SC42150 - Statistical Signal Processing

Python Assignment I: Stock Market Analysis

Team members

- Leon Kehler (4834186)
- Petar Velchev (4349253)

Introduction

The goal of this assignment is to help you get a more practical understanding of some of the concepts of Statistical Signal Processing. Specifically, you will work with stochastic processes and determine their moments, stationarity and ergodicity properties, and also try to detrend the data.

In order to achieve those objectives, you will use the Python programming language, and specifically the modules numpy, scipy (for numerical processing) and matplotlib (for plotting). In case you are completely unfamiliar with Python, you should learn the basics before attempting this exercise. You are allowed to use numpy's implementations of common routines, such as mean or cov amongst other, but if you do so make sure you have read the documentation and are sure that they accomplish exactly what is taught in the lecture notes, as sometimes there are small differences in definitions and implementations.

Finally, you are also going to use the Jupyter Notebook file format, which might require installing some dependencies on your system. You can find instructions on installing Jupyter here.

Reporting

The report of this python exercise consists of this Jupyter Notebook file, with your answers added after each question. Answers will consist of code and/or text (markdown) cells. Please use the already indicated cells and format. You will **also** need to deliver a `PDF` version of this file.

In order to create a PDF version of this file, go to the top left, click on `File` then `Export Notebook As` and Choose `PDF`. If this feature is not supported by your system, you can also choose `LaTeX`. Then, you can use your preferred `LaTeX` compiler to create the `PDF`. After generation you PDF report, please double check if all answers/plots are correctly displayed.

Moreover, you will also need to send the `.npy` file with the signals used in this exercise. Don't forget to edit your team member's name and student number above. You will submit one Jupyter notebook per team.

Stock market analysis

Lately you've been fascinated by the world of trading so you decided to learn more about it and see if you can learn any good strategies by examining the trading data and looking for patterns.

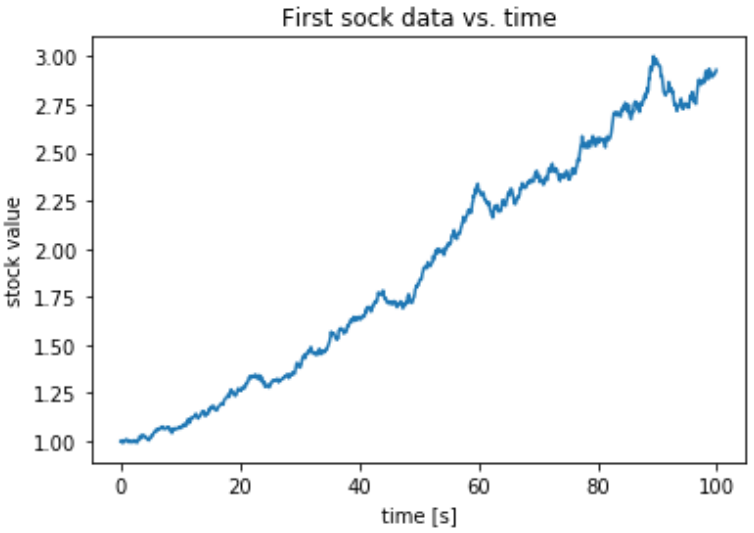
You are given a `.npy` file containing historical price data of various stocks. We will make the assumption that all the timeseries are just different realizations of the same stochastic process.

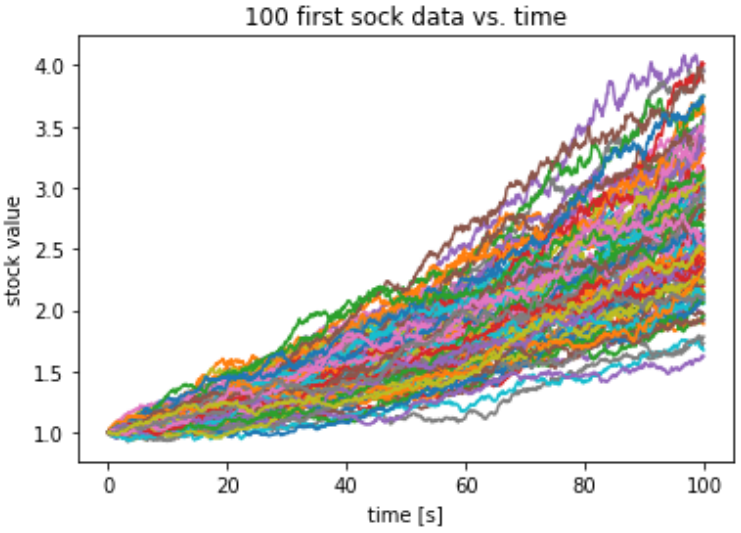
You will need to conduct the following steps to complete your investigation, each of which is worth 1 point:

1. (2 points) Import the `.npy` file provided using `np.load()` as indicated below. You should now have access to a $N \times K$ array, containing K timeseries of historical price data of various stocks, each of length N. Plot the first one and also, in a new figure, plot the first 100 of them together.

