In [1]:
```
import numpy as np
import scipy.linalg as sp
import math
import matplotlib.pyplot as plt
from helper import *
%matplotlib inline
```

1: Write a func that generates a random signal of length T

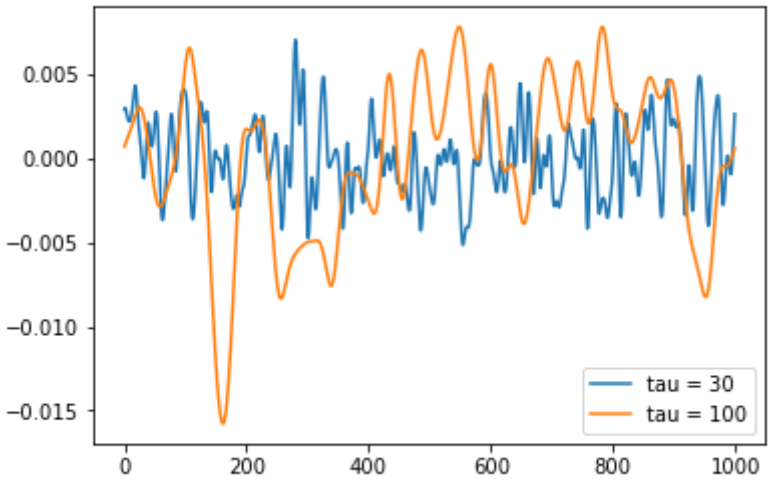
In [2]:
```
# function that creates a random signal with specified spectrum
def random_sig(T,epsilon):
    rand_sig = np.random.randn(T)
    return gaussian_spectrum_1D(rand_sig, epsilon)
```

A "random" signal was meant to be a white noise signal. Using functions like numpy.random.rand or numpy.random.normal is all that's needed here. Generate a Gaussian vector of the specified length with them. The helper function will take care of modifying the signal's frequency content, depending on the parameter epsilon.

2: use func from above...

In [3]:
```
T = 1000     #number of samples
tau_1 = 30
tau_2 = 100
# compute two signals with different epsilon
signal_1 = random_sig(T,1/math.pow(tau_1,2))
signal_2 = random_sig(T,1/math.pow(tau_2,2))
```

In [4]:
```
# normalize signals
signal_1 = signal_1 - signal_1.mean()
signal_2 = signal_2 - signal_2.mean()

# plot signals
plt.plot(signal_1,label='tau = 30')
plt.plot(signal_2, label='tau = 100')
plt.legend()
plt.show()

print("A higher tau leads to a signal with higher frequencies")
```




```
A higher tau leads to a signal with higher frequencies
```

3: perform linear SFA

In [5]:
```
# combine the vectors in a matrix
X = np.concatenate((signal_1[:,np.newaxis],signal_2[:,np.newaxis]),axis = 1).T
```

In [6]:
```
# compute covariance matrix on the data
C = np.cov(X)
# compute temporal derivative of the covariance matrix
C_diff = np.dot(np.diff(X),np.diff(X).T)/(T-1)
```

In [7]:
```
# compute eigenvecors and eigenvalues
eigval,eigvec = sp.eig(C_diff,C)
```

In [8]:
```
eigval = eigval.real
eigvec = eigvec.real

# normalize eigenvectors --> they are already normalized
np.linalg.norm(eigvec, axis = 0)
```

Out[8]:
```
array([ 1.,  1.])
```

In [9]:
```
print("eigenvectors:")
print(eigvec)
```

```
eigenvectors:
[[-0.99966112 -0.01905613]
 [-0.02603156  0.99981842]]
```

4: The eigenvectors determine how the input signals are mixed by the SFA to generate the new (slow) signals. What kind of mixture do you ideally expect given how the data was generated? What does the empirical mixture look like, judging from the extracted eigenvectors?

a mixture of normal distributions is expected, as the the data was generated with a gaussian specturm.

5: Repeat task 3 with different signal sample sizes (ranging logarithmically from 10 − 100000 in 20 steps) and store the resulting normalised eigenvectors (Hint: write another function that accepts as input the sample size and outputs the normalised SFA-eigenvectors). Plot the four individual components of the eigenvectors against the number of samples. How does the SFA-mixture change depending on the length of the input signal? Can you give an intuition why?

