HW4

310553040 原瑄

- Part I: Explain each step of your implementation in detail for three function mentioned above

Center: move the data points close to zero (data values minus mean of the data)

```
def center(x):
    # code here: (~ two lines)
    # centering by subtraction of the mean from our input X
    mean = np.mean(x, axis=1, keepdims=True)
    centered =  x - mean
    return centered, mean
```

Whiten: The goal here is to linearly transform the observed signals X in a way that potential correlations between the signals are removed and their variances equal unity.

Suppose X is a random (column) vector with non-singular covariance matrix Σ and mean 0. Then the transformation $X_w = W_{white} X$ with a **whitening matrix** W_{white} satisfying the condition $W_{white}^T W_{white} = \Sigma^{-1}$ yields the whitened random vector X_w with unit diagonal covariance. There are infinitely many possible whitening matrices $whiteM$ that all satisfy the above condition. In here, we use $W_{white} = UD^{-1/2}U^T$, where UDV^T is the singular value decomposition of Σ.

According to last paragraph, we know that UDV^T is the singular value decomposition of Σ. First we calculate the covariance Σ. Second, singular value decomposition. Third, calculate $D^{\frac{-1}{2}}$. Forth, calculate W_{white}. Finally, we get $X_w = W_{white}X$.

```
def whiten(x):
    # code here: (~ 5 lines)
    # Calculate the covariance matrix
    coVarM = covariance(X)

    # Single value decoposition. Hint: np.linalg.svd
    U, S, V = np.linalg.svd(coVarM)

    # Calculate diagonal matrix of eigenvalues
    d = np.diag(1.0 / np.sqrt(S))

    # Calculate whitening matrix
    W_white = np.dot(U, np.dot(d, U.T))

    # Project onto whitening matrix
    X_w = np.dot(W_white, X)

    return X_w, W_white
```

FastICA: FastICA seeks an orthogonal rotation of pre-whitened data, through a fixed-point iteration scheme, that maximizes a measure of non-Gaussianity of the rotated components.

$$f(u) = \log\cosh(u)$$

$$g(u) = \tanh(u)$$

$$g'(u) = 1 - \tanh^2(u)$$

$$for\ 1\ to\ number\ of\ components\ c:$$

$$w_p \equiv random\ initialisation$$

$$while\ w_p\ not\ < threshold:$$

$$\boxed{w_p \equiv \frac{1}{n}(Xg(\underline{W^T X}) - g'(\underline{W^T X})W)}$$