In [7]:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal

data = np.load('data2022.npz')
S =  data['S']
t =  data['t']

first = S[:,0]
first100 = S[:,:99]
plt.plot(t, first)
plt.title("First sock data vs. time")
plt.xlabel("time [s]")
plt.ylabel("stock value")
plt.show()
plt.plot(t, first100)
plt.title("100 first sock data vs. time")
plt.xlabel("time [s]")
plt.ylabel("stock value")

plt.show()
```



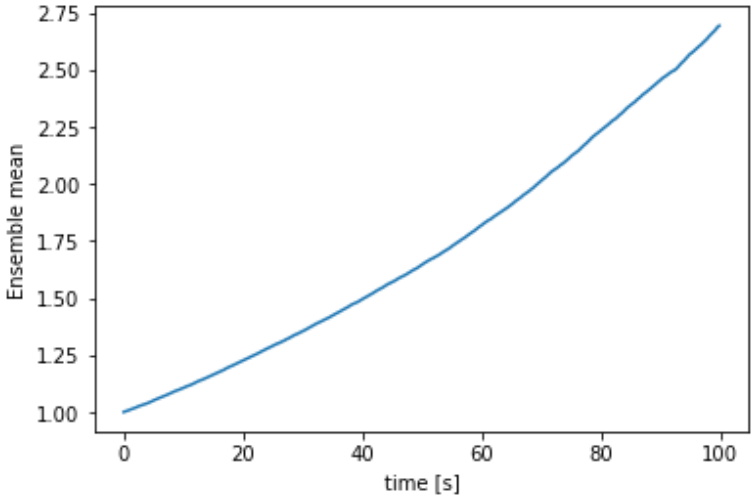


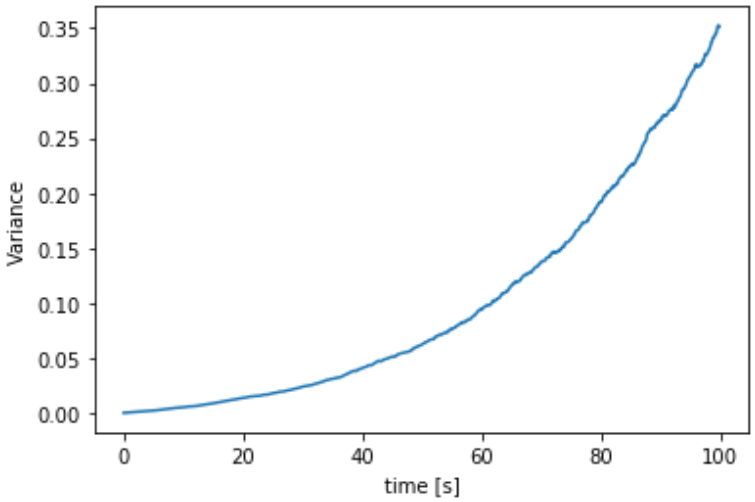
We will denote the generating stochastic process with $S_n \sim f_{S_n}$, where $n = t\Delta t$ refers to the time index. Further, we will denote its k-th realization with S_n^k.

2. (5 points) Compute the ensemble mean and variance and plot each one.

In [8]:

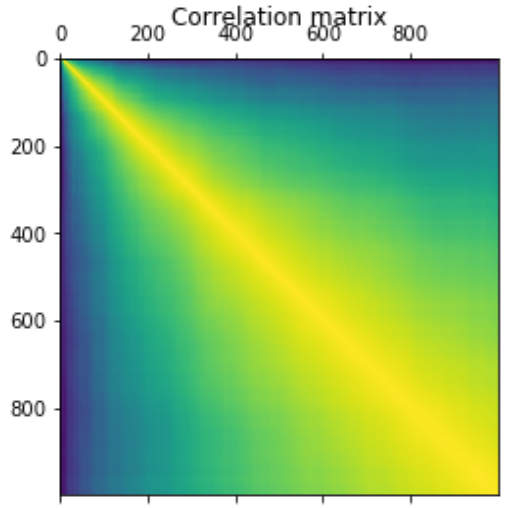
```
#ensemble mean
n = len(S[0])

def ensemble_mean(t):
    return 1 / n * np.sum(S[t,:])

t = np.arange(1, len(S[:,0]))
plt.plot(np.arange(0,99.9,0.1), [ensemble_mean(ti) for ti in t])
plt.ylabel("Ensemble mean")
plt.xlabel("time [s]")
plt.show()

# variance
# defined as E[(x-E[x])^2]
def variance(t):
    return 1/n * np.sum(S[t,:]**2) - ensemble_mean(t)**2

plt.plot(np.arange(0,99.9,0.1), [variance(ti) for ti in t])
plt.ylabel("Variance")
plt.xlabel("time [s]")

plt.show()

