python
def decorrelate(W):
    e,v = np.linalg.eigh(np.dot(W,W.T))
    D = np.diag(1.0/np.sqrt(e))
    return np.dot(np.dot(np.dot(v,D),v.T),W)


# Get a value for G(v) of eq.25 (objective function)
eGv = EGv()


# Initiate the ICA components randomly
W = np.random.normal(size=(components, X.shape[1]))
W = decorrelate(W)

#List of iterations in which a scatter plot should be done
it_plt = [0, 1, 3, 7, 15, 31, 63]

for i in range(iterations):
    y = np.dot(W,Xw.T)
    gy = g(y)
    gdxy = gdx(y)
    W = np.dot(gy,Xw)/Xw.shape[0] - np.dot(np.diag(np.mean(gdxy,axis=1)),W)
    W = decorrelate(W)
    if i in it_plt:
        utils.scatterplot(np.dot(W[0,:],Xw.T),np.dot(W[1,:],Xw.T),
                          xlabel='ICA 1',ylabel='ICA 2')
    print('it = %2d, J(W) = %.3f'%(i, obj(y,eGv)))

utils.render(W)
```



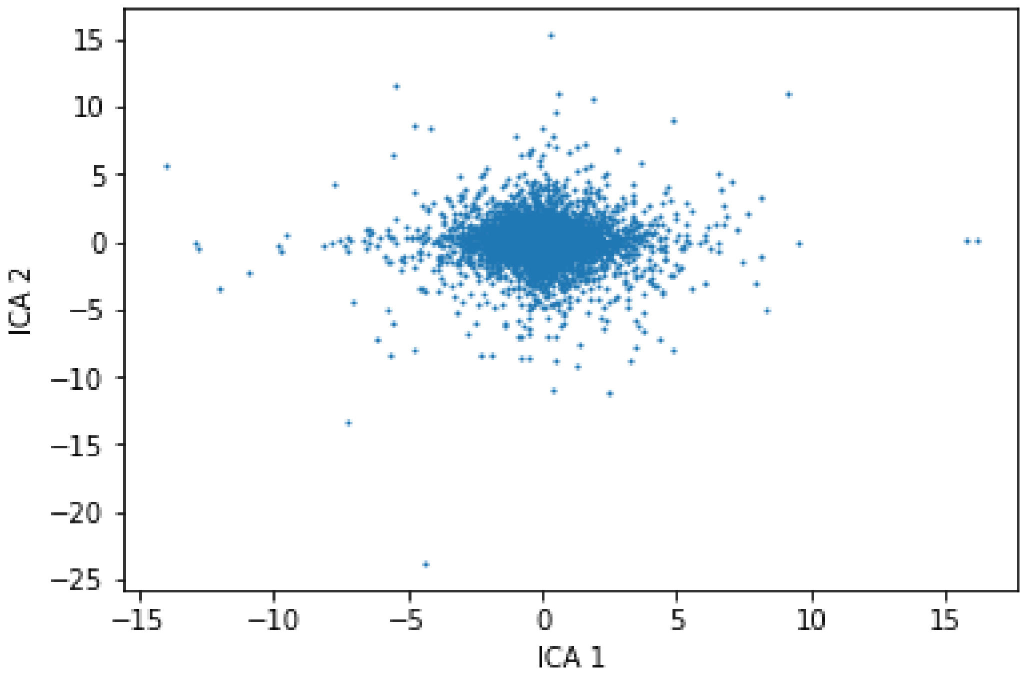
```
it =  0, J(W) = 0.920
