Data Driven Models for Engineering Problems
1st Assignment

Nikos Stamatis
MSc Student
Graduate Programme in Data Science and Machine Learning
SN: 03400115
nikolaosstamatis@mail.ntua.gr

The most important chunks of code used in our simulation can be found in the Appendix, or even within the main body, rather scarcely. A Jupyter notebook with the complete code has also been included in our submission, separately from this text.

Exercise A

Number of terms retained justification: By the Karhunen-Loeve theorem, we can decompose each realization as

$$\begin{aligned} X(t,\omega) &= \sum_{n=1}^{\infty} \sqrt{\lambda_n}\phi_n(t)\xi_n(\omega) \\ &= \sum_{n=1}^{M} \sqrt{\lambda_n}\phi_n(t)\xi_n(\omega) + \sum_{n=M+1}^{\infty} \sqrt{\lambda_n}\phi_n(t)\xi_n(\omega), \end{aligned}$$

where $(\xi_n)_n$ is an i.i.d. sequence of standard normal distributions. Let $\tilde{X}(t,\omega) = \sum_{n=1}^{M} \sqrt{\lambda_n}\phi_n(t)\xi_n$ denote the part of the sum that we will actually simulate and $e_M(t,\omega) = \sum_{n=M+1}^{\infty} \sqrt{\lambda_n}\phi_n(t)\xi_n(\omega)$ denote the error part. Our estimation $\tilde{X}$ for X has the property that $\|\tilde{X} - X\|_2 = \|e_M\|_2$, so in order for $\tilde{X}$ to be close to X, we need to pick an M large enough so that the $\|e_M\|_2$ will be acceptably small.

Since $\tilde{X}$ is created using the random variables $(\xi_n)_{n=1}^{M}$, there is an additional potential source of error: If we are unlucky and the ξ_n's are relatively large for large values of n, then the same may hold for $\|e_M\|_2$ regardless of our initial choice for M. However, given how fast $(\lambda_n)_n$ converges to zero, obtaining such large values for ξ_n's is rather unlikely. Thus, although we cannot assure that $\|e_M\|_2$ will always be acceptably small, we can assure that it will be small with high probability. So we may rephrase our goal as follows:

Problem 1. *For any given $\delta, \varepsilon > 0$ find an $M \in \mathbb{N}$ such that $\|e_M\|_2 < \delta$ with probability at least $1 - \varepsilon$.*

For the probabilistic aspect of our problem, we will rely on the Laurent-Massart Lemma, which provides concentration bounds for linear combinations of chi-squared distributions:

Lemma 2. *[LM00, Lemma 1] Let $Y_1, \dots, Y_D$ be i.i.d. standard normal random variables and $(a_i)_{i=1}^{D}$ be non-negative real numbers. For the random variable $Z = \sum_{i=1}^{D} a_i(Y_i^2 - 1)$ the following inequalities hold for any $x > 0$:*

$$\begin{aligned} P(X \geq 2\|a\|_2 \sqrt{x} + 2\|a\|_\infty x) &\leq e^{-x}, \\ P(X \leq -2\|a\|_2 \sqrt{x}) &\leq e^{-x}. \end{aligned}$$

Combining the last two inequalities, we also obtain that

$$P(|X| \geq 2\|a\|_2 \sqrt{x} + 2\|a\|_\infty x) \leq 2e^{-x}. \tag{1}$$

Proposition 3. *Consider a zero-mean stationary Gaussian field on $D = [-a, a]$ with correlation function $C(t,s) = e^{\frac{|t-s|}{b}}$. Then, for every $\delta, \varepsilon > 0$, we have that $\|e_M\|_2^2 < \delta$ with probability at least $1 - \varepsilon$ for every $M \geq \left(\frac{2a^2}{3b\pi^2} \frac{2\sqrt{-\ln(\varepsilon/2)} - 2\ln(\varepsilon/2) + 1}{\delta} \right)^{1/3} + 1$.*