# needed for later question:
ens_mean = [ensemble_mean(ti) for ti in t]
ens_var = [variance(ti) for ti in t]
```





3. (10 points) Compute the correlation matrix R_S and visualize it using the `plt.matshow()` command. What can you deduce from that plot?

In [10]:
```
r_s = np.corrcoef(S)
plt.matshow(r_s)
plt.title("Correlation matrix")
plt.show()
```



> Enter answer here. The market follows some trend, thus, timewise close instances are closly related, whilst an early and a late time instance are least correlated. It cannot be deduced what the direction of the trend is, ie up or down, but we can

> see, that the trend direction is not changing significantly over our observation period.

4. (10 points) Can you determine whether the process S_n is WSS or not? Explain why.

> Enter answer here. There are three requirements on WSS:
>
> - The average is finite, this is true, because all realizations are finite.
> - The cross correlation depends only on the time shift, not the absolute times. This is true because the correlation matrix visualization displays a linear behavior.
> - The variance must be finite. This is true, as shown in the variance vs time plot.
>
> Thus, the process S_n is WSS.

A friend of yours who has been into trading for some time now, briefly looks at the data and tells you that the stock price changes can be explained by asset model like the following:

$$S_{n+1} = S_n e^{(\mu - 0.5\sigma^2)\Delta t + \sigma\sqrt{\Delta t} Z_n}, \quad Z_n \sim \mathcal{N}(0,1) \tag{1}$$

However, you have no idea what this model represents and what values the parameters μ and σ should take. By inspecting the data you can at least infer that $\Delta t = 0.1$ time units, so that's a start.

In your efforts to make sense of (1), you notice that if you divide S_{n+1} by S_n and then take the logarithm you end up with

$$\log\left(\frac{S_{n+1}}{S_n}\right) = (\mu - 0.5\sigma^2)\Delta t + \sigma\sqrt{\Delta t} Z_t \tag{2}$$

This looks promising so you decide to try this transformation on the data you have.

5. (3 points) Apply the transformation $X_{n+1} = \log(S_{n+1}/S_n)$ to your data. Is the resulting system BIBO stable?

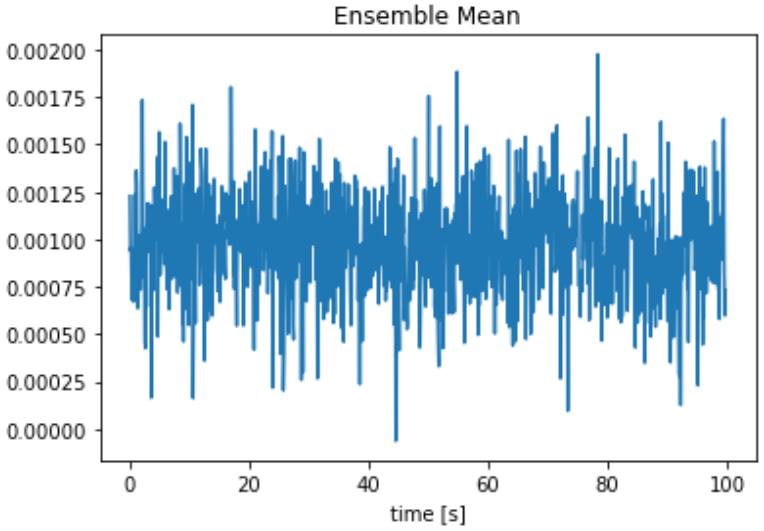
In [11]:
```
s_n = S[0:-1,:]
s_n1 = S[1:,:]


x_n1 = np.log(s_n1/s_n) # this denotes the natural logarithm in python
```

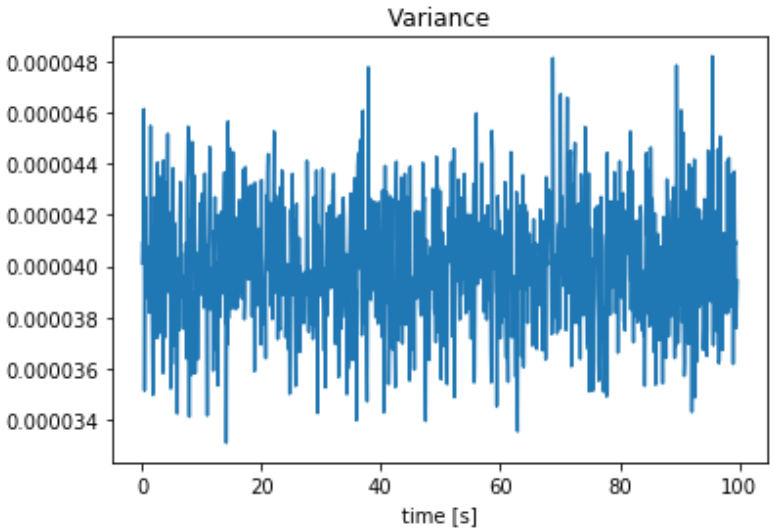
> Is this system BIBO stable?
> This system is BIBO stable, because S_{n+1} and S_n are both bounded, such that the ratio of them is also bounded, finally the log(x) is a bounded function.

6. (10 points) Compute and plot the ensemble average and variance for X_n. Next, compute and plot the time average and variance for a single realization of X_n, e.g. for $k = 1$. Based on the results, what can you say about the ergodicity and stationarity of X_n?

In [12]:
```
x_n = x_n1
#ensemble mean
n = len(x_n[0])
```

