National Technical University of Athens
MSc - Data Science and Machine Learning

Data Driven Models
Assignment 1

MSc student
Vasileios Depastas
A.M: 03400131
vasileiosdepastas@mail.ntua.gr

June 2023

1 Karhunen-Loeve series expansion method

In this exercise, we examine the Karhunen-Loeve series expansion method with a practical application.

The Karhunen-Loeve (KL) series expansion method is a mathematical technique used to represent a stochastic process as a sum of orthogonal functions. The method is based on the spectral decomposition of the covariance function of the process and is particularly useful for stationary processes.

The KL expansion expresses the stochastic process as an infinite sum of eigenfunctions, each weighted by a corresponding coefficient. These eigenfunctions are obtained by solving an integral equation involving the covariance function of the process. The coefficients are determined by projecting the process onto each eigenfunction and taking its inner product.

The KL expansion has many applications in signal processing, data analysis, and machine learning. It can be used to reduce the dimensionality of data, extract features from signals, and model complex systems. The method is particularly useful for processes with strong correlations or dependencies between different time points.

Overall, the KL series expansion method provides a powerful tool for analyzing and modeling stochastic processes and has many practical applications in various fields.

We consider the following stochastic field:

$$E(x) = 10\left(1 + f(x)\right),$$

where $f(x)$ is a zero-mean stationary Gaussian field with unit variance and $x \in [0, 5]$ (m). The autocorrelation function for f is:

$$R_f(\tau) = e^{-\frac{|\tau|}{2}}.$$

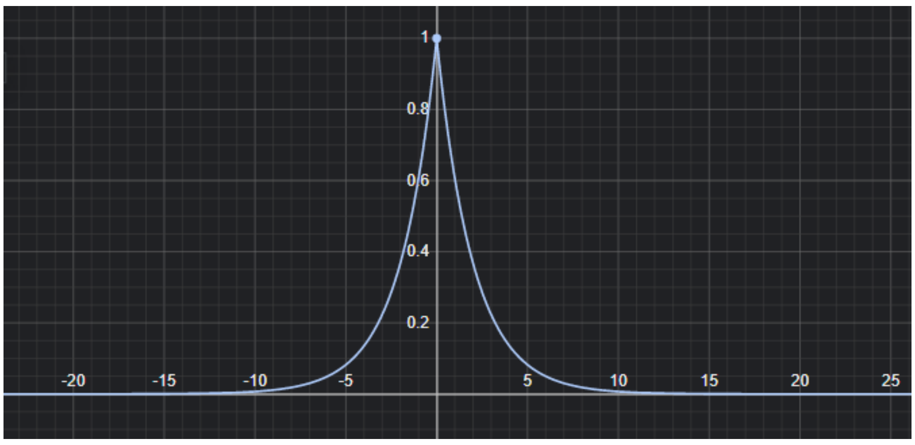

Figure 1: Autocorrelation function

For a zero-mean process, the autocorrelation and autocovariance coincide $R_f(t, s) = C_f(t, s)$, therefore the autocovariance function is of the following form / belongs to the following family of covariance functions:

$$C_f(t,s) = \sigma^2 e^{-\frac{|t-s|}{b}}.$$

The eigenvalues problem is then defined by the following equation:

$$\int_{D'} C_f(t,s)\phi_n(s)ds = \lambda_n \phi_n(t),$$

where C_f is the covariance function, which is by definition bounded, symmetric and non-negative definite and ϕ_n, λ_n are orthogonal deterministic eigenfunctions and eigenvalues of the covariance function.
First, we transform the domain $D = [0,5]$ to the symmetric $D' = [-a,a] = [-2.5, 2.5]$, where $a = 2.5$ and:

$$C_f(t, s-a) = R_f(t, s-a) = \sigma^2 e^{-\frac{|t-s+a|}{b}} = \sigma^2 e^{-\frac{|\tau+a|}{b}}.$$

For $a = 2.5$, $b = 2$ and $\sigma^2 = 1$, we get for the autocorrelation function:

$$R_f(t, s-2.5) = C_f(t, s-2.5) = e^{-\frac{|\tau+2.5|}{2}}$$

Introducing the variable $w^2 = \frac{2b-\lambda}{\lambda b^2}$, we get the following two equations:

$$\begin{cases} \frac{1}{b} - w_n \tan(w_n a) = 0 \\ w'_n + \frac{1}{b}\tan(w'_n a) = 0 \end{cases}$$

from which we can analytically derive its solutions w_n and w'_n with solutions lying respectively in the domains:

$$\begin{cases} \left[\frac{(n-1)\pi}{a}, \frac{(n-0.5)\pi}{a}\right] \\ \\ \left[\frac{(n-0.5)\pi}{a}, \frac{n\pi}{a}\right] \end{cases}$$

The equations lead to two families of eigenvalues λ_n, λ'_n and eigenfunctions $\phi_n(x)$, $\phi'_n(x)$:

$$\begin{cases} \lambda_n = \frac{2b}{1+w_n^2 b^2} \\ \\ \lambda'_n = \frac{2b}{1+{w'_n}^2 b^2} \end{cases}$$

$$\begin{cases} \phi_n(x) = \phi_n(x, w_n) = \frac{\cos w_n x}{\sqrt{a + \frac{\sin 2w_n a}{2w_n}}} \\ \\ \phi'_n(x) = \phi'_n(x, w'_n) = \frac{\sin w'_n x}{\sqrt{a - \frac{\sin 2w'_n a}{2w'_n}}} \end{cases}$$

The K-L expansion can then be formulated as:

$$f(x) = \sum_{n=1}^{\infty} \left(\sqrt{\lambda_n}\phi_n(x-a)\xi_n + \sqrt{\lambda'_n}\phi'_n(x-a)\xi'_n \right),$$

where $\xi_n = \xi_n(\theta)$ and $\xi'_n = \xi'_n(\theta) \sim N(0,1)$.