Proof. Recall that for some fixed orbit, $e_M(t) = \sum_{n=M+1}^{\infty} \sqrt{\lambda_n}\phi_n(t)\xi_n$ where the eigenvalues λ_n and eigenfunctions $\phi_n(t)$ are described in [PG18, p. 33] and the ξ_i's are realizations of independent standard normal random variables. The L_2-norm of e_M is equal to

$$\begin{aligned} \|e_M\|_2^2 &= \sum_{n=M+1}^{\infty} \lambda_n \xi_n^2 \\ &= \sum_{n=M+1}^{\infty} (\lambda_n \xi_n^2 - 1) + \sum_{n=M+1}^{\infty} \lambda_n. \end{aligned}$$

Let λ denote the sequence $\lambda = (\lambda_{M+1}, \lambda_{M+2}, \dots)$. By the Laurent-Massart Lemma,[1] we have that

$$P(\|e_M\|^2 \geq 2\|\lambda\|_2 \sqrt{x} + 2\|\lambda\|_\infty x + \|\lambda\|_1) \leq 2e^{-x}$$

[1] Here our sequence of scalars $(\lambda_n)_n$ is not finite, but $\|\lambda_n\|_2 \to \|\lambda\|_2$ and it is easy to see that inequality (1) still holds using a simple limit argument.

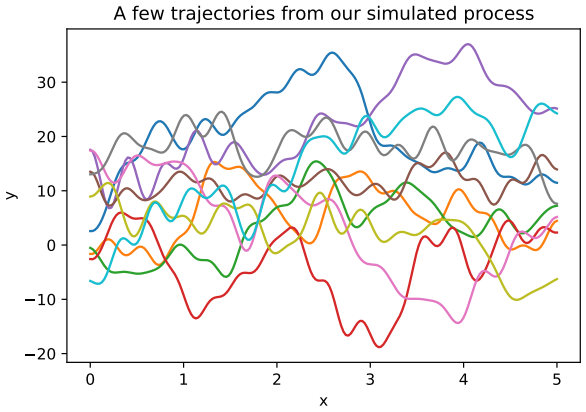

Figure 1. Sample trajectories from the stochastic field $E(x) = 10(1 + f(x))$.

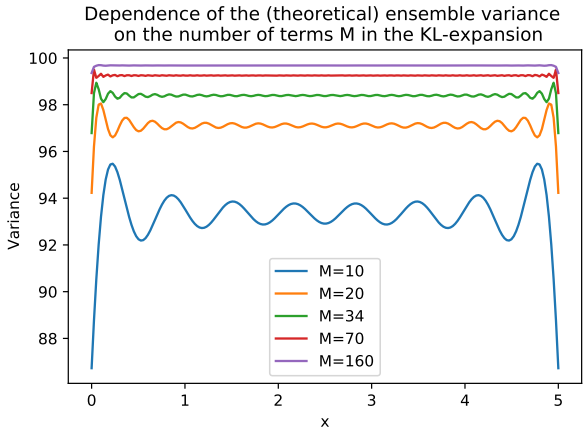

Figure 2. The theoretical ensemble variance $V[X(t)] = v_M(t) = \sum_{n=1}^{M} \lambda_n \phi_n(t-2.5)^2$ for various values of M. As M increases, v_M converges to the constant function $v_\infty \equiv 100$.

for every $x > 0$. In particular, since the ℓ_1-norm always dominates the rest of the ℓ_p-norms, we obtain that

$$P\left(\|e_M\|^2 \geq \|\lambda\|_1 (2\sqrt{x} + 2x + 1)\right) \leq 2e^{-x} \tag{2}$$

for every $x > 0$.

What is left to be done, is to determine an upper bound for $\|\lambda\|_1$ as an expression involving M. Notice that for every n, the value of ω_n is always larger than or equal to $\frac{(n-1)\pi}{a}$. Since $\lambda_n = \frac{2b}{1+\omega_n^2 b^2}$, this implies that

$$\lambda_n \leq \frac{2a^2}{\pi^2 b} \frac{1}{(n-1)^2}, \tag{3}$$

so

$$\begin{aligned} \|\lambda\|_1 &\leq \frac{2a^2}{b\pi^2} \sum_{n=M+1}^{\infty} \frac{1}{(n-1)^2} = \frac{2a^2}{b\pi^2} \sum_{n=M}^{\infty} \frac{1}{n^2} \\ &\leq \frac{2a^2}{b\pi^2} \int_{M-1}^{\infty} \frac{1}{x^2} dx \\ &= \frac{2a^2}{b\pi^2} \frac{1}{3(M-1)^3} := C_M. \end{aligned} \tag{4}$$