```
def ensemble_mean(t):
    return 1 / n * np.sum(x_n[t,:])

t = np.arange(1, len(x_n))
plt.plot(np.arange(0,99.8,0.1), [ensemble_mean(ti) for ti in t])
plt.title("Ensemble Mean")
plt.xlabel("time [s]")
plt.show()

# variance
# defined as E[(x-E[x])^2]
def variance(t):
    return 1/n * np.sum(x_n[t,:]**2) - ensemble_mean(t)**2

plt.plot(np.arange(0,99.8,0.1), [variance(ti) for ti in t])
plt.title("Variance")
plt.xlabel("time [s]")

plt.show()
```





> Enter answer here.
> Stationarity
> Stationarity is the notion of time invariance, as can be seen from the above plots, the variance and mean stay close to a constant value, thus, the change of the system does not vary in time.
> Ergodicity

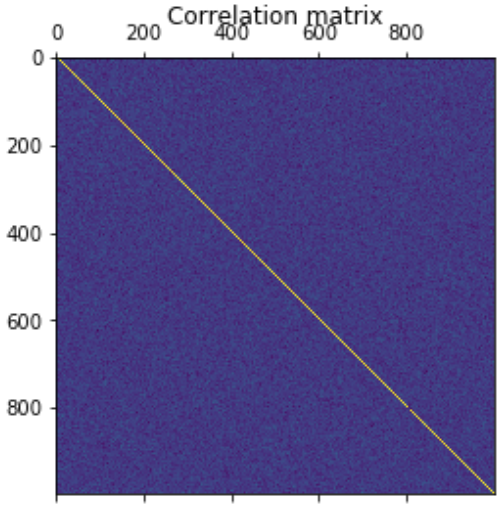
> Since the time series is ergodic, as all the stocks follow a trend and a typical stock can show the overall market behavior, also the log of the growth rate is ergodic.

7. (10 points) Compute the correlation matrix R_X and visualize it using the `plt.matshow()` command.

Can you determine whether the process X_n is WSS? If so, what are its first moments, i.e. mean and variance?

Express $\mathbb{E}(X)$ and $\mathrm{Var}(X)$ analitically in terms of μ, σ and Δt.

In [14]:
```
R_X = np.corrcoef(x_n)
plt.matshow(R_X)
plt.title("Correlation matrix")
plt.show()
mean123 = 1/(len(x_n) * 500) * np.sum(x_n)
variance123 = 1/(len(x_n) * 500) * np.sum(x_n**2) - mean123**2
print(f"Mean is {round(mean123, 5)}, matching the above ensemble mean graph")
print(f"Variance is {round(variance123, 5)}, also matching the above variance graph")
```




```
Mean is 0.00097, matching the above ensemble mean graph
Variance is 4e-05, also matching the above variance graph
```

> Enter answer here. The criteria hold for WSS for X_n, the mean must stay bounded which is shown by the mean graph of xn. The variance must be finite, which is also shown by the variance graph of xn. Finally, the correlation must only depend on the time difference, ie the correlation matrix above should have constant lines parallel to the diagonal. This seems not to be the case, but they are all 0, so 0 is also constant.
> Thus, the process is WSS

$$x_n = \log\left(\frac{S_{n+1}}{Sn}\right) = \left(\mu - 0.5\sigma^2\right)\Delta t + \sigma\sqrt{\Delta t}Z$$

$$\mathbb{E}(x_n) = \frac{1}{N}\sum x_n = \frac{1}{N}\sum\left[\left(\mu - 0.5\sigma^2\right)\Delta t + \sigma\sqrt{\Delta t}Z\right]$$

$$\mathrm{Var}(x_n) = E\left[\left|x_n - E\left[x_n\right]\right|^2\right] = E\left[x_n^2\right] - \left(E\left[x_n\right]\right)^2 = \frac{1}{N}\sum\left(x_n^2\right) - \left(\frac{1}{N}\sum x_n\right)^2$$

Where dividing by N means dividing by the product of the dimensions of the matrix x_n

8. (10 points) Choose one of the transformed timeseries X_n, compute its auto-correlation function $r_x(k)$ and visualize it. Verify whether **Properties 5.8 - 5.10** apply to this process.

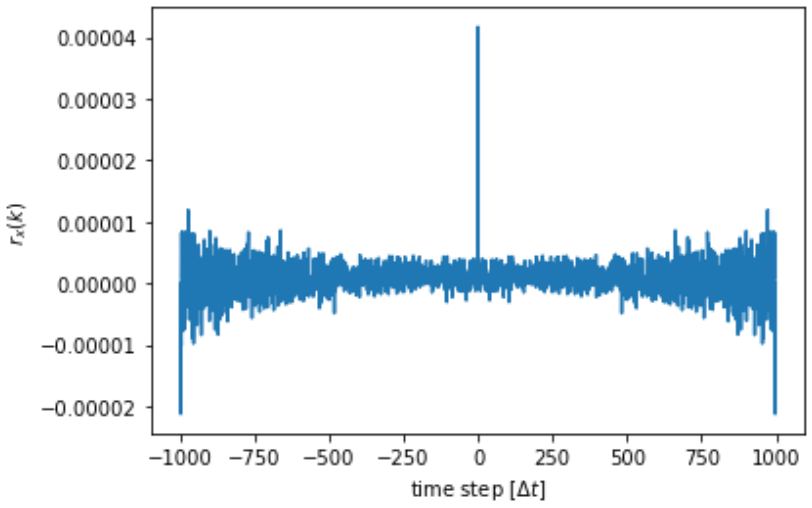
Note: if the signal $x(n)$ is auto-correlation ergodic, the auto-correlation function $r_x(k)$ can be estimated as (if we have only values $x_1, x_2, \ldots, x_N$):

$$r_x(k) = \frac{1}{N-k} \sum_{i=k+1}^{N} x(i)x^*(i-k)$$

You may also use Numpy's built in functions such as numpy.correlate(), though you should be careful to read the documentation beforehand.

Note: if you are unfamiliar with python, take care when copying `np.arrays` . Read more here.

In [15]:
```
t_series = x_n[:,0]
N = len(t_series)
ks = np.arange(- t_series.shape[0]+1, t_series.shape[0], 1)
auto_corr = []