```



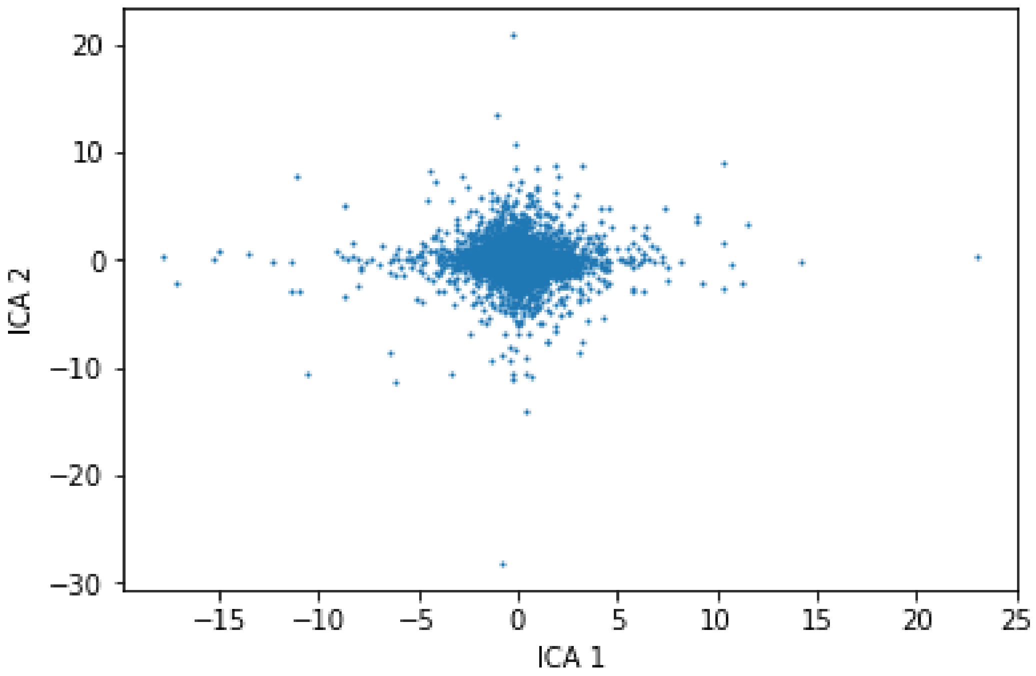
```
it =   1, J(W) = 1.405
it =   2, J(W) = 1.819
```



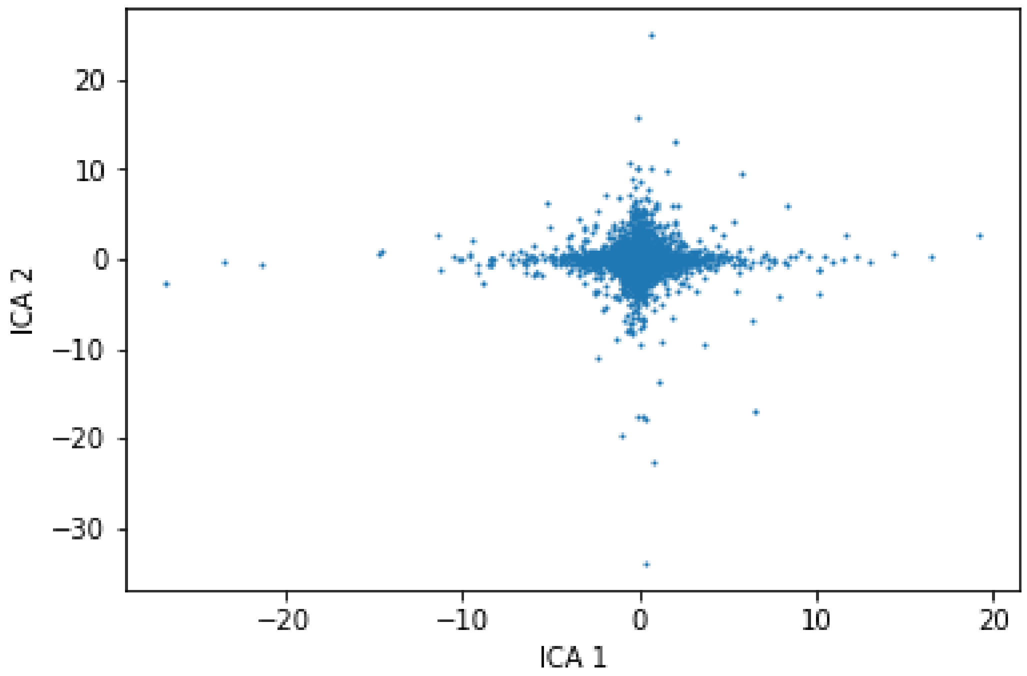
```
it =   3, J(W) = 2.158
it =   4, J(W) = 2.435
it =   5, J(W) = 2.664
it =   6, J(W) = 2.855
```



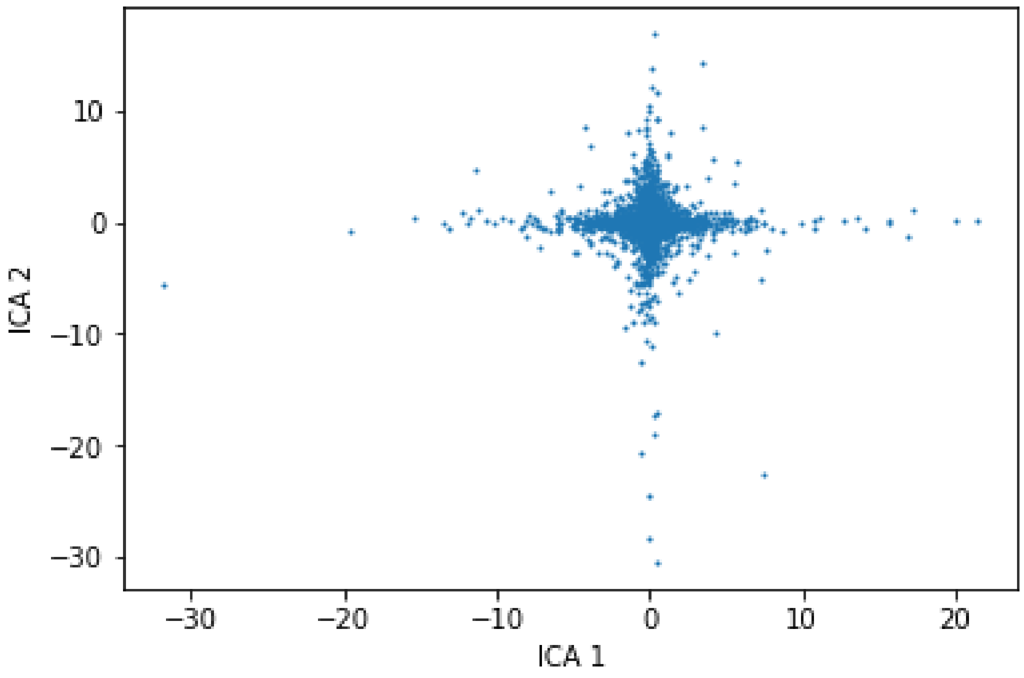
```
it =  7, J(W) = 3.018
it =  8, J(W) = 3.162
it =  9, J(W) = 3.292
it = 10, J(W) = 3.415
it = 11, J(W) = 3.534
it = 12, J(W) = 3.653
it = 13, J(W) = 3.773
it = 14, J(W) = 3.895
```



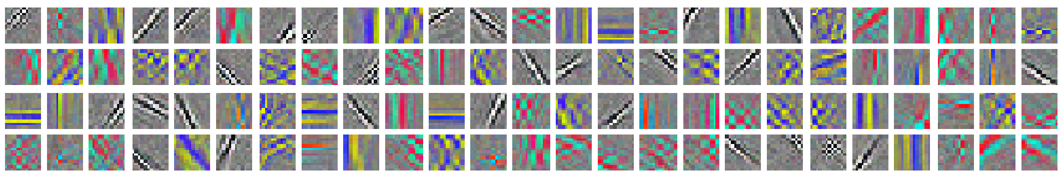
```
it = 15, J(W) = 4.019
it = 16, J(W) = 4.142
it = 17, J(W) = 4.264
it = 18, J(W) = 4.381
it = 19, J(W) = 4.494
it = 20, J(W) = 4.600
it = 21, J(W) = 4.700
it = 22, J(W) = 4.794
it = 23, J(W) = 4.884
it = 24, J(W) = 4.969
it = 25, J(W) = 5.052
it = 26, J(W) = 5.134
it = 27, J(W) = 5.214
it = 28, J(W) = 5.294
it = 29, J(W) = 5.373
it = 30, J(W) = 5.452
```