Therefore,

$$P\left(\|e_M\|^2 \geq C_M (2\sqrt{x} + 2x + 1)\right) \leq 2e^{-x} \tag{5}$$

also holds for every $x > 0$. The requirement that the probability of error should be less than ε implies that $2e^{-x} \leq \varepsilon$, so $x \geq -\ln\frac{\varepsilon}{2}$. We substitute this value of x in (5) to obtain:

$$P\left(\|e_M\|^2 \geq C_M \left(2\sqrt{-\ln\frac{\varepsilon}{2}} - 2\ln\frac{\varepsilon}{2} + 1\right)\right) \leq \varepsilon. \tag{6}$$

The last relation holds for any choice of M. We finish the proof by choosing an M such that $C_M\left(2\sqrt{-\ln\frac{\varepsilon}{2}} - 2\ln\frac{\varepsilon}{2} + 1\right) < \delta$, namely any integer M larger than

$$M \geq \left(\frac{2a^2}{3b\delta\pi^2}\left(2\sqrt{-\ln\frac{\varepsilon}{2}} - 2\ln\frac{\varepsilon}{2} + 1\right)\right)^{1/3} + 1 \tag{7}$$

will suffice. □

Simulation: In order to simulate from the field $E(t) = 10(1 + f(t))$ for $t \in [0,5]$, we initially simulate from $f(t)$ for $t \in (-2.5, 2.5)$ using the relevant theory. Then, each simulated orbit $X(t,\omega)$, $t \in [-2.5, 2.5]$, can be easily transformed into $Y(t,\omega) = 10 + 10X(t-2.5,\omega)$ for $t \in [0,5]$ which is then a realization from the desired stochastic field $E(t)$.

Applying Proposition 3 to our exercise for $a = 2.5$, $b = 2$ and for a choice of $\varepsilon = 10^{-4}$ and $\delta = 0.01$, we obtain that $M \geq 34$. It is worth stressing out that the previous bound is by no means a sharp one, and one could potentially achieve the same result with a smaller M.

```
def M_estimate(a, b, d, e):
  return(((2 * np.sqrt(-np.log(e/2)) -
           2 * np.log(e/2) + 1) *
           2 * a**2/(3 * d * b *
           np.pi**2))**(1. / 3.) + 1)
```

```
M_estimate(2.5, 2, 0.01, 0.01)
> 7.99

M_estimate(2.5, 2, 0.0001, 0.01)
> 33.45
```

We proceeded with the simulation, first simulating from $f(t)$ using $M = 34$ terms in the KL-expansion, and then translating the results so as to obtain the orbits of $E(x)$. In Figure 1 we depict a few orbits from our simulated processes.

Theoretical ensemble average and variance: In the general setting, each simulated trajectory has the form $X(t) = \sum_{n=1}^{M} \sqrt{\lambda_n}\phi_n(t)\xi_n$, where $(\xi_n)_n$ are i.i.d. standard

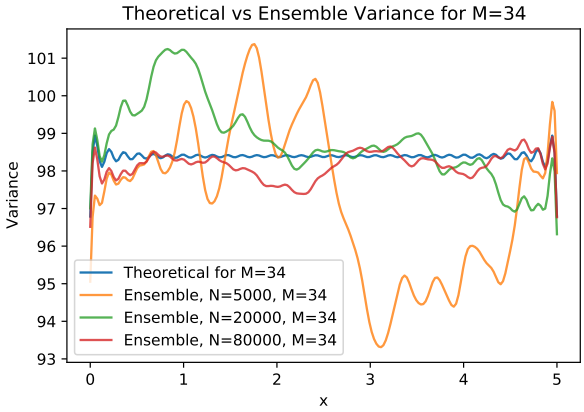

Figure 3. The ensemble variance converges a.s. to $v_{34}(t) = \sum_{n=1}^{34} \lambda_n \phi_n(t-2.5)^2$ as N increases.

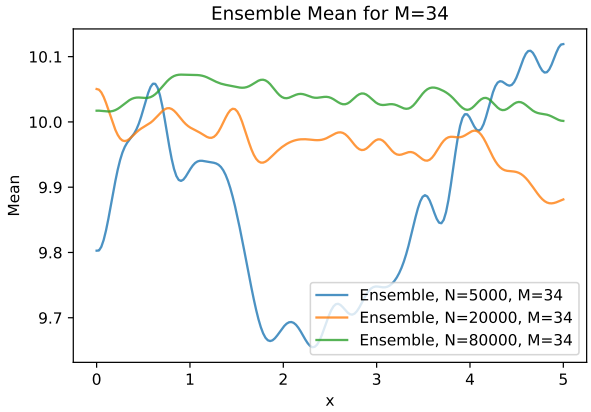

Figure 4. The ensemble mean converges a.s. to $m(t) = 10$ as N increases.

normal. From this formula we can immediately compute the expectation and the variance of $X(t)$ for each $t \in [-a, a]$:

$$\mathbb{E}[X(t)] = \sum_{n=1}^{M} \sqrt{\lambda_n}\phi_n(t)\,\mathbb{E}[\xi_n] = 0 \tag{8}$$

$$\begin{aligned}\mathrm{Var}[X(t)] &= \sum_{n=1}^{M} \lambda_n \phi_n(t)^2 \mathrm{Var}(\xi_n) \\ &= \sum_{n=1}^{M} \lambda_n \phi_n(t)^2. \end{aligned} \tag{9}$$

By the Strong Law of Large Numbers, the ensemble average and variance will converge to the functions $m(t) = 0$ and $v_M(t) = \sum_{n=1}^{M} \lambda_n \phi_n(t)^2$ respectively, for almost every $t \in [-a, a]$. For the stochastic field of the exercise the corresponding limits are $m(t) = 10$ and $v_M(t) = \sum_{n=1}^{M} \lambda_n \phi_n(t-2.5)^2$ for $t \in [0,5]$. Since all the terms appearing in the last sum are nonnegative, $(v_M)_{M\in\mathbb{N}}$ is an increasing sequence of functions in $[0,5]$ with $\lim_{M\to\infty} v_M(t) = \sum_{n=1}^{\infty} \lambda_n \phi_n(t-2.5)^2$. We expect the latter to be equal to $v_\infty(t) = 100$ for all $t \in [0,5]$, due to the stationarity of the underlying process.

In Figure 2 we show the theoretical variance $v_M(t)$ for various values of the number of terms M retained in the Karhunen-Loeve expansion. For our simulation we chose $M = 34$. In Figure 3 we compare the variances for the simulated process to the theoretical ones for various values of the number of experiments N. We can visually confirm that for large values of N, the ensemble variance approaches the theoretical function v_M. Similarly, the ensemble average converges to the function $m(t) = 10$ as N increases (Figure 4).

Exercise B

To compare the metrics of the simulated process with the theoretical ones, we first need to compute the autocorrelation function R_X using the formula $R_X(\tau) = \int_0^\infty G_X(\omega)\,d\omega$:

$$\begin{aligned} R_X(\tau) &= \int_1^2 (\omega - 1)\cos\omega\tau\, d\omega + \int_2^3 (3-\omega)\cos\omega\tau\, d\omega \\ &= \left[\frac{\cos\omega\tau}{\tau^2} + \frac{\omega\sin\omega\tau}{\tau}\right]_1^2 - \left[\frac{\cos\omega\tau}{\tau^2} + \frac{\omega\sin\omega\tau}{\tau}\right]_2^3 + \\ &\quad + \frac{3}{\tau}[\sin\omega\tau]_2^3 - \frac{1}{\tau}[\sin\omega\tau]_1^2 \\ &= \frac{2\cos 2\tau}{\tau^2} - \frac{\cos 3\tau}{\tau^2} - \frac{\cos\tau}{\tau^2}. \end{aligned} \tag{10}$$

The process is zero-mean, stationary and its variance is equal to

$$\begin{aligned} \mathrm{Var}[X(t)] &= \lim_{\tau\to 0} R_X(\tau) \\ &= \lim_{\tau\to 0}\left(\frac{2\cos 2\tau}{\tau^2} - \frac{\cos 3\tau}{\tau^2} - \frac{\cos\tau}{\tau^2}\right) \\ &= -4 + \frac{9}{2} + \frac{1}{2} \\ &= 1 \end{aligned} \tag{11}$$

for every $t \in [0, 10]$.

A realization of $X(t,\omega)$ can be written as

$$X(t,\omega) = \sqrt{2}\sum_{n=0}^{\infty} A_n \cos(\omega_n t + \Phi_n(\omega)), \tag{12}$$

where $(\Phi_n)_n$ are i.i.d. standard normal. When simulating such a realization, we may only keep finitely many such terms. As in the previous exercise, one can determine the theoretical ensemble mean and variance using the formula $\tilde{X}(t,\omega) = \sum_{n=0}^{M} A_n \cos(\omega_n t + \Phi_n(\omega))$. It is easy to see that when $\Phi_n \sim N(0,1)$, then $\mathbb{E}[\cos(\omega_n t + \Phi_n)] = 0$ and

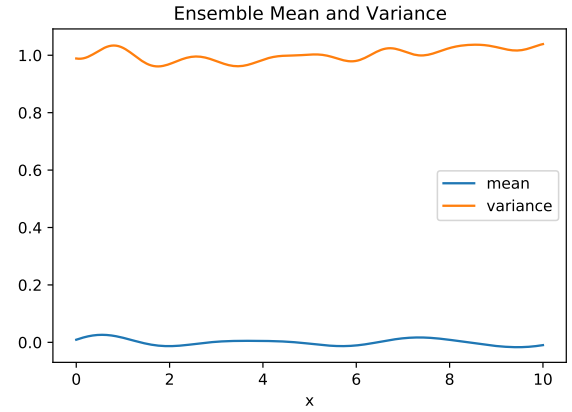

Figure 5. The ensemble mean and variance converge a.s. to $m(t) = 0$ and $v(t) = 1$ respectively as N and M increase. Our simulation was based on $N = 5000$ realizations, each one keeping $M = 500$ terms in the spectral series expansion.

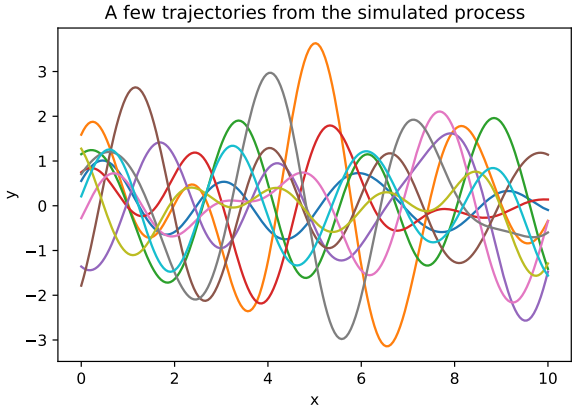

Figure 6. Some simulated trajectories from our process.

$\mathrm{Var}[\cos(\omega_n t + \Phi_n)] = \frac{1}{2}$. By the independence of $(\Phi_n)_n$ we obtain that $\mathbb{E}[X(t)] = 0$ and $\mathrm{Var}[X(t)] = \sum_{n=1}^{M} A_n^2 = \sum_{n=1}^{M} G_X(\omega_n)\Delta\omega$. As $M \to \infty$, the last quantity becomes the Riemman integral of G_X, which is equal to 1, as expected.

By the previous discussion, for large values of N and M[2], the ensemble average and variance will be close to $m(t) = 0$ and $v(t) = 1$, $t \in [0, 10]$ respectively. This can be confirmed in Figure 5.

The process is also ergodic in the autocorrelation [PG18, p. 7] since it is stationary with

$$\lim_{\tau\to\infty} R_X(\tau) = \lim_{\tau\to\infty}\left(\frac{2\cos 2\tau}{\tau^2} - \frac{\cos 3\tau}{\tau^2} - \frac{\cos\tau}{\tau^2}\right) = 0.$$

This implies that both the temporal average and variance should converge to the corresponding ensemble

[2]A large M allows for the sum $v_M(t) = \sum_{n=1}^{M} G_X(\omega_n)\Delta\omega$ to be close to the Riemman integral of G_X, whereas a large N allows for the sample variance to be close to v_M.