$$\boxed{w_p \equiv w_p - \sum_{j=1}^{p-1}(w_p^T w_j)w_j}$$

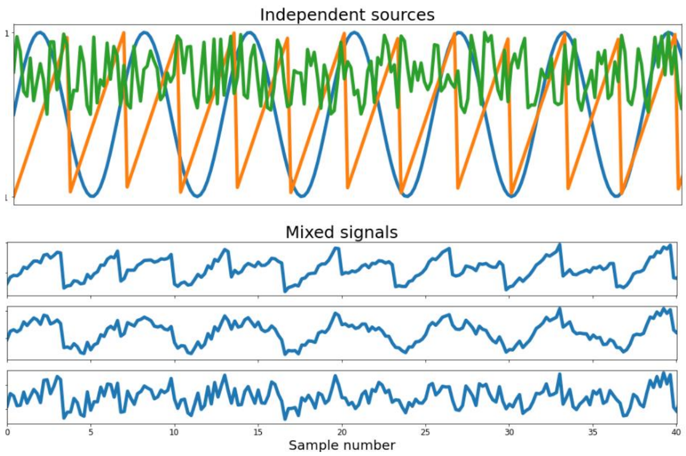
$$\boxed{w_p \equiv w_p / ||w_p||}$$

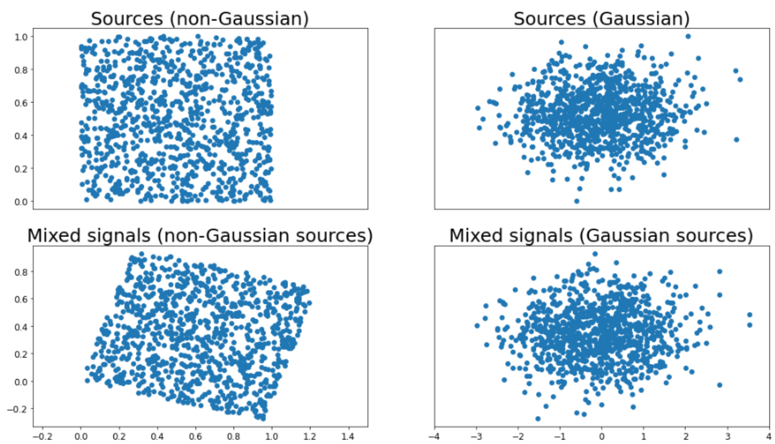
$$W \equiv [w_1, \ldots, w_c]$$ ¶

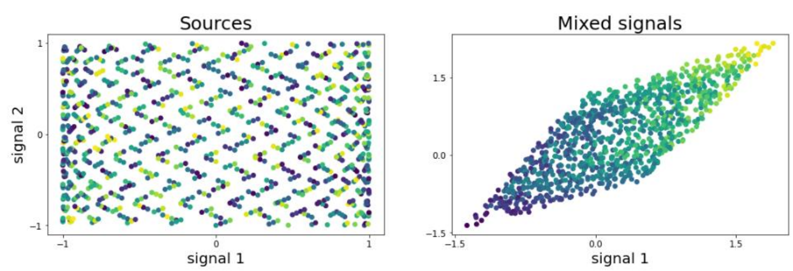
The formula above the same color corresponds to the code below the same color

```
def fastIca(signals,  alpha = 1, thresh=1e-8, iterations=5000):
    m, n = signals.shape

    # Initialize random weights
    W = np.random.rand(m, m)

    # code here:
    for c in range(m):
            w = W[c, :].copy().reshape(m, 1)
            w = w / np.sqrt((w ** 2).sum())

            i = 0
            lim = 100
            while ((lim > thresh) & (i < iterations)):

                # Dot product of weight and signal
                ws = w.T@signals

                # Pass w*s into contrast function g
                wg = np.tanh(ws*alpha).T

                # Pass w*s into g prime
                wg_ = (1-np.square(np.tanh(ws)))*alpha

                # Update weights
                wNew = (signals*wg.T).mean(axis=1) -\
                        wg_.mean()*w.squeeze()

                # Decorrelate weights
                wNew = wNew - np.dot(np.dot(wNew, W[:c].T), W[:c])
                wNew = wNew / np.sqrt(sum(wNew**2))

                # Calculate limit condition
                lim = np.abs(np.abs((wNew * w).sum()) - 1)

                # Update weights
                w = wNew

                # Update counter
                i += 1

            W[c, :] = w.T
    return W
```

- Part II: Screenshot of the all output results in ICA.ipynb. Below is an example

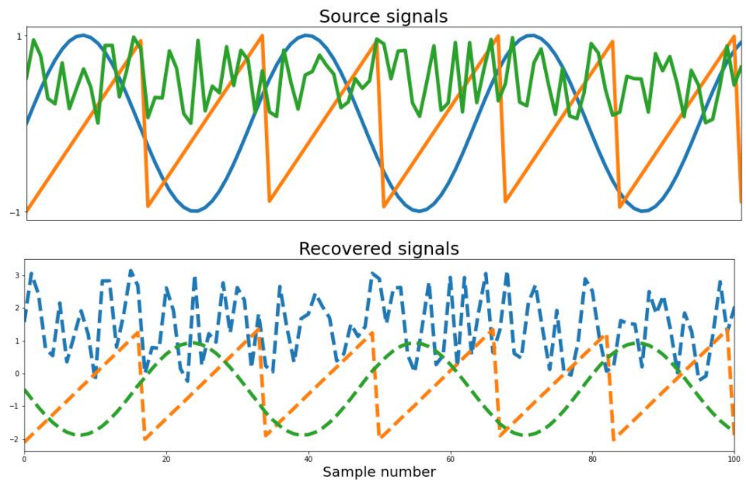


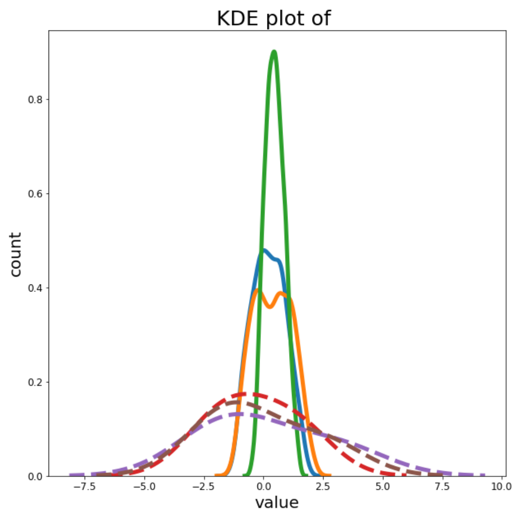





```
# Check if covariance of whitened matrix equals identity matrix
print(np.round(covariance(Xw)))
```

```
[[ 1. -0.  0.]
 [-0.  1.  0.]
 [ 0.  0.  1.]]
```


Source signals
1
−1
Recovered signals
3
2
1
0
−1
−2
0
20
40
60
80
100
Sample number


KDE plot of
count
0.8
0.6
0.4
0.2
0.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
value