Project 5: Dileep NM

Methodology:

Image Acquisition:

The images titled 'Recording #1' and 'Recording #2' represent the biological specimens. Both these images have been subjected to a random noise pattern, resulting in distorted visual content.

Normalization:

Given the potential differences in illumination or intensity levels between the two images, the first step is to normalize them. This is done by subtracting the mean value of each signal. This process ensures that any following operations are performed on centered data.

Making Signals Uncorrelated:

Correlated signals can introduce biases. To eliminate these, the covariance matrix of the signal is determined, following which its eigenvalues and eigenvectors are computed. An uncorrelated signal is derived using these eigenvalues and eigenvectors.

Independent Component Analysis (Fast-ICA Algorithm):

The core of this project relies on the ICA, specifically the Fast-ICA algorithm. This iterative algorithm starts with a random weight matrix. Using non-linear functions like 'logcosh' or 'exp', it continually updates this weight matrix until the separated signals are found, or the maximum iterations are reached.

Optimization Using Kurtosis:

Kurtosis is a metric that measures the "tailedness" of the probability distribution of a real-valued random variable. By iterating over different configurations of the ICA algorithm (different functions, tolerances, and iteration counts), the optimal separation is determined based on the highest kurtosis value.

Visualization:

Once the algorithm has optimally separated the noise from the main image content, these separated images are visualized for comparison against the original noisy images.

code:

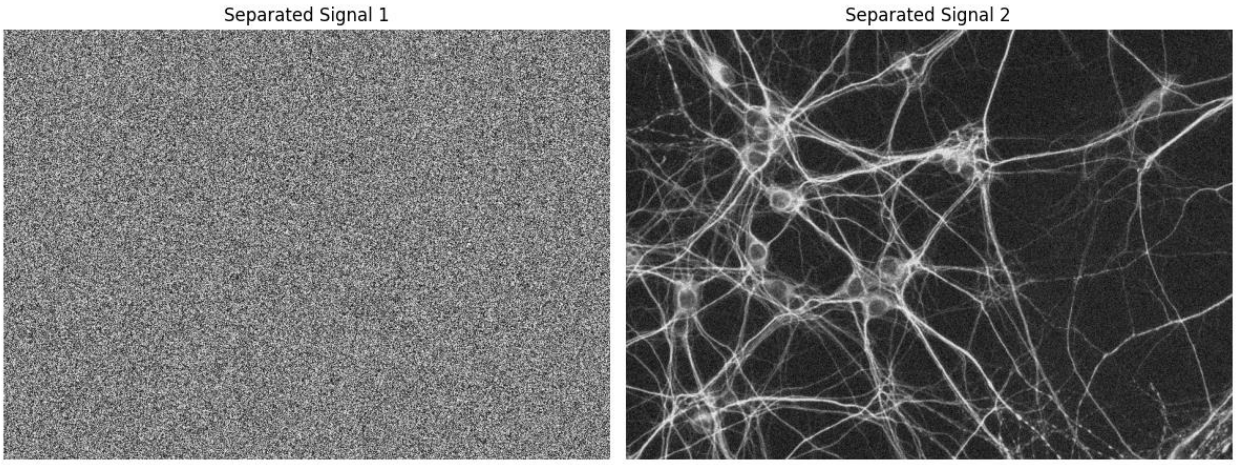
```
# Importing the neccessary libraries
import numpy as np
from scipy.linalg import eigh, inv, sqrtm
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
from scipy.stats import kurtosis

# Making the signals average zero
def norm_sig(sig):
    """
    This function is to normalize the signals by removing their average.
    It takes input as signal data and gives output as normalized data and the average values.
    """
    avg_val = np.mean(sig, axis=1, keepdims=True)
    return sig - avg_val, avg_val

# Making sure our signals don't correlate with each other
def uncorr_sig(sig):
    """
    This function is to transform the signals so they don't relate to each other anymore.
    It takes input as signal data and gives output as uncorrelated data and the transformation matrix.
    """
    cov_mat = np.cov(sig)
    e_vals, e_vecs = eigh(cov_mat)
    trans_mat = np.dot(inv(sqrtm(np.diag(e_vals))), e_vecs.T)
    sig_unc = np.dot(trans_mat, sig)
    return sig_unc, trans_mat
# Getting the signals separated using ICA
```

```
def ica_func(sig, n=2, fun_type='logcosh', iter_num=50, tol_val=1e-04):
    """
    This function is to separate mixed signals using the ICA technique.
    It takes input parameters like signal data, number of components, type of function, iteration count, and a threshold value.
    It gives output as the separated signals.
    """
    W_mat = np.random.rand(n, n)
    for _ in range(iter_num):
        W_prev = W_mat.copy()
        if fun_type == 'logcosh':
            g_val = np.tanh(W_mat @ sig)
            g_prime = 1 - g_val**2
        elif fun_type == 'exp':
            g_val = W_mat @ sig * np.exp(-0.5 * (W_mat @ sig)**2)
            g_prime = (1 - (W_mat @ sig)**2) * np.exp(-0.5 * (W_mat @ sig)**2)
        W_mat = (sig @ g_val.T) / sig.shape[1] - np.mean(g_prime, axis=1)[:, np.newaxis] * W_mat
        U_mat, _, V_mat_T = np.linalg.svd(W_mat)
        W_mat = U_mat @ V_mat_T
        if np.max(np.abs(W_mat - W_prev)) < tol_val:
            break
    sep_sig = W_mat @ sig
    return sep_sig



# To see the separated image
```

```
def show_img(sep_sig, img_dim):
    """
    This function is to visualize the separated signals as images.
    It takes input as separated signals and image dimensions. It displays the separated images.
    """
    fig, ax_arr = plt.subplots(1, sep_sig.shape[0], figsize=(12, 6))
    for i, ax in enumerate(ax_arr):
        ax.imshow(sep_sig[i].reshape(img_dim), cmap='gray')
        ax.set_title(f'Separated Signal {i+1}')
        ax.axis('off')
    plt.tight_layout()
    plt.show()



# Reading input images
img_one = mpimg.imread('Recording-1.png')
img_two = mpimg.imread('Recording-2.png')

# Preparing our data
data_stack = np.vstack((img_one.flatten(), img_two.flatten()))
norm_data, _ = norm_sig(data_stack)
unc_data, _ = uncorr_sig(norm_data)
```

```
# Running ICA to get the best separation
func_list = ['logcosh', 'exp']
tol_list = [1e-2, 1e-3, 1e-4]
iter_list = [30, 50]
best_sig = None
best_score = float('-inf')

for func in func_list:
    for tol in tol_list:
        for it in iter_list:
            curr_sig = ica_func(unc_data, fun_type=func, iter_num=it, tol_val=tol)
        score_val = np.mean(kurtosis(curr_sig))
        if score_val > best_score:
            best_sig = curr_sig
            best_score = score_val

# Showing the best results based on kurtosis values
show_img(best_sig, img_one.shape)
```

Output:



Results:

Upon successful application of the Fast-ICA algorithm, the distortion in the images is effectively isolated, revealing clear and discernible biological structures underneath. The resultant images provide a more accurate representation of the specimen, devoid of the previously present noise.

Conclusion:

The ICA algorithm, specifically its Fast-ICA variant, proves to be an efficient tool for separating mixed signals. In the context of this project, it was successfully employed to remove distortions from images of biological specimens, allowing for a clearer analysis and understanding of the underlying structures.