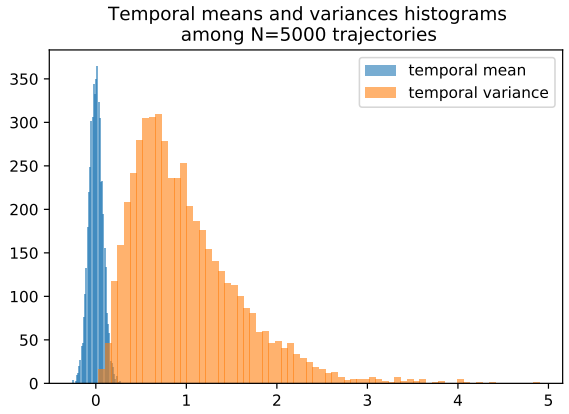

Figure 7. The distribution of the temporal mean and variance for our drawn sample.

ones, namely to $m = 0$ and $v = 1$. We computed the temporal means and variances for each of the $N = 5000$ simulated trajectories and we can see their distribution in Figure 7. Their respective means are $\overline{\mu} = 0.000686$ and $\overline{v} = 0.996004$ which are very close to 0 and 1, as we expected.

References

[AB06] C. Aliprantis, K. Border, *Infinite Dimensional Analysis: A Hitchhiker's Guide*, Springer, 3rd ed., 2006. doi: 10.1007/978-3-662-03004-2

[LM00] B. Laurent, P. Massart, Adaptive estimation of a quadratic functional by model selection, *The Annals of Statistics*, **28** (5), pp. 1302–1338, 2000. doi: 10.1214/aos/1015957395 Cited on p. 1

[PG18] V. Papadopoulos, D. Giovanis, *Stochastic finite element methods*, Springer, 2018. isbn: 9783319645278 Cited on p. 1, 4

Appendix

Exercise A

Importing packages:

```
import numpy as np
import pandas as pd
from scipy import optimize
import scipy.stats as stats
import matplotlib.pyplot as plt
from google.colab import files
from google.colab import drive
import os
```

The two equations which need to be solved so as to compute the ω_n's:

```
def eq_odd(x, a=2.5, b=2):
  return 1/b - x*np.tan(a*x)

def eq_even(x, a=2.5, b=2):
  return (1/b)*np.tan(a*x) + x
```

The corresponding eigenvalues λ_n's and the eigenfunctions ϕ_n's:

```
 def lam(w, a=2.5, b=2):
    return 2*b/(1 + w**2 * b**2)

def ell(w, a=2.5):
  return 1/np.sqrt(a - np.sin(2*w*a)/(2*w))

def c(w, a=2.5):
  return 1/np.sqrt(a + np.sin(2*w*a)/(2*w))

def phi_odd(w, t, a=2.5):
    return c(w, a)*np.cos(w*t)

def phi_even(w, t, a=2.5):
    return ell(w, a)*np.sin(w*t)
```

Computes the first n solutions of the previously defined equations:

```
def new_odd_omega(a=2.5, b=2, n=10):
  result = []
  count = 1
  for k in range(1, n):
    result.append(optimize.fsolve(eq_odd,
    (count-0.5)*np.pi/a - 0.1/count))
    result = [float(x) for x in result]
    count = count + 1
  return result

def new_even_omega(a=2.5, b=2, n=10):
  result = []
  count = 1
  for k in range(1, n):
    result.append(optimize.fsolve(eq_even,
    (count-0.5)*np.pi/a + 0.1/count))
    count = count + 1
    result = [float(x) for x in result]
  return result
```

Simulates N trajectories keeping M terms in the KL-expansion:

```
def simulate(a=2.5, b=2, points=200, N=5000, M=17):
  xx = np.linspace(-a, a, points)
  omega_odd = new_odd_omega(a, b, M)
  omega_even = new_even_omega(a, b, M)
  M_odd = len(omega_odd)
  M_even = len(omega_even)
  simulation_results = np.zeros((N, points))
  for j in range(N):
    ran_sample_odd = np.random.normal(size = M_odd)
    ran_sample_even = np.random.normal(size = M_even)
    summand = np.zeros((M_odd, points))
    for i in range(M_odd):
      summand[i, :] = np.sqrt(lam(omega_odd[i]))*
      phi_odd(omega_odd[i], xx)*ran_sample_odd[i]
    my_sum = np.sum(summand, axis = 0)
    summand = np.zeros((M_even, points))
    for k in range(M_even):
      summand[k, :] = np.sqrt(lam(omega_even[k]))*
      phi_even(omega_even[k], xx)*ran_sample_even[k]
    my_sum = my_sum + np.sum(summand, axis = 0)
    simulation_results[j, :] = my_sum
  return(simulation_results)
```