```
it = 31, J(W) = 5.529
it = 32, J(W) = 5.605
it = 33, J(W) = 5.679
it = 34, J(W) = 5.749
it = 35, J(W) = 5.814
it = 36, J(W) = 5.875
it = 37, J(W) = 5.931
it = 38, J(W) = 5.985
it = 39, J(W) = 6.036
it = 40, J(W) = 6.084
it = 41, J(W) = 6.130
it = 42, J(W) = 6.174
it = 43, J(W) = 6.215
it = 44, J(W) = 6.254
it = 45, J(W) = 6.291
it = 46, J(W) = 6.326
it = 47, J(W) = 6.358
it = 48, J(W) = 6.388
it = 49, J(W) = 6.415
it = 50, J(W) = 6.440
it = 51, J(W) = 6.463
it = 52, J(W) = 6.484
it = 53, J(W) = 6.504
it = 54, J(W) = 6.522
it = 55, J(W) = 6.539
it = 56, J(W) = 6.555
it = 57, J(W) = 6.569
it = 58, J(W) = 6.582
it = 59, J(W) = 6.594
it = 60, J(W) = 6.605
it = 61, J(W) = 6.615
it = 62, J(W) = 6.624
```



```
it = 63, J(W) = 6.632
```



Index of comments

1.1	Why can you make this assumption?
1.2	What did you use to obtain these integrals?
3.1	No ticks on the axes makes it impossible to compare projections.
3.2	What are these two curves?
4.1	The task was "find the principal components". What is your solution? Uncommented calculations are not sufficient.
4.2	Again, it is not evident what your solution to the problem is or even what problem you are solving. Uncommented calculations and a sketch are not sufficient.
8.1	Not necessary here. Also, note that you take the sqrt of 10**(-6), meaning you actually add 10**(-3) in the denominator which is quite large for such an eps and this is why your assertion only holds for relatively large bounds on delta
11.1	Rather inefficient and unneccessarily verbose. Use np.random.normal(0,1,n_samples) instead