Here I will show how well DCT performs on temporal decorrelation using one segment of the signal. I also illustrate the effects of removing the lowest highest frequency coefficients.

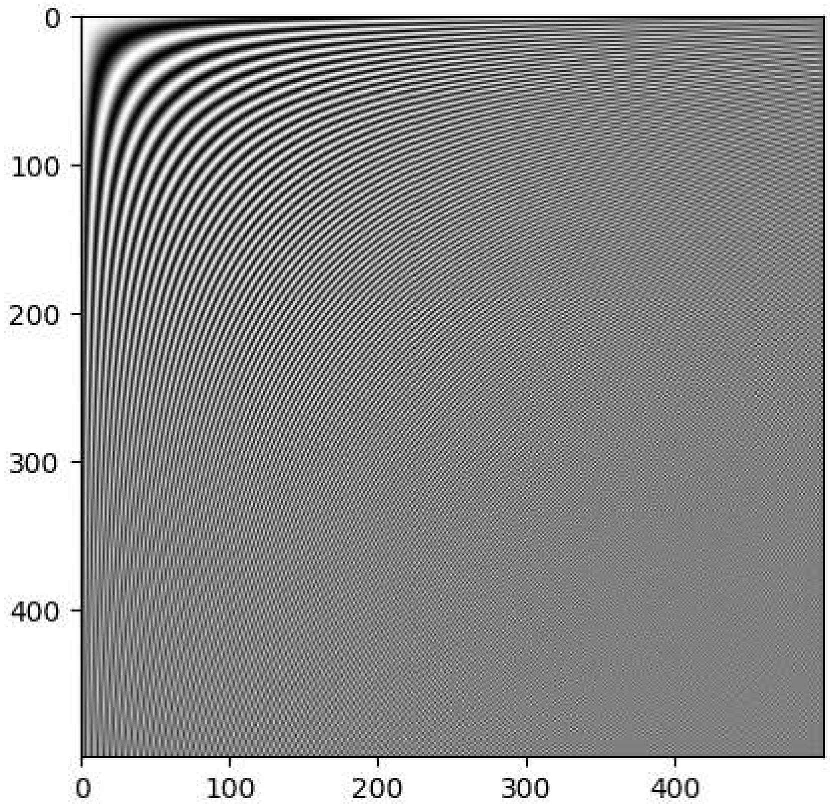
In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.fftpack import idct

s = np.load('../test_data_sintef.npy')
```

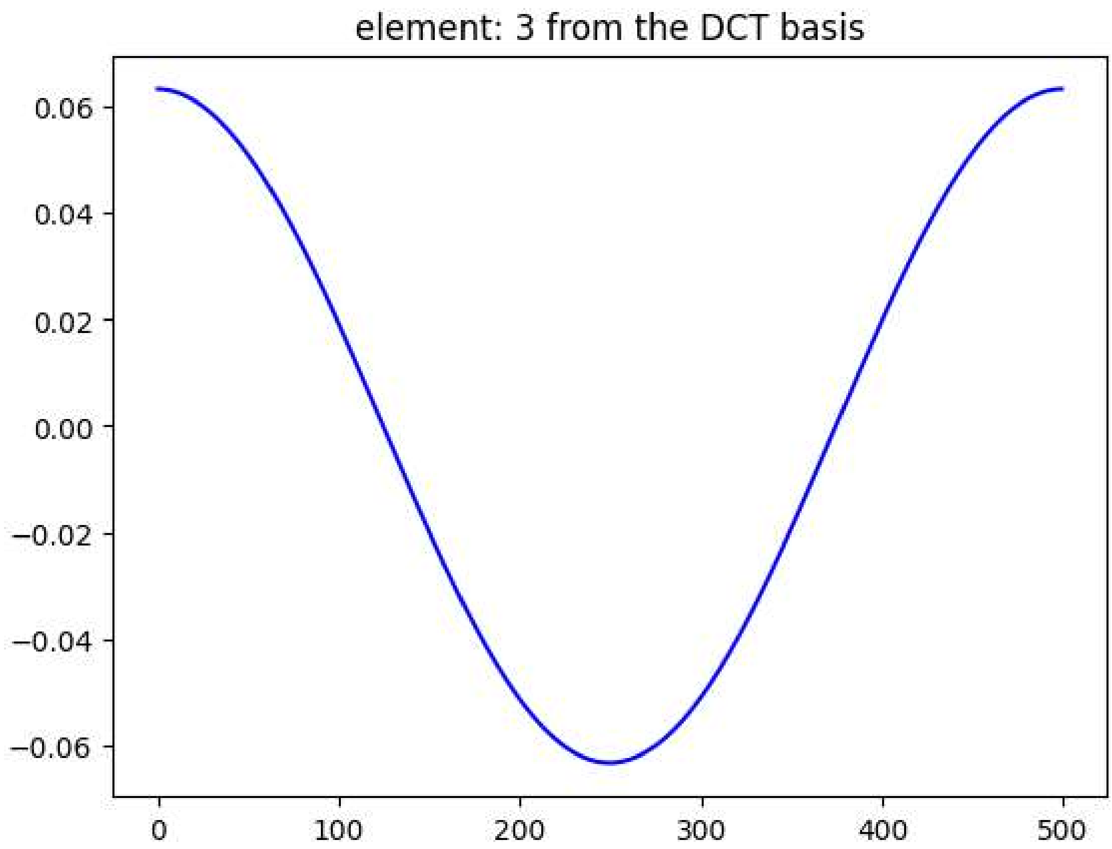
DCT temporal coding

setup

In [ ]:
```
# generate DCT basis
pp = 500