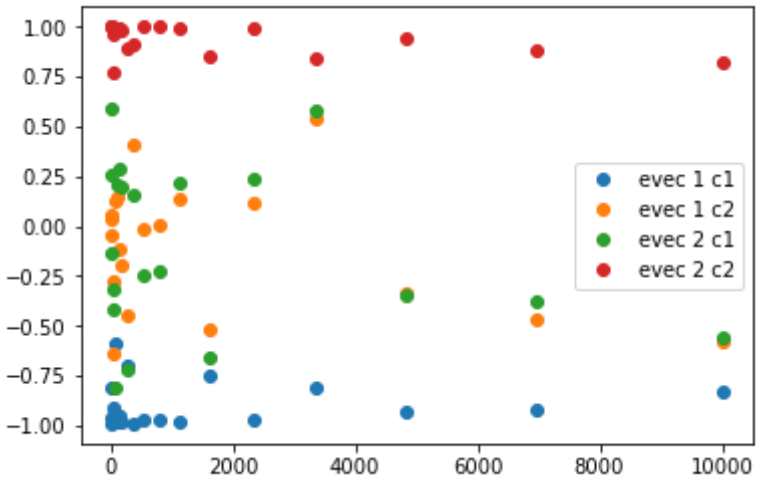
In [146]:
```
def create_X(T, tau_1=100, tau_2=100):
    signal_1 = random_sig(T,1/math.pow(tau_1,2))
    signal_2 = random_sig(T,1/math.pow(tau_2,2))
    signal_1 -= signal_1.mean()
    signal_2 -= signal_2.mean()
    X = np.concatenate((signal_1[:,np.newaxis],signal_2[:,np.newaxis]),axis = 1).T
    return X

def return_norm_eig_v(X):
    T = np.shape(X)[1]
    # compute covariance matrix on the data with zero mean
    C = np.cov(X)
    # compute temporal derivative of the covariance matrix
    C_diff = np.dot(np.diff(X),np.diff(X).T)/(T-1)
    # compute eigenvecors and eigenvalues
    eigval,eigvec = sp.eig(C_diff,C)
    #print(eigval.real)
    return eigvec
```

In [147]:
```
start = 1
stop = 4
num = 20

log_scale = np.logspace(start, stop, num=num, endpoint=True)
sample_sizes = log_scale.astype(int)

eigv_matrix = np.zeros([len(sample_sizes), 2, 2])
# samplesize * index of eigv * values of an eigvector with spec index:
# eg: eigv_matrix[-1,0,1] is the 2nd coordinate of the first normalized eigv of the signal with sample si
ze 10000 (last in sample_sizes vector (-1))
for i,sample in enumerate(sample_sizes):
    eigv_matrix[i,:,:] = return_norm_eig_v(create_X(sample))

eigv_matrix.shape
```

Out[147]: (20, 2, 2)

In [148]:
```
s = sample_sizes
m = eigv_matrix

#plt.figure(figsize=(40,20))
plt.plot (s,m[:,0,0],"o",label = "evec 1 c1")
plt.plot (s,m[:,0,1],"o",label = "evec 1 c2")
plt.plot (s,m[:,1,0],"o",label = "evec 2 c1")
plt.plot (s,m[:,1,1],"o",label = "evec 2 c2")
plt.legend()
```

Out[148]: <matplotlib.legend.Legend at 0xa67106ac>



For longer sample sizes the eigenvectors anneal [-1,0] and [0,1]. SFA gets more robust, so a high temporal resolution is suggested.SFA gets better as the sample-size grows

6. How do the weights depend on the Tau's: τ_1 is depending on the first eigenvalue and τ_2 is depending on the second. The larger the difference of the $\tau's$ the less samples are needed for a good SFA

Exercise 2

1. Generate a 1D-image

In [132]:
```
T_im = 10000
eps_im = 0.04

im = random_sig(T_im, eps_im)

im.shape
#np.array([im[0:100]]).shape
```

Out[132]:
```
(10000,)
```

2. extract time-dependent receptive field input

In [112]:
```
#super slow... idk why
def extract_window(im,l):
    #construct matrix for time-dependent receptive field (tdrf)
    #tdrf = np.zeros((l,len(im)-100))
    tdrf = np.array([im[0:l]]).T

    for i in range(1, len(im)-l):
        #tdrf[:,i] = im[i:i+l]
        tdrf = np.concatenate((tdrf, np.array([im[i:i+l]]).T) , axis = 1)

    return tdrf
```