The theoretical ensemble variance:

```
def target_variance(a=2.5, b=2, points=200, M=17):
    xx = np.linspace(-a, a, points)
    omega_odd = new_odd_omega(a, b, M)
    omega_even = new_even_omega(a, b, M)
    M_odd = len(omega_odd)
    M_even = len(omega_even)
    summand = np.zeros((M_odd, len(xx)))
    for i in range(M_odd):
      summand[i, :] = lam(omega_odd[i]) *
      phi_odd(omega_odd[i], xx)**2
    my_sum = np.sum(summand, axis = 0)
    summand2 = np.zeros((M_even, len(xx)))
    for k in range(M_even):
      summand2[k, :] =  lam(omega_even[k]) *
      phi_even(omega_even[k], xx)**2
    my_sum = my_sum + np.sum(summand2, axis = 0)
    return my_sum
```

Simulating the trajectories and computing the ensemble variance.

```
bro80k = simulate(N=80000)
bro20k = simulate(N=20000)
bro5k = simulate(N=5000)

mu5k = np.mean(10 + 10*bro5k, axis = 0)
var5k = np.var(10 + 10*bro5k, axis = 0)

mu20k = np.mean(10 + 10*bro20k, axis = 0)
var20k = np.var(10 + 10*bro20k, axis = 0)


mu80k = np.mean(10 + 10*bro80k, axis = 0)
var80k = np.var(10 + 10*bro80k, axis = 0)

x = np.linspace(0, 5, 200)
plt.plot(x,  100*target_variance(M=17),
label='Theoretical for M=34')
plt.plot(x, var5k, label="Ensemble, N=5000, M=34",
alpha=0.8)
plt.plot(x, var20k, label="Ensemble, N=20000, M=34",
alpha=0.8)
plt.plot(x, var80k, label="Ensemble, N=80000, M=34",
alpha=0.8)
plt.legend()
plt.title('Theoretical vs Ensemble Variance for M=34')
plt.xlabel('x')
plt.ylabel('Variance')
```

Exercise B

```
def f1(w):
  # The first part of the power spectrum.
  return w-1

def f2(w):
  # The second part of the power spectrum.
  return 3-w

def simulation_B(w_u=3, lower=0, upper=10, points=200,
```

```
N=5000, M=100):
  # Simulates N trajectories from the process using M terms
  # omega_n in its spectral representation.

  M2 = int(M/2)
  Delta_w = w_u/M
  omega = np.linspace(0, 3*(M-1)/M, M)
  omega1 = []
  omega2 = []
  for i in range(len(omega)):
    if (omega[i]<2 and omega[i]>=1):
      omega1.append(omega[i])
    elif (omega[i]>=2 and omega[i]<=3):
      omega2.append(omega[i])
  M_omega1 = len(omega1)
  M_omega2 = len(omega2)
  pp = np.linspace(lower, upper, points)

  simulation_results = np.zeros((N, points))
  for j in range(N):
    unif1 = np.random.uniform(low=0.0,
    high=2*np.pi, size=M2)
    unif2 = np.random.uniform(low=0.0,
    high=2*np.pi, size=M2)
    summand1 = np.zeros((M2, points))
    for i in range(M_omega1):
      summand1[i,:] = np.sqrt(2) * np.sqrt(f1(omega1[i]) *
      Delta_w) * np.cos(omega1[i]*pp + unif1[i])
    my_sum1 = np.sum(summand1, axis = 0)
    summand2 = np.zeros((M2, points))
    for k in range(M_omega2):
      summand2[k,:] = np.sqrt(2) * np.sqrt(f2(omega2[k]) *
      Delta_w) * np.cos(omega2[k]*pp + unif2[k])
    my_sum2 = np.sum(summand2, axis = 0)
    my_sum = my_sum1 + my_sum2
    simulation_results[j, :] = my_sum
  return(simulation_results)
```