D = np.zeros((pp, pp))

# generate DCT basis
for k in range(pp):
    a = np.zeros(pp)
    a[k] = 1
    D[:, k] = idct(a, norm='ortho') # use idct on the atom

# plot DCT matrix
plt.figure()
plt.imshow(D, cmap='gray')

# plot one element from the DCT matrix
k = 2
plt.figure()
plt.plot(D[:, k], 'b')
plt.title(f'element: {k+1} from the DCT basis')
```

Out[ ]:
```
Text(0.5, 1.0, 'element: 3 from the DCT basis')
```





In [ ]:

```
def getDCT(temp):
    nSteps = int(s.shape[0]/pp)
    c_mat = np.zeros((pp, nSteps, s.shape[1]))

    for j in range(s.shape[1]):
        for i in range(nSteps):
            c_mat[:,i,j] = D.T@temp[pp*i:pp*i + pp, j]
```

```
    return c_mat

def getiDCT(temp):
    nSteps = int(s.shape[0]/pp)
    d_hat = np.zeros_like(s)

    for j in range(s.shape[1]):
        for i in range(nSteps):
            d_hat[pp*i:pp*i + pp, j] = D@temp[:,i,j]

    return d_hat
```

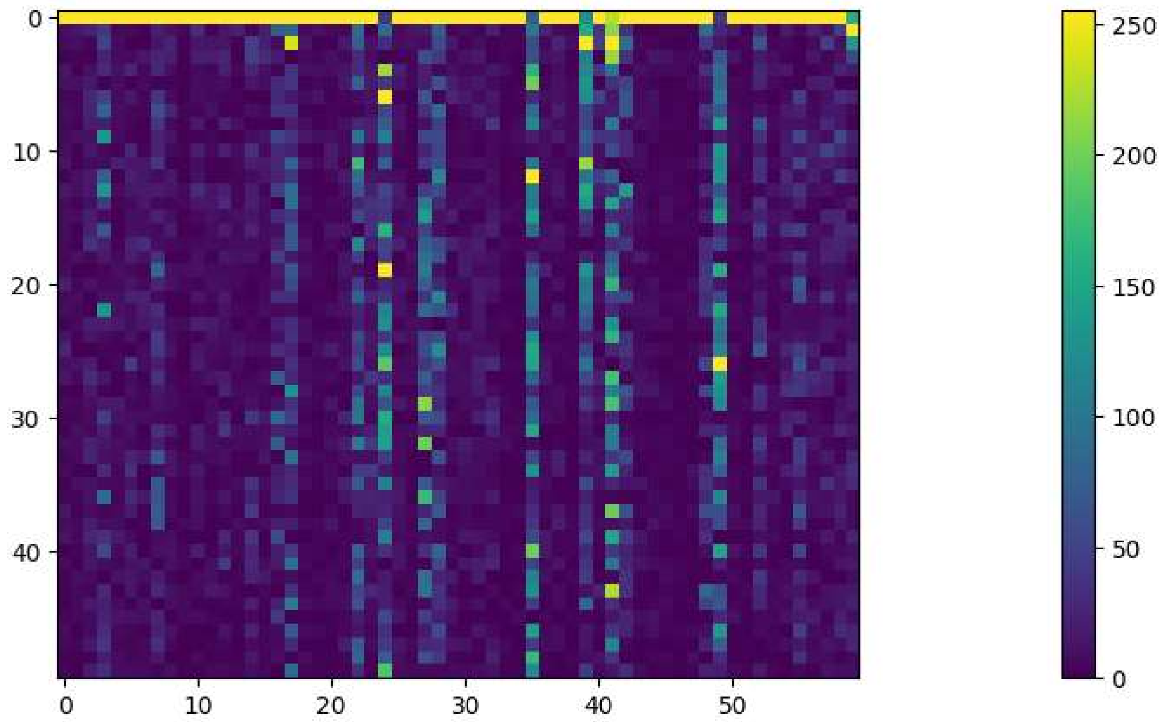
perform

In [ ]:

```
temp = s.astype(int)

# function to normalize each column of the dct coefficients
def visualize_dct(c):
    for i in range(c.shape[0]):
        c[i,:] = 255* (c[i,:] - c[i,:].min()) / (c[i,:].max() - c[i,:].min())
    return c