We are going to keep the first $k \in \mathbf{N}$ terms of the series expansion, i.e.:

$$f_k(x) \approx f(x) = \sum_{n=1}^{k} \left(\sqrt{\lambda_n}\phi_n(x-a)\xi_n + \sqrt{\lambda'_n}\phi'_n(x-a)\xi'_n \right),$$

where we keep the first k eigenvalues $\lambda_1, \lambda_2, ..., \lambda_k$ in descending order. As a result, we have an approximation error and the explained variance will be:

$$EV = \frac{\int_D Var\{f_k(x)\}dx}{\int_D Var\{f(x)\}dx} = \frac{\sum_{n=1}^{k}(\lambda_n + \lambda'_n)}{\int_0^5 1dx} = \frac{1}{5}\sum_{n=1}^{k}(\lambda_n + \lambda'_n),$$

where we utilized the fact that $f(x)$ has unit variance, i.e. $Var\{f(x)\} = 1$ and that ϕ_n and ϕ'_n are orthogonal, as well as $Var\{\xi_n\} = Var\{\xi'_n\} = 1$.
We will now search for the minimum number of terms k for which the expected variance is larger than a threshold. We set this threshold to 99% or 0.99. After simulating and deriving solutions w_n, w'_n of the equations and subsequently calculating λ_n, λ'_n in descending order, we can keep adding terms until the explained variance score goes over the set threshold of 0.99. We concluded, that the number of terms to keep is $k = 26$ based on the simulation results. We can see this result in a graphical way below:

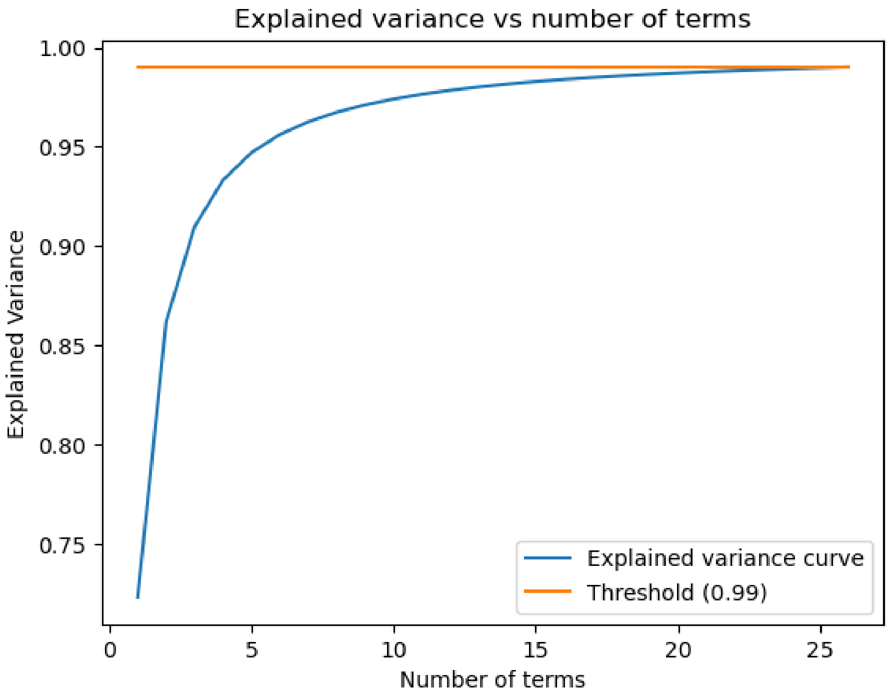

Figure 2: Explained variance versus number of terms n

The code used for the simulation is below;

```
% imports and helper functions

import numpy as np
from scipy.optimize import fsolve
import matplotlib.pyplot as plt
import plotly.express as px
import plotly.graph_objects as go

def omega_equation_1(w: float, a: float = 2.5, b: float = 2):
    """
    Return the left part of the equation to solve for w:
    1/b - w * tan(w * a) = 0
    """
    return 1/b - w * np.tan(w * a)


def omega_equation_2(w_toned: float, a: float = 2.5, b: float = 2):
    """
    Return the left part of the equation to solve for w':
    w' + (1/b) tan(w' * a) = 0
    """
    return w_toned + (1/b) * np.tan(w_toned * a)


def lambdas_values_calculation(w: float, a: float = 2.5, b: float = 2):
    """
    Calculate lambda from w:
    l = 2b / (1 + w^2 * b^2)
    """
    return 2.0*b / (1.0 + (w**2) * (b**2))


def omega_solutions(n: int, a: float = 2.5, b: float = 2):
    """
    Solve equations for omega (w) and omega tonned (w') and return their values.
    """
    w = fsolve(omega_equation_1, (n-1) * np.pi/a + 1e-4).item()
    w_toned = fsolve(omega_equation_2, (n-0.5) * np.pi/a + 1e-4).item()
    return w, w_toned


def explained_variance(lambda_values: np.array, lambda_toned_values: np.array) -> float:
    """
    Return the calculated explained variance.
    """
    return (1/5) * np.sum(lambda_values + lambda_toned_values)


def eigenfunction(x: float, w: np.array, a: float = 2.5):
    """
    Return the _n(x) value of the eigenfunction.
    """
    phi = np.cos(w * x) / np.sqrt(a + np.sin(w*2*a)/(w*2))
    return phi


def eigenfunction_toned(x: float, w_toned: float, a: float = 2.5):
    """
    Return the _n'(x) value of the toned eigenfunction.
    """
    phi_toned = np.sin(w_toned * x) / (np.sqrt(a - np.sin(w_toned*2*a)/(w_toned*2)))
    return phi_toned


def realizations(k:int, w_values, w_toned_values, sqrt_lambda_values,
                 sqrt_lambda_toned_values, a=2.5, b=2, N=5000):
    """
    Generate N realizations of E(x).
    """
    number_of_x_points = 1000
    realizations_array = np.zeros((N,number_of_x_points))
    Xs = np.linspace(-a, a, number_of_x_points)
    w_array, w_toned_array = np.array(w_values), np.array(w_toned_values)
    term1 = np.zeros((k, number_of_x_points))
    term2= np.zeros((k, number_of_x_points))

    # generate all k terms of the series expansion for each x point of the Xs range
    for idx in range(k):
        term1[idx,:] = sqrt_lambda_values[idx]*eigenfunction(Xs, w_array[idx], a=a)
        term2[idx,:] = sqrt_lambda_toned_values[idx]*eigenfunction_toned(Xs,
                                                                 w_toned_array[idx], a=a)

    # generate N realizations
    for idx in range(N):
        # generate k xis
        xis = np.random.standard_normal(k)
        xis_toned = np.random.standard_normal(k)
        first_terms = np.zeros((k, number_of_x_points))
        second_terms = np.zeros((k, number_of_x_points))
        for idx2 in range(k):
            first_terms[idx2, :] = term1[idx2, :] * xis[idx2]
            second_terms[idx2, :] = term2[idx2,:] * xis_toned[idx2]
        realizations_array[idx,:] = 10 * (1 + first_terms.sum(0) + second_terms.sum(0))

    return realizations_array


# code for terms calculation for Explained Variance to surpass the threshold value

lambda_values, lambda_toned_values = [], []
w_values, w_toned_values = [], []
explained_variance_values = []
n = 0
explained_variance_value = 0
EXPLAINED_VARIANCE_THRESHOLD = 0.99

while (n <= 10000) and (explained_variance_value < EXPLAINED_VARIANCE_THRESHOLD):
    n += 1
    w, w_toned = omega_solutions(n)
    w_values.append(w)
    w_toned_values.append(w_toned)

    # w_values and w_toned_values verified to be correct

    lambda_value, lambda_toned_value = lambdas_values_calculation(w), lambdas_values_calculation(w_toned)
    lambda_values.append(lambda_value)
    lambda_toned_values.append(lambda_toned_value)

    # lambda_values and lambda_toned_values verified to be correct

    explained_variance_value = explained_variance(np.array(lambda_values), np.array(lambda_toned_values))
    explained_variance_values.append(explained_variance_value)

print(f'Number of terms kept: {n}')   # this results in printing Number of terms kept: 26

# plot code of Explained Variance vs number of terms

fig, ax = plt.subplots()
ax.plot(np.arange(1, n+1), explained_variance_values, label='Explained variance curve')
ax.plot(np.arange(1, n+1), 0.99 * np.ones(n), label=f'Threshold ({EXPLAINED_VARIANCE_THRESHOLD})')
ax.set_xlabel('Number of terms')
ax.set_ylabel('Explained Variance')
plt.title('Explained variance vs number of terms')
plt.legend()
plt.show()
```

Now, we generate $N = 5000$ realizations of the field $E(x)$, based on the previous result of 26 terms kept in the KL series expansion of $f(x)$.

```
k = n  # number of terms to keep

E_realizations = realizations(k,
                              w_values,
                              w_toned_values,
                             sqrt_lambda_values,
                             sqrt_lambda_toned_values
                           )

# plot the first 5 realizations
fig, ax = plt.subplots()
Xs = np.linspace(0, 5, 1000)

for realization_idx in range(5):
    ax.plot(Xs, E_realizations[realization_idx])

# add true average line
ax.axhline(10,
           linestyle='--',
           label='True Average',
           color='blue'
    )

# add estimated average line
estimated_average = np.mean(E_realizations, 0)
ax.plot(Xs,
        estimated_average,
        label='Estimated Average',
        alpha=0.6,
        color='black'
    )

ax.set_title(f'(x) K-L series expansion. First 5 realizations.')
ax.set_xlabel('x')
ax.set_ylabel(r'$E(x)$')
ax.legend()
plt.show()
```

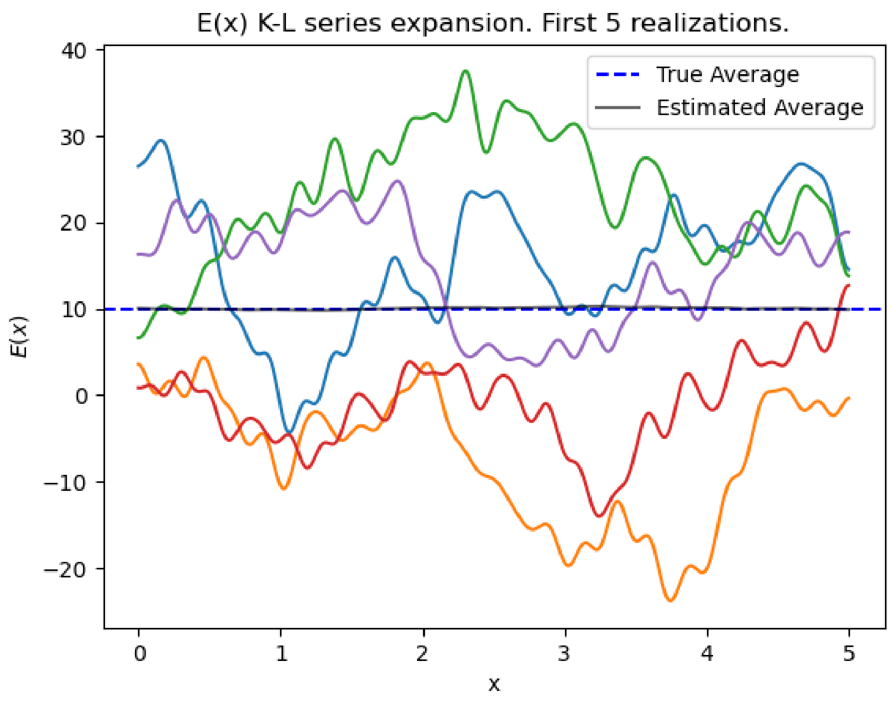

Figure 3: First 5 E(x) realizations

As we saw on the above plot, the ensemble average is 10 and this is pretty accurately found by the estimated average, coinciding more and more as we increase the number of realizations. In fact, we can show that:

$$E[E_k(x)] = 10.$$

Next, we calculate and plot the ensemble variance from these realizations. We can show that:

$$Var\left[E_k(x)\right] = 100 \sum_{n=1}^{k} \left(\lambda_n \phi_n^2(x-a) + \lambda'_n {\phi'_n}^2(x-a)\right)$$

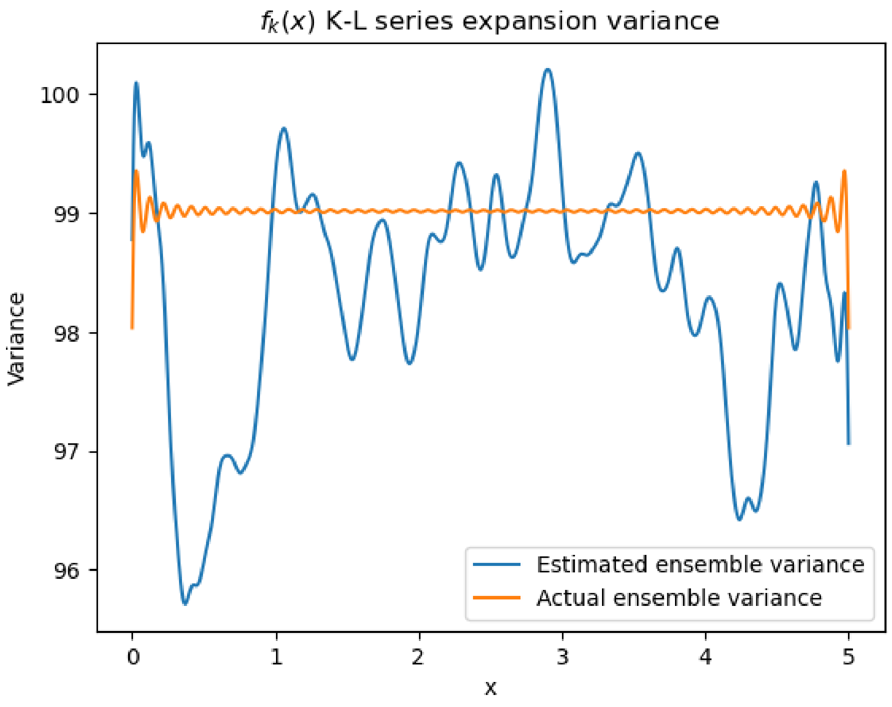

Figure 4: Variance of ensemble

By increasing the number of realizations, the estimated ensemble variance converges to the actual one as seen in the previous figure. The code used for the variance calculation and plotting is below;

```
number_of_x_points = 1000
a = 2.5
Xs = np.linspace(-a, a, number_of_x_points)

w_array, w_toned_array = np.array(w_values), np.array(w_toned_values)

term1 = np.zeros((k, number_of_x_points))
term2= np.zeros((k, number_of_x_points))

for idx in range(k):
    term1[idx,:] = lambda_values[idx] * (eigenfunction(Xs, w_array[idx], a=a))**2
    term2[idx,:] = lambda_toned_values[idx] * (eigenfunction_toned(Xs, w_toned_array[idx], a=a))**2
```

```
true_variance = 100.0*(term1.sum(0) + term2.sum(0))

# plot

fig, ax = plt.subplots()
ax.set_title('$f_k(x)$ K-L series expansion variance')
ax.plot(np.linspace(0, 5, number_of_x_points), np.var(E_realizations, axis=0), label='Estimated ensemble variance')
ax.plot(np.linspace(0, 5, number_of_x_points), true_variance, label='Actual ensemble variance')
ax.set_xlabel('x')
ax.set_ylabel('Variance')
ax.legend()
plt.show()
```

2 The Spectral Representation method

The Gaussian process $X(t)$, $t \in [0, 10]$(sec) has zero mean and its one-sided spectrum is defined as:

$$G(w) = \begin{cases} w-1, & \text{if } 1 \leq w \leq 2 \\ 3-w, & \text{if } 2 < w \leq 3 \\ 0, & \text{otherwise.} \end{cases}$$

$X(t)$ can be expanded in a series of terms by utilizing the spectral representation method as below:

$$X(t) = \sqrt{2} \sum_{n=0}^{\infty} \left[A_n \cos\left(w_n t + \Phi_n\right)\right].$$

We can get the truncated series expansion with k terms by keeping its first k ones, i.e.:

$$X_k(t) = \sqrt{2} \sum_{n=0}^{k-1} \left[A_n \cos\left(w_n t + \Phi_n\right)\right] \approx X(t),$$

where:

$$\Phi_n \sim U(0, 2\pi),$$

are independent phase angles uniformly distributed in the range $[0, 2\pi]$,

$$A_n = \sqrt{2G(w_n)\Delta w}, \text{ for } n = 0, 1, 2, ..., k-1$$

$$w_n - w_0 = n \cdot \Delta w, \text{ or } \Delta w = \frac{w_n - w_0}{n},$$

with Δw being the step with which w increases and $w_n \in [1, 3]$. Thus $w_0 = 1$ and for $n = k-1$, $w_{k-1} = 3$ and we can calculate:

$$\Delta w = \frac{w_{k-1} - w_0}{k-1} = \frac{3-1}{k-1} = \frac{2}{k-1}.$$

Then, for any $n = 0, 1, 2, ..., k-1$:

$$w_n = w_0 + n \cdot \Delta w = 1 + n \cdot \frac{2}{k-1} = \frac{2n + k - 1}{k-1}.$$

Also

$$A_0 = \sqrt{2 \cdot G(w_0)\Delta w} = \sqrt{2 \cdot G(1)\Delta w} = 0,$$

since $G(1) = 0$ from its definition. We assume $S_x(w)$ to be the two-sided power-spectral function of $X(t)$, $G(w)$, its one-sided power-spectral function as defined previously, and $R_x(t_i, t_j)$ its autocorrelation function. The power spectral density function is even, i.e.:

$$2\int_0^\infty G(w)dw = 2\int_0^\infty S_x(w)dw = \int_{-\infty}^\infty S_x(w)dw.$$

The autocorrelation function can be calculated as:

$$\begin{aligned}
R_x(\tau) &= \int_{-\infty}^\infty S_x(w)\cos(w\tau)dw \\
&= 2\int_0^\infty G(w)\cos(w\tau)dw \\
&= 2\left[\int_1^2 (w-1)\cos(w\tau)dw + \int_2^3 (3-w)\cos(w\tau)dw\right] \\
&= \frac{8}{\tau^2} sin^2\left(\frac{\tau}{2}\right)\cos(2\tau).
\end{aligned}$$

For $\tau = 0$:

$$\begin{aligned}
R_x(0) &= 2\left[\int_1^2 (w-1)dw + \int_2^3 (3-w)dw\right] \\
&= 2\left(\left[\frac{w^2}{2} - w\right]_1^2 + \left[3w - \frac{w^2}{2}\right]_2^3\right) \\
&= 2\left(2 - 2 - \frac{1}{2} + 1 + 9 - \frac{9}{2} - 6 + 2\right) \\
&= 2\,(6-5) \\
&= 2,
\end{aligned}$$

and we've got that $R_x(t_1, t_2) = R_x(t_2 - t_1) = \mathbf{E}[X(t_1)X(t_2)]$, for every t_1, t_2, where we usually write $R_x(\tau) = \mathbf{E}[X(t+\tau)X(t)]$, from which we get that $R_x(0) = \mathbf{E}[X(t)X(t)] = \mathbf{E}[X^2(t)]$. However, we also have that $\mathbf{Var}[X(t)] = \mathbf{E}\left[(X(t) - \mathbf{E}[X(t)])^2\right] = \mathbf{E}[X^2(t)] - \mathbf{E}^2[X(t)] = \mathbf{E}[X^2(t)]$, and finally then $\mathbf{Var}[X(t)] = R_x(0) = 2$.

For $R_X(\tau)$, we've got additionally:

$$|R_x(\tau)| = \frac{8}{\tau^2}|sin^2\left(\frac{\tau}{2}\right)\cos(2\tau)| \leq \frac{8}{\tau^2}\cdot 1 = \frac{8}{\tau^2},$$

therefore

$$\lim_{\tau\to\infty} R_x(\tau) \to 0,$$

and the stochastic process $X(t)$ is ergodic in its autocorrelation. Since it is zero-mean, and thus its mean is constant then it comes that it is also ergodic in the mean.

For the Gaussian process $X(t)$, we have then found that:

$$\mathbf{E}[X(t)] = 0 \text{ (zero-mean)}$$

$$\mathbf{Var}[X(t)] = 2.$$

For the truncated series, we would expect the mean and variance to approximate these values.

$$\begin{aligned}\mathbf{E}[X_k(t)] &= \mathbf{E}\left[\sqrt{2}\sum_{n=0}^{k-1}\left[A_n\cos\left(w_nt+\Phi_n\right)\right]\right]\\ &= \sqrt{2}\sum_{n=0}^{k-1}\left(A_n\mathbf{E}\left[\cos\left(w_nt+\Phi_n\right)\right]\right),\end{aligned}$$

where:

$$\mathbf{E}\left[\cos\left(w_nt+\Phi_n\right)\right] = \int_0^{2\pi}\frac{1}{2\pi}\cos(w_nt+\Phi_n)d\Phi_n = \frac{1}{2\pi}\left[sin(w_nt+\Phi_n)\right]_0^{2\pi} = 0.$$

Therefore, we get:

$$\mathbf{E}[X_k(t)] = 0.$$

$$\begin{aligned}\mathbf{Var}[X_k(t)] &= \mathbf{Var}\left[\sqrt{2}\sum_{n=0}^{k-1}\left[A_n\cos\left(w_nt+\Phi_n\right)\right]\right]\\ &= 2\sum_{n=0}^{k-1}\left(A_n^2\mathbf{Var}\left[\cos\left(w_nt+\Phi_n\right)\right]\right),\end{aligned}$$

since the Φ_n random variables are independent between each other for different n values. Now, we calculate:

$$\begin{aligned}\mathbf{Var}\left[\cos\left(w_nt+\Phi_n\right)\right] &= \mathbf{E}\left[\cos^2\left(w_nt+\Phi_n\right)\right] - \mathbf{E}^2\left[\cos\left(w_nt+\Phi_n\right)\right]\\ &= \mathbf{E}\left[\cos^2\left(w_nt+\Phi_n\right)\right] - 0\\ &= \int_0^{2\pi}\frac{1}{2\pi}\cos^2(w_nt+\Phi_n)d\Phi_n\\ &= \frac{1}{2\pi}\left[\frac{\Phi_n}{2}+\frac{sin(2w_nt+2\Phi_n)}{4}\right]_0^{2\pi}\\ &= \frac{1}{2}\end{aligned}$$

and thus:

$$\begin{aligned}\mathbf{Var}[X_k(t)] &= 2\sum_{n=0}^{k-1}\left(A_n^2\frac{1}{2}\right)\\ &= \sum_{n=0}^{k-1}\left(A_n^2\right)\\ &= \sum_{n=0}^{k-1}\left(2G(w_n)\Delta w\right).\end{aligned}$$

When $k \to \infty$, then:

$$
\begin{aligned}
\mathbf{Var}[X_k(t)] &\to \int 2G(w)dw \\
&= 2\int G(w)dw \\
&= 2 \cdot 1 \\
&= 2.
\end{aligned}
$$

As a result, if we get sufficient large number of terms, we expect the truncated series expansion variance to approximate the true variance value of 2.

Below we have selected to keep $k = 20$ terms in the series and get 5000 realizations of the process $X(t)$. Then, we present the first 5 realizations based on the power-spectral representation method and the true average value as well as the ensemble's mean average value that we can see approximates 0.

```
% imports and helper functions

def G(w):
    """
    One-sided spectrum definition G(w) of the process X(t).
    """
    if 1<=w<=2:
        return w-1
    elif 2<w<=3:
        return 3-w
    else:
        return 0

np.random.seed(42)  # set random state for reproducability

N = 5000  # number of realizations
M = 20  # number of terms to keep in the series expansion

number_of_t_points = 1000  # number of time points to take
t_range = np.linspace(0, 10, number_of_t_points)  # time range in [0, 10]
w_range = np.linspace(1, 3, M)

dw = (w_range[-1] - w_range[0]) / (M-1)  # w

G_vectorized = np.vectorize(G)  # vectorize function so that
# it can be applied to numpy arrays and not single numbers

A_n = np.sqrt(2 * G_vectorized(w_range) * dw)  # A_n for n=0,1,2,...,M-1
A_n = np.reshape(A_n, (-1, 1))
"""
Will create the matrix B_n = w_n * t :
[
 [w0*t0, w0*t1, ..., w0*t999],
 [w1*t0, w1*t1, ..., w1*t999],
 ...
 [w_M-1*t0, w_M-1*t1, ...., w_M-1*t999]
]
"""
B_n = np.outer(w_range, t_range)

# realizations = []
realizations = np.array([])
for realization_idx in range(N):
    # then we add _n to B_n to form C_n = B_n + _n = w_n * t + _n
    Phi_n = np.random.uniform(0, 2 * np.pi, size = (M, 1))
    C_n = B_n + Phi_n

    # then we for the cosine D_n = cos(C_n) = cos(w_n * t + _n)
    D_n = np.cos(C_n)
    # then we form the term in the series E_n = A_n * cos(w_n * t + _n)
    E_n = A_n * D_n

    X_t = np.sqrt(2) * np.sum(E_n, axis=0)
    # realizations.append(X_t)
    realizations = np.append(realizations, X_t)

realizations = realizations.reshape(N, -1)

fig, ax = plt.subplots()

Ts = np.linspace(0, 10, number_of_t_points)

for realization_idx in range(5):
    ax.plot(Ts, realizations[realization_idx])

# add true average line
ax.axhline(0,
           linestyle='--',
           label='True Average',
           color='blue'
)

# add estimated average line
estimated_average = np.mean(realizations, 0)

ax.plot(Ts,
        estimated_average,
        label='Estimated Ensemble Average',
        color='black'
)

ax.set_title(f'X(t) Gaussian process spectral representation. First 5 realizations.')
ax.set_xlabel('t')
ax.set_ylabel(r'$X^P(t)$')
ax.legend()
plt.show()
```

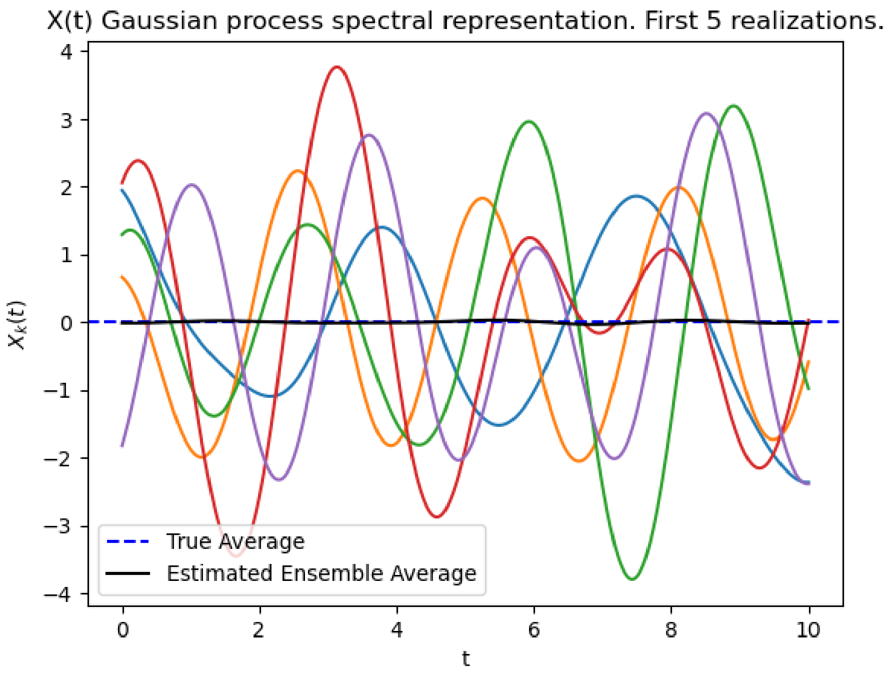

Figure 5: First 5 realizations and true mean and ensemble mean for power spectral representation

Next, we calculate the estimated variance of the ensemble ($k = 20$ terms series expansion for the 5000 realizations ensemble) and the true variance of the ensemble and compare with the theoretical value.

```
true_variance = 2 * dw * np.sum(G_vectorized(w_range))
print(f'True variance for the series terms selected: {true_variance}')

estimated_variance = np.var(realizations)
print(f'Estimated variance from {N} realizations: {estimated_variance}')

# printed results below:
# True variance for the series terms selected: 1.9944598337950137
# Estimated variance from 5000 realizations: 1.9777644794718492

fig, ax = plt.subplots()

Ts = np.linspace(0, 10, number_of_t_points)

# add estimated variance line
estimated_variance_2 = np.var(realizations, 0)

ax.plot(Ts,
        estimated_variance_2,
        label='Estimated Ensemble Variance',
        color='black'
)


# add true ensemble variance line
ax.plot(Ts,
        np.array([true_variance] * number_of_t_points),
        label='True Ensemble Variance',
        color='blue'
)

# add true infinite series variance line
ax.plot(Ts,
        np.array([2] * number_of_t_points),
        label='True Infinite Series Expansion Variance',
        color='red'
)


ax.set_title(f'Spectral representation method variance.')
ax.set_xlabel('t')
ax.set_ylabel(r'$X^P(t)$')
ax.legend()
plt.show()
```

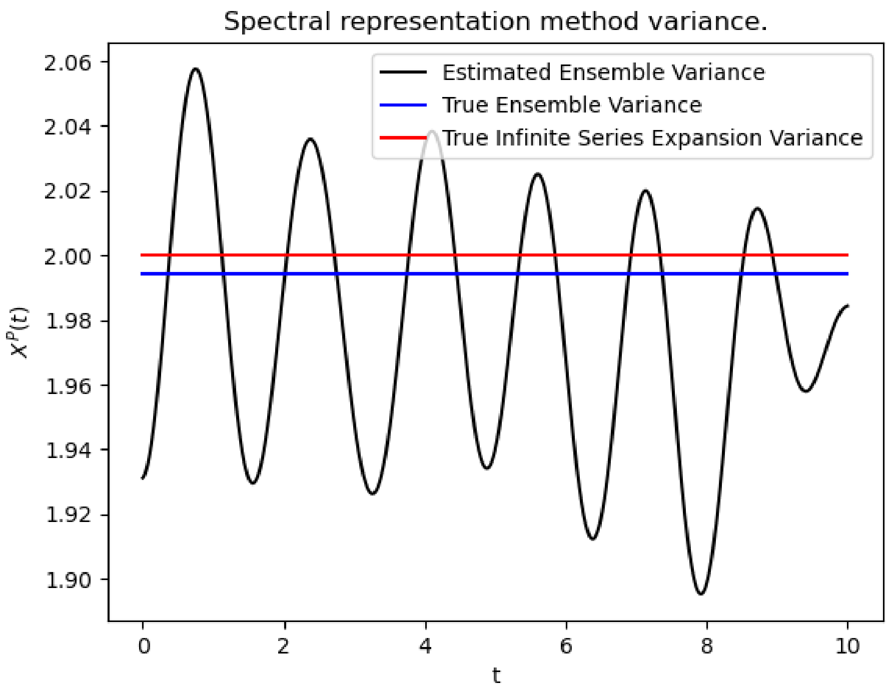

Figure 6: Variance of realizations ensemble.n

We can see that the true ensemble variance approximates very well the theoretical variance value of 2. However, the estimated one fluctuates but approximates the true variance. As we increase the number of realizations taken, we can expect the true and estimated ensemble variance values to approximate the true variance value better and converge there.

Finally, we select a single realization (single index out of the 5000 generated realizations of the process $X(t)$) and explore its mean and variance.

```
realization_selected_idx = 0

realization = realizations[realization_selected_idx]

realization_mean = np.mean(realization)
```

```python
realization_variance = np.var(realization)

print(f'Realization with idx={realization_selected_idx}:')
print(f'Mean={realization_mean}')
print(f'Variance={realization_variance}')

# visualization
fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(12, 8))
fig.suptitle(f'Realization with index={realization_selected_idx} - spectral representation method.', fontsize=16)

# first plot - average
ax[0].plot(Ts,
        np.array([realization_mean] * number_of_t_points),
        label="Estimated realization's Mean",
        color='black'
)

# add true infinite series mean line
ax[0].plot(Ts,
        np.array([0] * number_of_t_points),
        label='True Infinite Series Expansion Mean',
        color='red'
)


ax[0].set_title(f'Mean')
ax[0].set_xlabel('t')
ax[0].set_ylabel(r'$X_k(t)$')
ax[0].legend()

# second plot - variance
ax[1].plot(Ts,
        np.array([realization_variance] * number_of_t_points),
        label="Estimated realization's Variance",
        color='black'
)

# add true infinite series variance line
ax[1].plot(Ts,
        np.array([2] * number_of_t_points),
        label='True Infinite Series Expansion Variance',
        color='red'
)


ax[1].set_title(f'Variance')
ax[1].set_xlabel('t')
ax[1].set_ylabel(r'$X_k(t)$')
ax[1].legend()


plt.show()
```

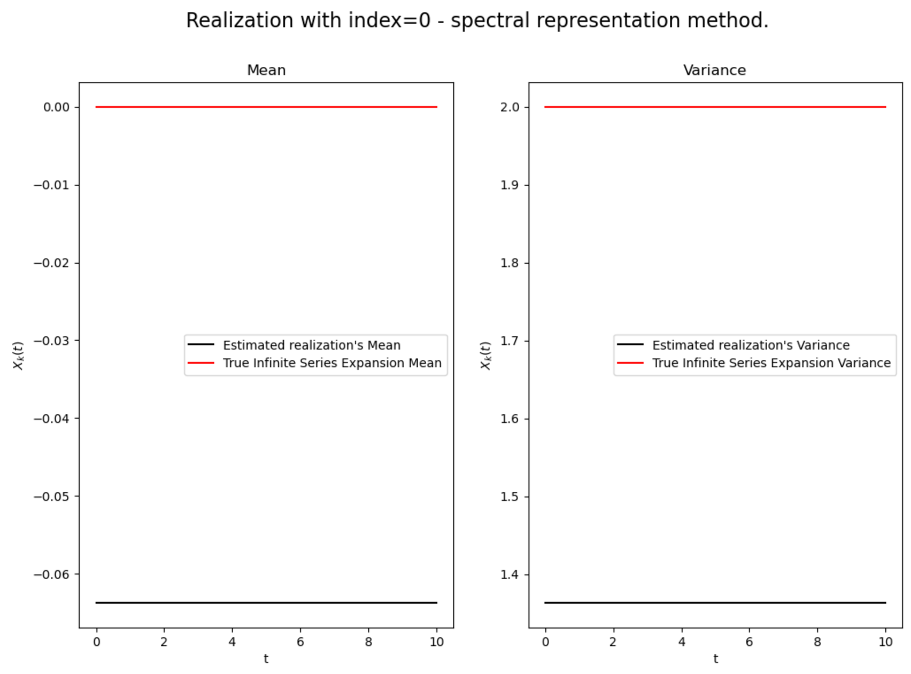

Figure 7: Mean and variance of single realization

We can see that the realization's temporal mean is very close to the true mean. This is a result we did expect since the process is ergodic in the mean, and hence we nly need a realization to find the mean of the process $X(t)$ as they should coincide. However, this is not true for the variance and this is why the realization's variance deviates a lot from the true variance.