for k in ks:
    a_n = 0
    for i in np.arange(np.abs(k) + 1, N):
        a_n += t_series[i] * t_series[i - np.abs(k)]
    auto_corr.append(1 / (N - np.abs(k)) * a_n)

plt.plot(ks, auto_corr)
plt.xlabel("time step [$\Delta t$]")
plt.ylabel("$r_x(k)$")
plt.show()
```



> Enter answer here.
> Property 5.8 asks for symmetry, ie $r_x(k) = r_x^*(-k)$, this is true as our r_x is fully real and the plot above clearly shows symmetry with respect to the y axis.
> Property 5.9 is about non-negative variance, ie $r_x(0) \geq 0$ This clearly true according to the plot above.
> Property 5.10 is about the maximum, ie $r_x(0) \geq |r_x(k)|$ for all k, this can also be deduced from the above plot.

9. (10 points) Calculate the Periodogram for the timeseries selected above. How does the Power Spectrum relate to the Autocorrelation function plotted above?

Can you show that the total power of the signal is equal to the $r_x(0)$ value computed above? (**Note:** the value may not be precisely equal because you are using limited data sets and

samples)

Hint: be careful with the indexing. You can use *numpy.fft.fft()* and *numpy.fft.fftfreq()* functions.

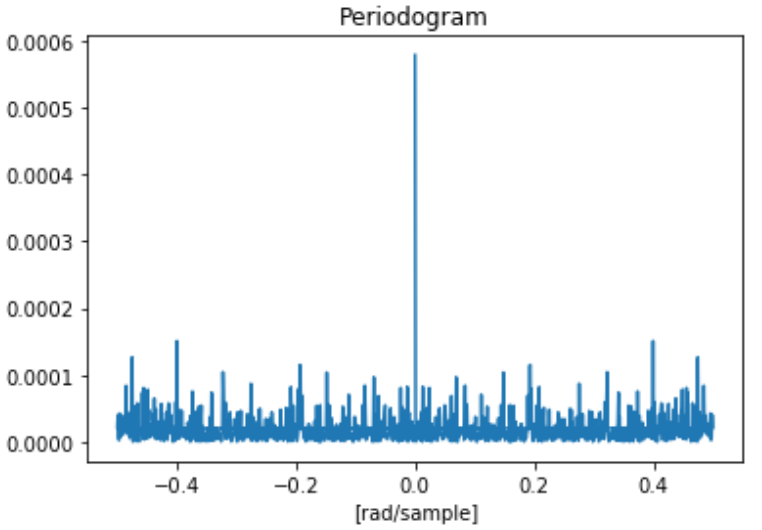
Hint 2: the resulting periodogram may not be represented in $\mathrm{rad/sample}$, but rather in $\mathrm{cycles/unit\ time}$. Be careful with the representation of your periodogram when calculating its integral / sum.

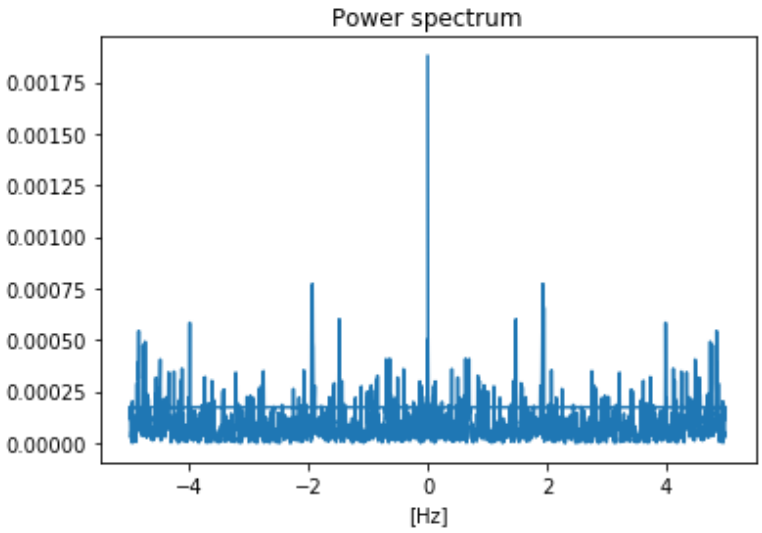
> The autocorrelation maps to the power spectrum, by taking the former's fourier transform.
> This formula is not enumarated in the reader, but is:
> $P_x\left(e^{j\omega}\right) = \sum_{k=-\infty}^{\infty} r_x(k)e^{-j\omega k}$ The inverse formula is enumerated in the reader as formula 5.11

In [21]:

```
# #periodogram
dt = 0.1
N = len(t_series)
sampling_rate = 0.1
#time = np.arange(-len(t_series)/2, len(t_series)/2, 1/sampling_rate)
fourier_transform = np.fft.fft(t_series)
abs_fourier_transform = np.abs(fourier_transform)
power_spectrum = 1/(2*N +1) * np.square(abs_fourier_transform)
frequency = np.fft.fftfreq(N)
plt.title("Periodogram")
plt.xlabel("[rad/sample]")
plt.plot(frequency, power_spectrum)

#power spectrum using correlation
plt.figure()
N_correl = len(auto_corr)
correl_fr = np.fft.fftfreq(N_correl , dt)
correl_fft = np.fft.fft(auto_corr)
plt.title("Power spectrum")
plt.xlabel("[Hz]")
plt.plot(correl_fr, abs(correl_fft))

power = np.sum(power_spectrum) / (n)
corr_zero = np.max(auto_corr)
print('Total Power =', power)
print("rx at k=0 =", corr_zero)
```

```
Total Power = 4.163868452094188e-05
rx at k=0 = 4.1686999965501665e-05
```





According to (2), X_t should be Gaussian White Noise since Z_t is a Gaussian ZMWN. However, since you are only given data, it's good that you verify this. Remember that a stochastic process is Gaussian if the joint distribution of any combination of elements of X_n is Gaussian. For now we will just take two random time instances to verify this, but feel free to try other combinations as well.

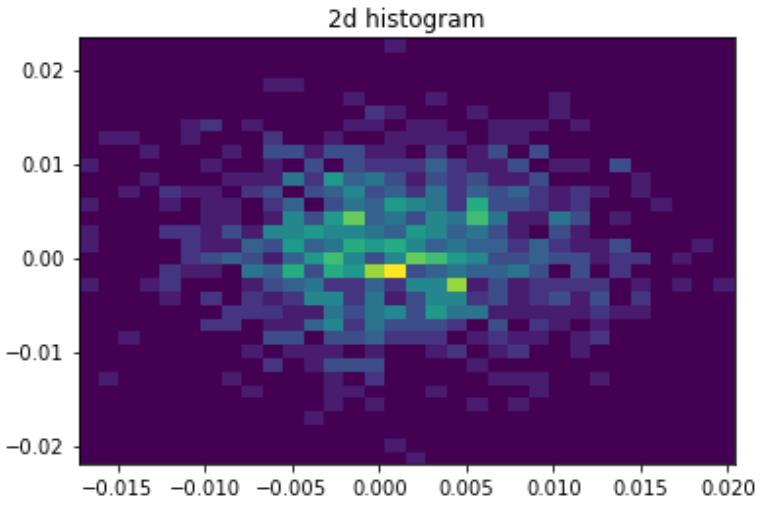
10. (5 points) Take two time instances of X_n, e.g. for $n = \{30, 70\}$, and plot their 2d histogram across K realizations $\{X_n^k\}_{k=1}^K$ by using $\sqrt{K}$ bins. Can you adequately fit a multivariate normal distribution on this histogram?

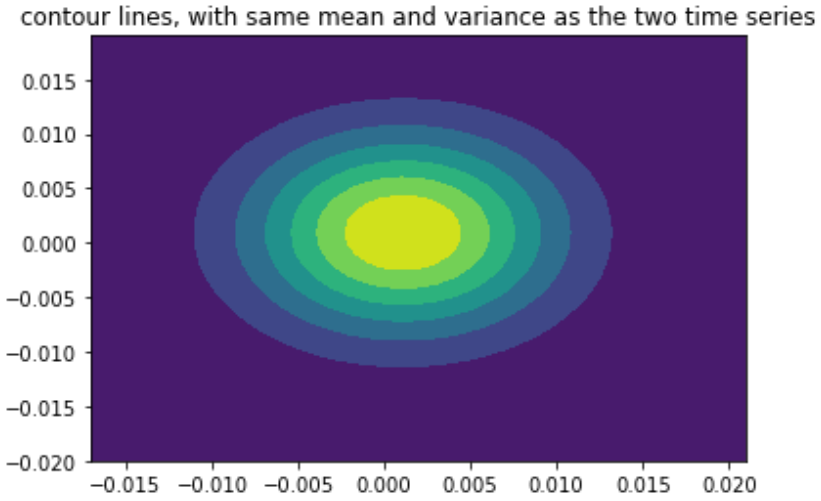
Hint: You may want to look into multivariate_normal.pdf() to help plot the levels of fitting distribution in 2D. To plot the experimental values, you may use plt.hist2d().

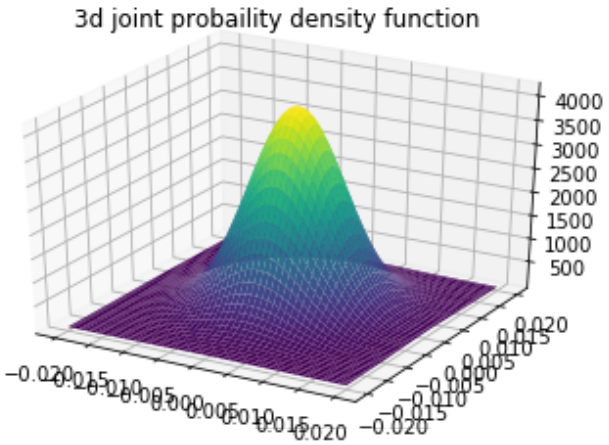
In [22]:

```
from mpl_toolkits.mplot3d import Axes3D

s1 = x_n[:, 30]
s2 = x_n[:, 70]

mu_s1 = np.mean(s1)
var_s1 = np.var(s1)
mu_s2 = np.mean(s2)
var_s2 = np.var(s2)
a = int(round(np.sqrt(s1.size)))
collect = plt.hist2d(s1, s2, bins=a)
plt.title("2d histogram")
```

```
x, y = np.mgrid[-0.017:0.021:.001, -.02:.02:.001]
pos = np.dstack((x, y))
rv = multivariate_normal([mu_s1, mu_s2], [[var_s1, 0], [0, var_s2]]) # mean, cov,
fig2 = plt.figure()
ax2 = fig2.add_subplot(111)
ax2.contourf(x, y, rv.pdf(pos))
plt.title("contour lines, with same mean and variance as the two time series")


x = np.linspace(-0.02,0.02,100)
y = np.linspace(-0.02,0.02,100)
xx, yy = np.meshgrid(x,y)
pos = np.empty(xx.shape + (2,))
pos[:, :, 0] = xx; pos[:, :, 1] = yy
rv = multivariate_normal([mu_s1, mu_s2], [[var_s1, 0], [0, var_s2]])

fig = plt.figure()
ax = fig.add_subplot(projection='3d')
ax.plot_surface(xx, yy, rv.pdf(pos),cmap='viridis',linewidth=0)
plt.title("3d joint probaility density function")
plt.show()
```



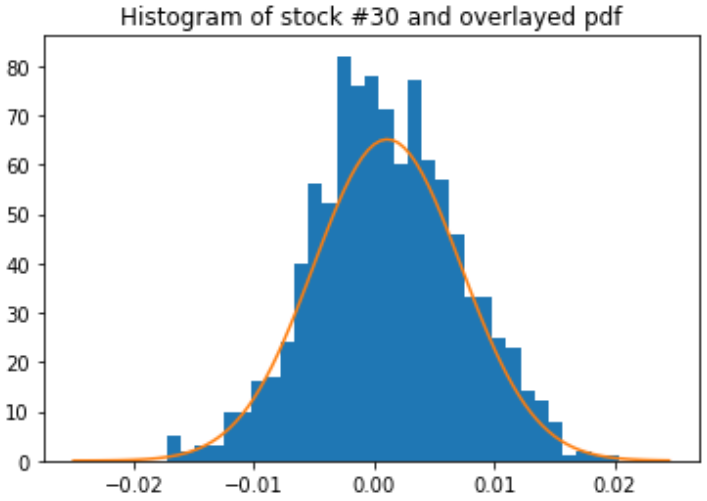




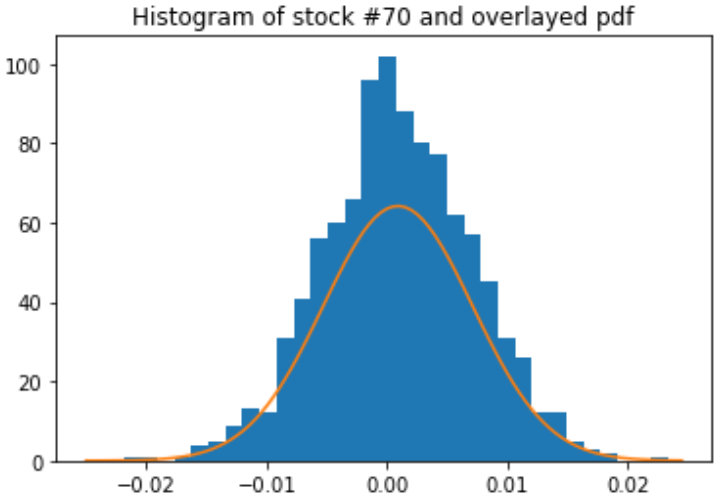
11. (5 points) Now verify that the marginals of the above distribution are also Gaussians, that is take the (1d) histograms of the above time instances (with $\sqrt{K}$ bins), plot them, and also fit the correspoding PDFs on top.

Hint: You may want to look into multivariate_normal.pdf() to help plot the levels of fitting distribution in 2D. To plot the experimental values, you may use plt.hist2d().

In [23]:
```
collect = plt.hist(s1, bins=a, log=False)
x = np.linspace(-0.025, .025, 100, endpoint=False)
y = multivariate_normal.pdf(x, mean=mu_s1, cov=var_s1)
plt.plot(x, y)
plt.title("Histogram of stock #30 and overlayed pdf")
plt.show()
collect = plt.hist(s2, bins=a, log=False)
x = np.linspace(-0.025, .025, 100, endpoint=False)
y = multivariate_normal.pdf(x, mean=mu_s2, cov=var_s2)
plt.title("Histogram of stock #70 and overlayed pdf")
plt.plot(x, y)
```



Out[23]: [<matplotlib.lines.Line2D at 0x226ddf676a0>]



From (1), it follows that the first two statistical moments can be computed analytically:

$$\mathbb{E}\left[S_n\right] = S_0 e^{\mu n \Delta t} \tag{3}$$

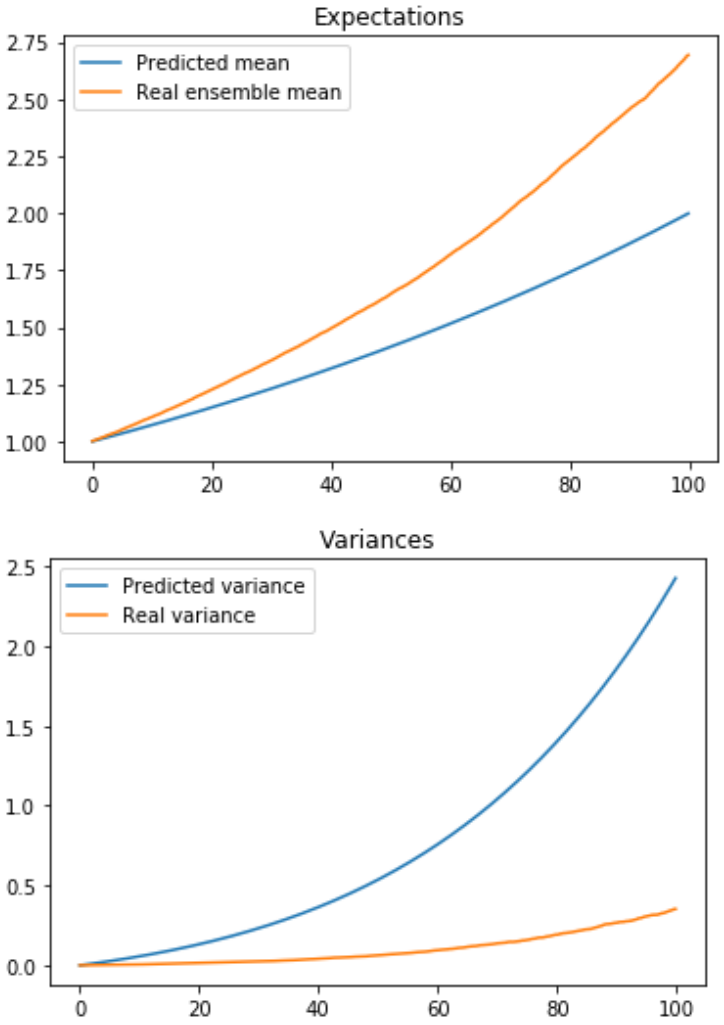
$$\mathrm{Var}\left[S_n\right] = S_0^2 e^{2\mu n \Delta t}\left(e^{\sigma^2 n \Delta t} - 1\right) \tag{4}$$

Having computed the ensemble mean and variance of X_n you should be able to determine estimates for the values of μ and σ.

12. (10 points) Using the results from Question 7, compute estimates for μ and σ. Use those estimates to calculate the moments (3)-(4) and plot those along with the ensemble moments of S_n of Question 2.

In [24]:

```
s0 = 1
n = 1000
dt = 0.1
mu = np.log(mean123 / s0) / (n * dt)
var =  np.sqrt((np.log(variance123 / (s0**2 * np.exp(2 * mu * n * dt))) + 1)/(n * dt))

t = np.arange(0, 99.9, .1)
expectation_sn = s0 * np.exp(np.abs(mu) * t * dt)
plt.title("Expectations")
plt.plot(t, expectation_sn, label="Predicted mean")
plt.plot(t, ens_mean, label="Real ensemble mean")
plt.legend()
plt.show()
variance_sn = s0**2 * np.exp(2 * np.abs(mu) * t * dt) * (np.exp(var**2 * t * dt) - 1)
plt.plot(t, variance_sn, label="Predicted variance")
plt.plot(t, ens_var, label="Real variance")
plt.title("Variances")
plt.legend()
plt.show()
```



13. (10 points) Having done all the above, you should now be in a position where you can discuss whether the model your friend suggested was good enough to describe the data.

> Enter answer here. The model seems to have predicted the mean quite well, with an inaccuracy of <40% at the final time step. However, the variance was not very well predicted. Yet, the overall trend of the stocks was captured.