# perform DCT
c_mat = getDCT(temp).astype(int)

# flatten coefficient matrix
c = c_mat.reshape(c_mat.shape[0]*c_mat.shape[1], c_mat.shape[2]).T

# reduce the number of columns and rows for better visualization
M = 60
N = 50
c_new = np.zeros((N,M))
step_col = c_mat.shape[1] // M
step_row = c_mat.shape[0] // (N)
for j in range(N):
    for i in range(M):
        c_new[j,i] = np.mean(c_mat[j*step_row :j*step_row + step_row, i*step_col


# visulaize the DCT coefficients
plt.figure(figsize=[30,5])
plt.imshow(visualize_dct(abs(c_new).T).T)
plt.colorbar()
```

Out[ ]:

```
<matplotlib.colorbar.Colorbar at 0x1ea68782750>
```



this shows the coefficients of the first channel, we see that the DC offset has the biggest contribution.

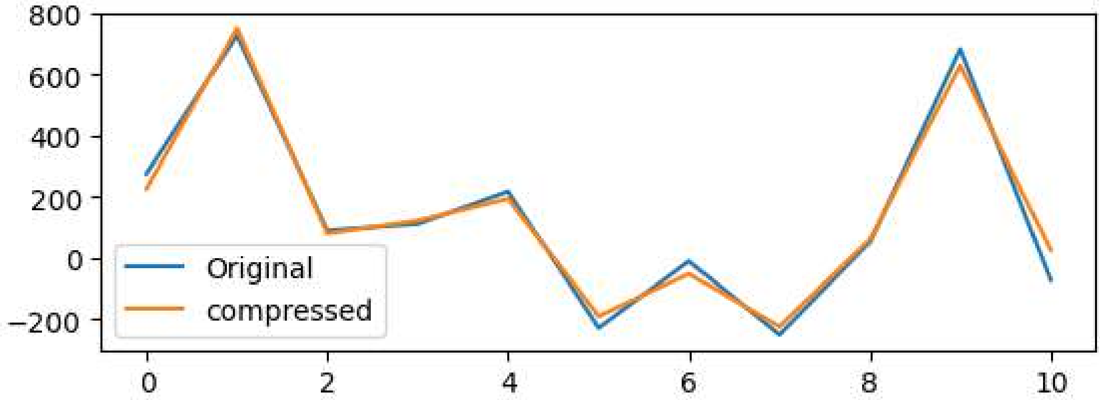
Threshold coefficients

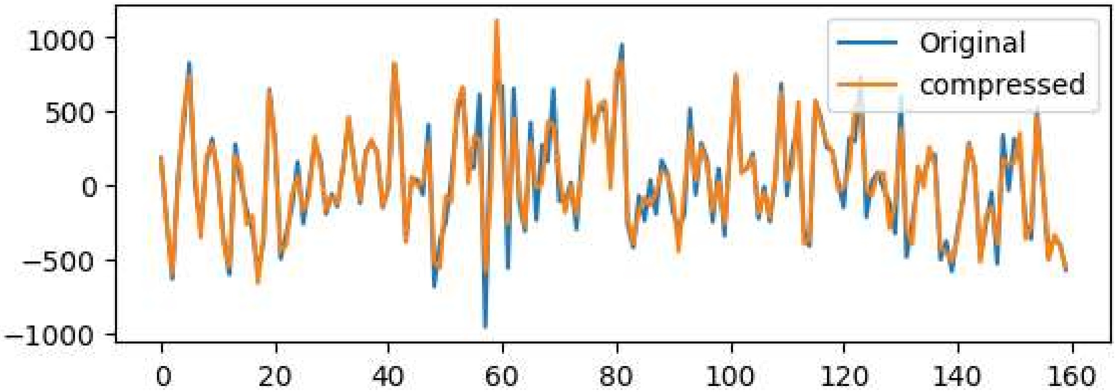
Threshold to see how much loss we get from removing the highest frequency coefficients

In [ ]:

```
# threshold coefficients
n_remove = 100
c_mat = getDCT(temp).astype(int)

c_mat_th = c_mat[:-n_remove,:,:] # remove n_remove highest freqs


temp_hat = getiDCT(np.pad(c_mat_th, ((0,n_remove), (0,0), (0,0)), 'constant', cc

# plot
fig2, axs2 = plt.subplots(2,1)
axs2[0].plot(temp[140:151,5])
axs2[0].plot(temp_hat[140:151,5])
axs2[0].legend(['Original', 'compressed'])
axs2[1].plot(temp[40:200,5])
axs2[1].plot(temp_hat[40:200,5])
axs2[1].legend(['Original', 'compressed'])
plt.show()

# compute SNDR
SNDR = 0
for i in range(temp.shape[1]):
    P_d = np.linalg.norm(temp[:,i])
    P_error = np.linalg.norm(temp[:,i] - temp_hat[:,i])
```

```
    SNDR += 1/(temp.shape[1]) * 20*np.log10(P_d/P_error)
print(f'Mean SNDR across channels: {SNDR:.2f} dB')
```






```
Mean SNDR across channels: 17.94 dB
```