In [113]:
```
def extract_window1(im,l):
    #construct matrix for time-dependent receptive field (tdrf)
    tdrf = np.zeros((l,len(im)-100))

    for i in range(len(im)-l):
        tdrf[:,i] = im[i:i+l]
        #print(i," - ",i+l)
    return tdrf
```

In [118]:
```
tdrf = extract_window1(im,100)
#normalize -> average input is zero
tdrf = (tdrf.T - tdrf.mean(axis = 1)).T

#tdrf.mean(axis=1)
tdrf.shape
```

Out[118]: (100, 9900)

3. Extrace eigenvalues and eigenvectors

In [131]:
```
C2 = np.cov(tdrf)
C2_diff = np.dot(np.diff(tdrf),np.diff(tdrf).T)/(T_im-1)

eva,eve = sp.eig(C2_diff,C2)
eve.shape
```

Out[131]: (100, 100)

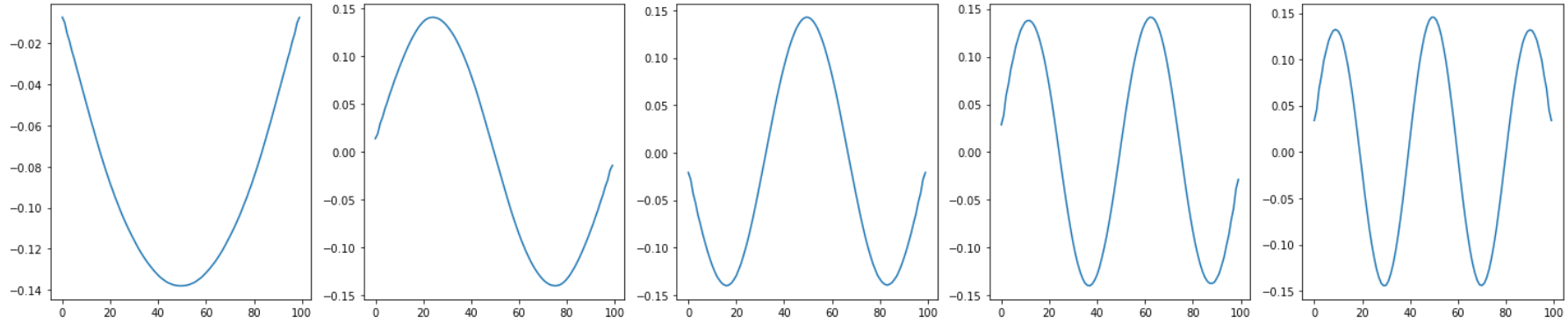
4. Plot 5 eigenvectors corresponding to the five smallest eigenvalues

In [128]:
```
sort_arg = np.argsort(eva)
five = eve[:,sort_arg[0:5]]
five.shape
```

Out[128]: (100, 5)

In [127]:
```
fig,ax = plt.subplots(nrows=1,ncols=5,figsize=(25,5))
ax[0].plot(five[:,0])
ax[1].plot(five[:,1])
ax[2].plot(five[:,2])
ax[3].plot(five[:,3])
ax[4].plot(five[:,4])
```

Out[127]:
```
[<matplotlib.lines.Line2D at 0xa7b99a6c>]
```



5. Interpretation

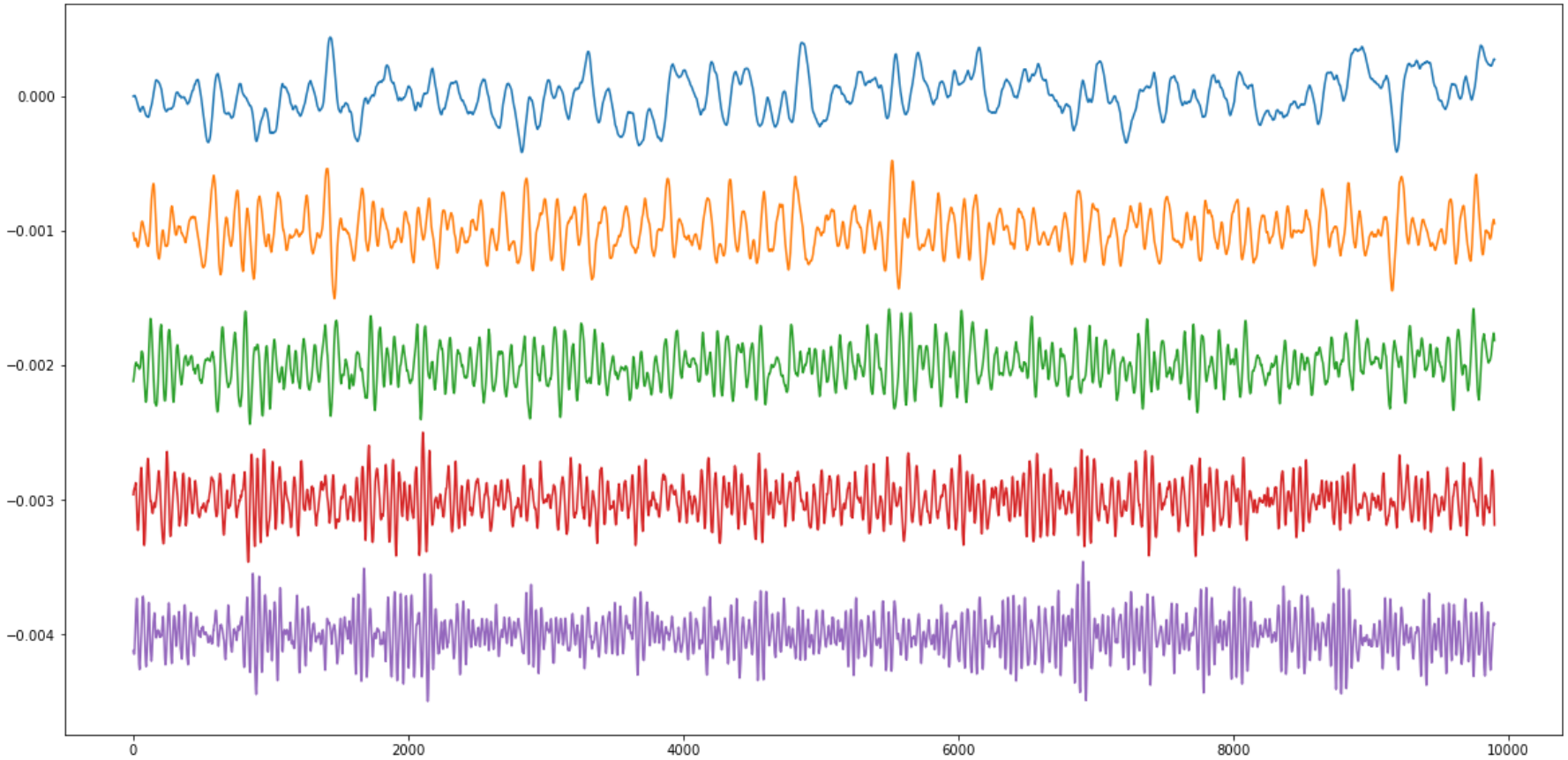
The slower the SFA-component, the smaller the frequency.That means that this feature is changing less with time. primary sensory signals like respones of retinal receptors are sensitive to very small changes in enviroment that are not relevant for the brain. the internal representation of the enviroment in the brain should vary on a slow time scale.

6. project original signal

In [149]:
```
offset = 0.001
Y0 = np.dot(five[:,0].T,tdrf)
Y1 = np.dot(five[:,1].T,tdrf) -offset
Y2 = np.dot(five[:,2].T,tdrf) -offset*2
Y3 = np.dot(five[:,3].T,tdrf) -offset*3
Y4 = np.dot(five[:,4].T,tdrf) -offset*4
```

In [150]:
```
plt.figure(figsize=(20,10))

plt.plot(Y0)
plt.plot(Y1)
plt.plot(Y2)
plt.plot(Y3)
plt.plot(Y4)
```

Out[150]: [<matplotlib.lines.Line2D at 0xa670458c>]



it looks like that the SFA worked. the upper graph which is corresponding to the slowest compontent has the lowest frequency. going downwards, the frequency is increasing.

In [ ]: