AM5340
Stochastic Processes in Mechanics
Project

Archish S
me20b032@smail.iitm.ac.in

Indian Institute of Technology, Madras

May 22, 2023

Contents

1 SDoF Oscillators

Consider the response of two components in an automobile, both modeled as oscillators, subjected to support motion acceleration modeled as mean zero stationary process with auto power spectral density,

$$S_{FF}(\omega) = \frac{6}{400 + \omega^2} \tag{1}$$

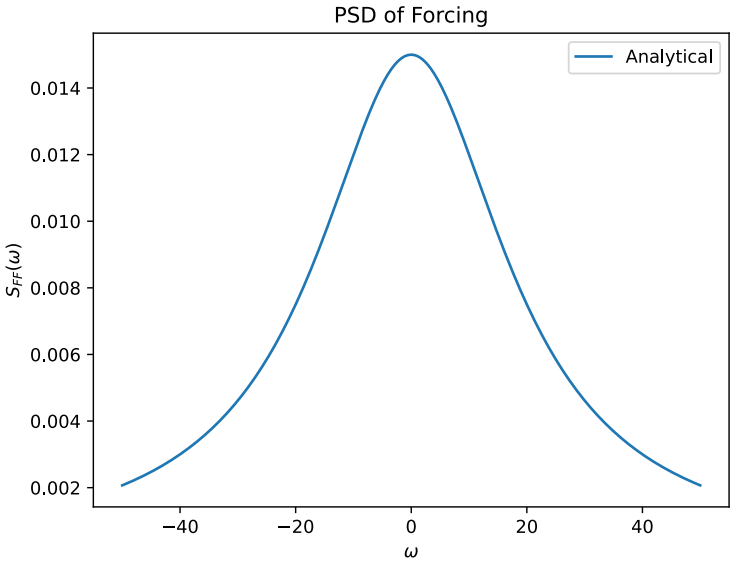


Figure 1.1: PSD $S_{FF}(\omega)$ of Forcing

Each oscillator is modeled as a SDoF system,

$$\ddot{x}_j(t) + 2\zeta_j\omega_j\dot{x}_j + \omega_j^2 x_j(t) = f(t) \tag{2}$$

with $x_j(t)$ representing the displacement of the j^{th} component. $\omega_1 = 10$ rad/s, $\omega_2 = 15$ rad/s, $\zeta_1 = 0.01$ and $\zeta_1 = 0.005$.

The auto correlation can be computed from the auto power spectral density as,

$$\begin{aligned} R_{FF}(\tau) &= \frac{1}{2\pi}\int_{-\infty}^{\infty} S_{FF}(\omega)e^{i\omega\tau}d\omega \\ &= \frac{1}{2\pi}\int_{-\infty}^{\infty} \frac{6}{400 + \omega^2}e^{i\omega\tau}d\omega \end{aligned}$$

Computing $R_{FF}(\tau)$ at $\tau = 0$,

$$R_{FF}(0) = \sigma_F^2 = \frac{1}{2\pi}\int_{-\infty}^{\infty} \frac{6}{400 + \omega^2}d\omega$$

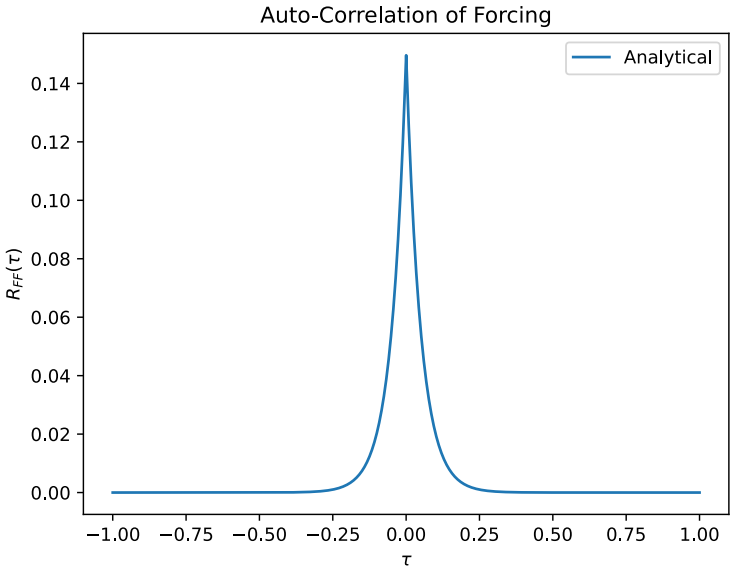


Figure 1.2: Auto-Correlation $R_{FF}(\tau)$ of Forcing

Assuming F to be a second-order weak-sense stationary process, the pdf of F can be expressed for an ergodic process as,

$$p_{_F}(f) = \frac{1}{\sqrt{2\pi}\sigma_{_F}} e^{\frac{-1}{2\sigma_F^2}(f-\mu_F)^2}$$

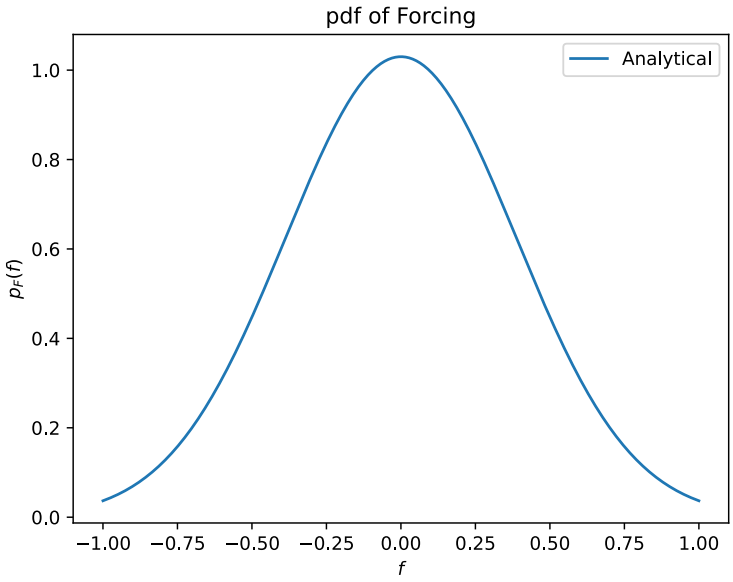


Figure 1.3: pdf $p_{_F}(f)$ of Forcing

1.1 Part A

Let b be the static clearance between the two oscillators. This gives the clearance between the two components during operating conditions as $Y = b - X_1 + X_2$. Analytically, find the stationary pdf for Y, assuming that X_1 and X_2 are independent.

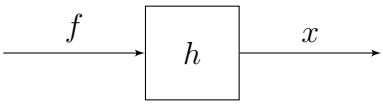
1.1.1 Method

$$\begin{aligned}\text{PDF}\left[Y \leq y\right] &= \text{PDF}\left[b - X_1 + X_2 \leq y\right] \\ &= \text{PDF}\left[X_2 \leq y - b + X_1\right]\end{aligned}$$

$$\begin{aligned}P_Y(y) &= P_{X_2}(y - b + X_1) \\ &= \int_{-\infty}^{\infty}\int_{-\infty}^{y-b+X_1} p_{X_1X_2}(x_1, x_2)dx_1dx_2 \\ &= \int_{-\infty}^{\infty}\int_{-\infty}^{y-b+X_1} p_{X_1}(x_1)p_{X_2}(x_2)dx_2dx_1\end{aligned}$$

$$p_Y(y) = \int_{-\infty}^{\infty} p_{X_1}(x_1)p_{X_2}(y - b + x_1)dx_1$$

Since the governing equation of $x(t)$ is Linear and Time Independent, according to [1], we have



$$S_{XX}(\omega) = S_{FF}(\omega)|H(\omega)|^2$$

Since F is Gaussian second-order weak-sense stationary (and ergodic), X is characterized by

$$p_X(x) = \frac{1}{\sqrt{2\pi}\sigma_X}e^{\frac{-1}{2\sigma_X^2}(x-\mu_X)^2}$$

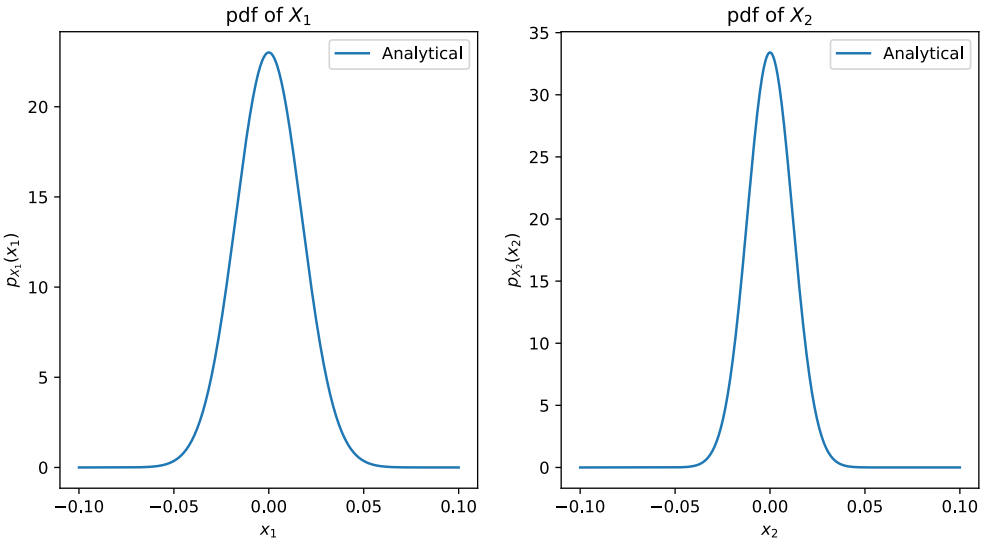


Figure 1.4: pdf $p_x(x)$ of Responses

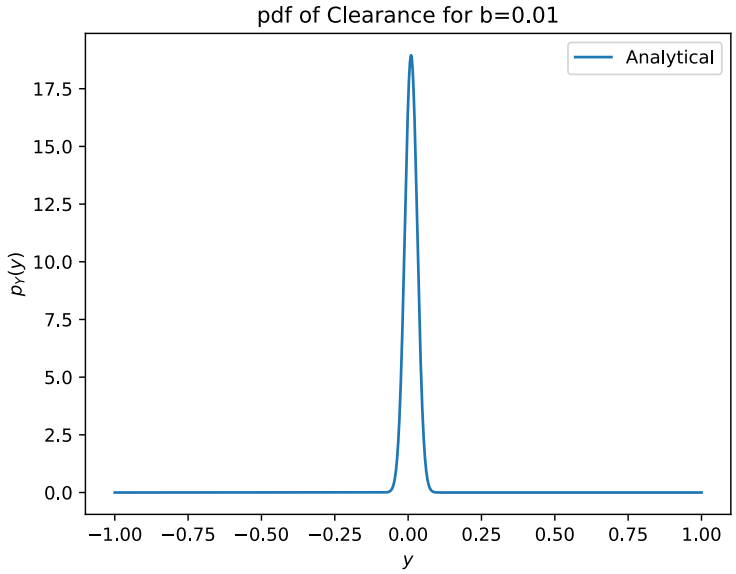


Figure 1.5: pdf of Clearance $p_Y(y)$ for $b = 0.01$

1.2 Part B

Next, simulate the random forcing and show that the simulated forcings match the target pdfs and the target PSDs.

1.2.1 Method

For $-\omega_0 \leq \omega < \omega_0$, let $\omega[i] = -\omega_0 + \frac{2\omega_0}{N} i$ for $i = 0, \ldots, N-1$; We compute R_{FF} according to [1] as,

$$R_{FF} = \text{IFFT}\{S_{FF}\}$$

Consider, a white-noise process $z \sim \mathcal{N}(0, I_{N\times N})$. The desired process F can be obtained as

$$f = [D]\, z$$

where $D^T D = C$ is the covariance matrix. The covariance matrix can be generated as

$$[C] = \begin{bmatrix} R_{FF}[0] & R_{FF}[1] & \ldots & R_{FF}[N-1] \\ R_{FF}[1] & R_{FF}[0] & \ldots & R_{FF}[N-2] \\ \vdots & \vdots & \ddots & \vdots \\ R_{FF}[N-1] & R_{FF}[N-2] & \ldots & R_{FF}[0] \end{bmatrix}$$

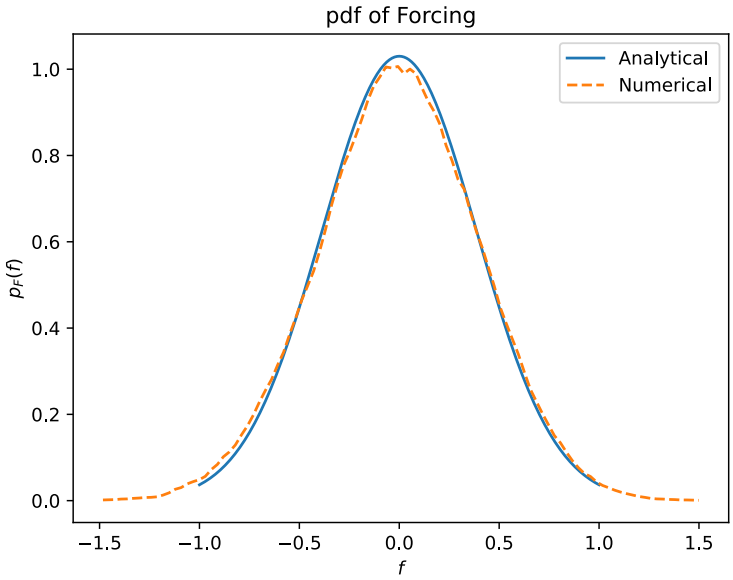


Figure 1.6: pdf $p_{_F}(f)$ of Forcing

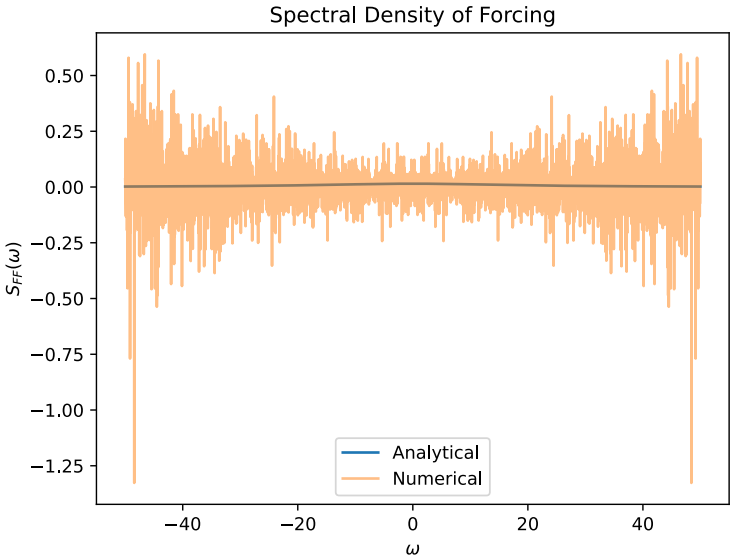


Figure 1.7: PSD $S_{FF}(\omega)$ of Forcing

1.3 Part C

Use the simulated forcings as input and numerically integrate the equations. Plot sample time histories of $x_1(t)$ and $x_2(t)$. Show that the probabilistic characterizers match the analytically derived pdfs and PSDs.

1.3.1 Method

Consider the governing equation [Eq. 2] re-written in the state space form as

$$X = \begin{bmatrix} x(t) \\ \dot{x}(t) \end{bmatrix}$$

$$\dot{X} = \begin{bmatrix} \dot{x}(t) \\ \ddot{x}(t) \end{bmatrix} = \begin{bmatrix} X[1] \\ f(t) - 2\zeta_j\omega_j X[1] - \omega_j^2 X[0] \end{bmatrix}$$

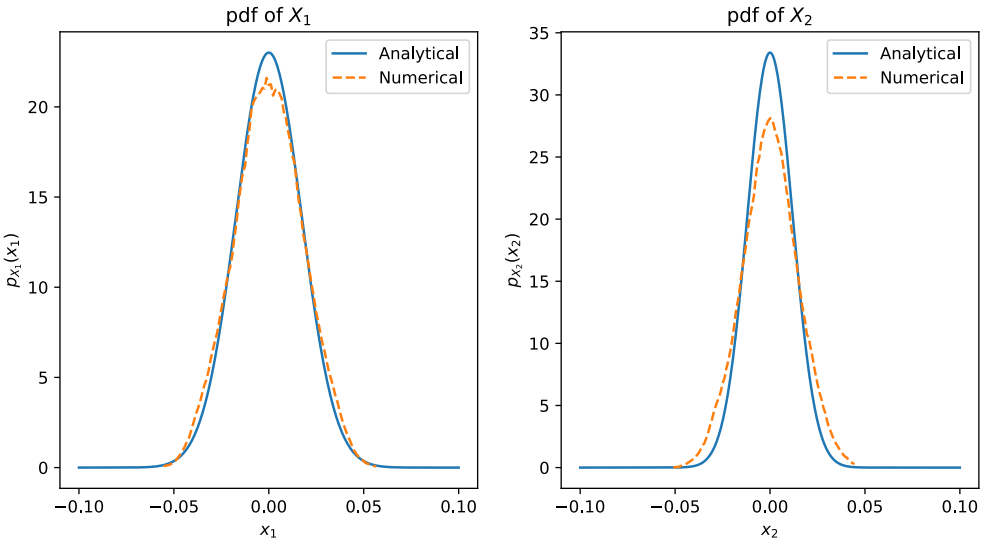


Figure 1.8: pdf $p_X(x)$ of Responses

With the simulated forcing, the PSD of the forcing can be estimated according to [1] as,

$$\begin{aligned} S_{FF} &= \lim_{N\to\infty} \frac{1}{2N+1}\mathbb{E}\left[|f|^2\right] \\ &\sim \frac{1}{2N+1}|f|^2 \end{aligned}$$

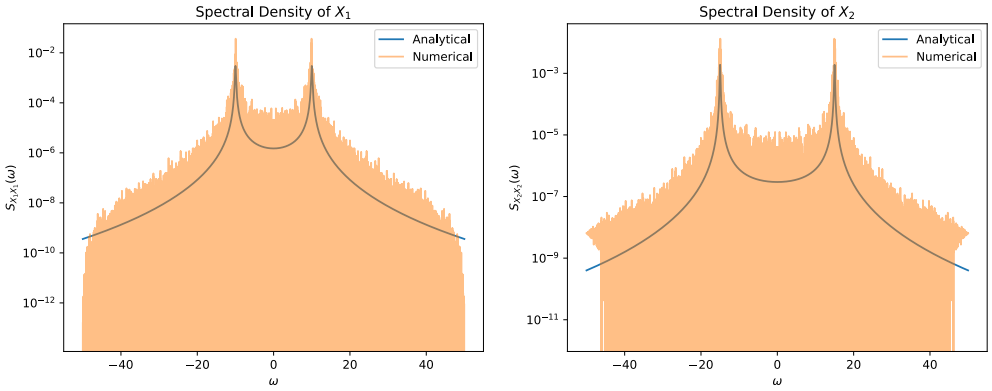


Figure 1.9: PSD $S_{XX}(\omega)$ of Responses

1.4 Part D

Next, derive the formulation for representing the response in the PCE framework. Using PCE, numerically estimate the pdf of Y. Show graphically the accuracy of your PCE based estimate with the numerically obtained estimates. Comment on the accuracy and computational costs.

1.4.1 Method

For any random process $F(t) \in \mathcal{L}^2$ space, the N^{th} generalized polynomial chaos expansion (gPCE) [2] is

$$f(t) = \sum_{k=0}^{N} \hat{f}_k \Phi_k(Z)$$

where $\Phi_k(Z)$ are orthogonal basis functions with $\mathbb{E}\left[\Phi_k(Z)\Phi_k(Z)\right] = \gamma_k$ and $\hat{f}_k = \frac{1}{\gamma_k}\mathbb{E}\left[f(t)\Phi_k(Z)\right]$.

Let $Z \sim \mathcal{N}(0,1)$ be a standard Gaussian random variable, with pdf $p_Z(z) = \frac{1}{\sqrt{2\pi}} e^{\frac{-1}{2z^2}}$, the corresponding orthogonal basis functions (Hermite Polynomial basis) is

$$
\begin{aligned}
H_0(z) &= 1 \\
H_1(z) &= z \\
H_2(z) &= z^2 - 1 \\
H_n(z) &= zH_{n-1}(z) - (n-1)H_{n-2}(z)
\end{aligned}
$$

The corresponding coefficient $\hat{f}_k$ can be computed as

$$
\hat{f}_k = \frac{1}{\gamma_k} \int_0^1 P_F^{-1}(u)\Phi_k(P_F^{-1}(u))du
$$

where P_F is the PDF of F.

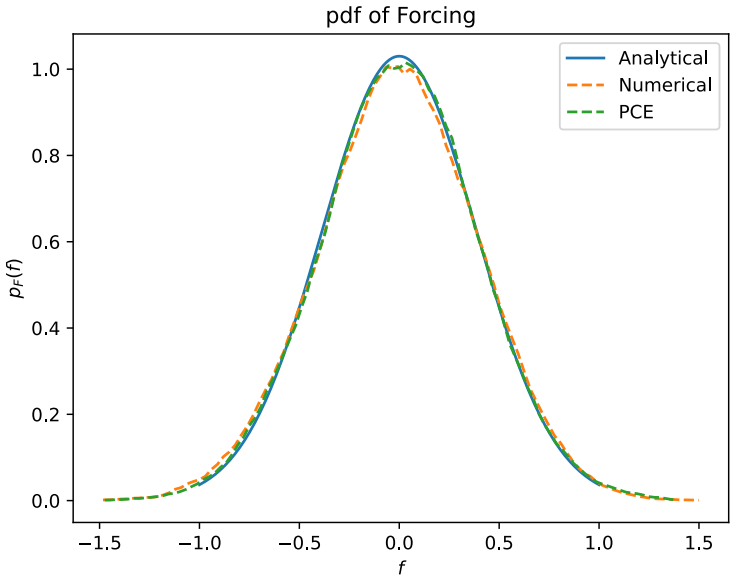


Figure 1.10: pdf $p_{_F}(f)$ of Forcing

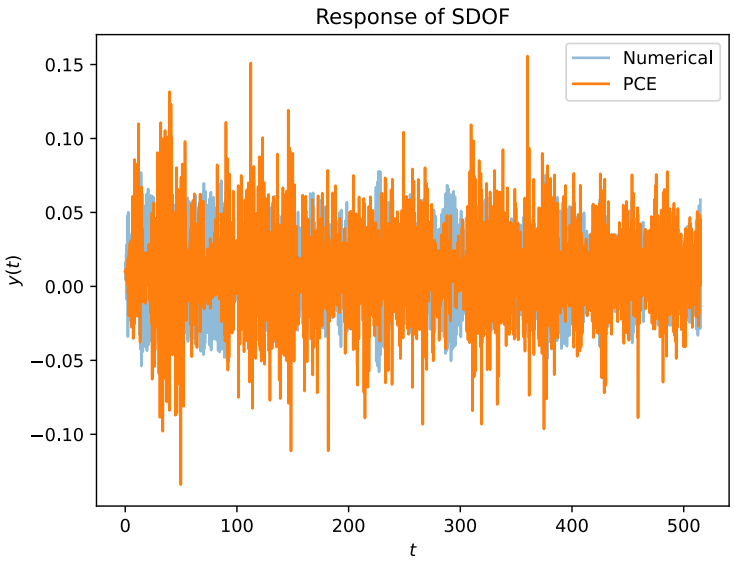


Figure 1.11: SDoF Response of Clearance

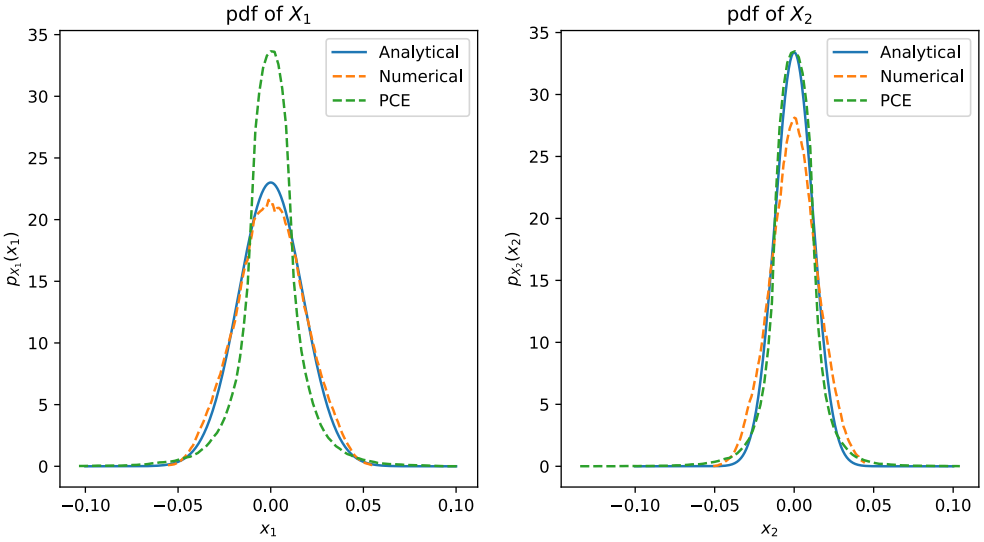


Figure 1.12: pdf $p_X(x)$ of Responses

1.5 Part E

Estimate the probability of collisions analytically, and numerically using PCE as well as through direct numerical integration. What should the value of b be such that the probability of collisions is less than 1×10^{-3}.

1.5.1 Method

Analytically, the probability of collisions is given by the value of $p_Y(y)$ at $y = 0$. We can interpolate the values from the generated pdfs at 0 for both numerically integrated and PCE techniques.

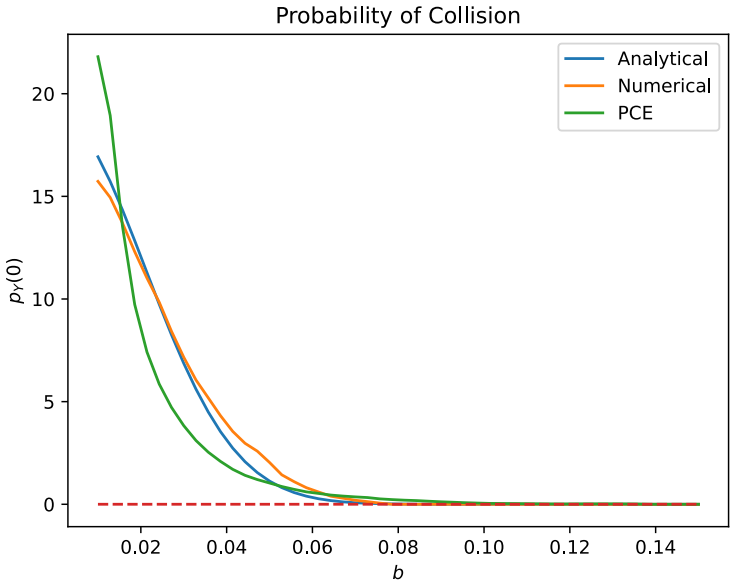


Figure 1.13: Probability of Collision

Method	b
Numerical	0.0728
Analytical	0.0957
PCE	0.1442

Table 1.1: Value of b for $P[\text{Collision}] < 10^{-3}$

References

[1] A. V. Oppenheim and G. C. Verghese, *Signals, Systems and Inference*. Prentice Hall Signal Processing Series, Prentice Hall, 2015.

[2] D. Xiu, *Numerical Methods for Stochastic Computations: A Spectral Method Approach*. Princeton University Press, 2010.

2 Code

The implementation was carried out in Python.

Setup

```
import numpy as np
import sympy as sp
import scipy as sc
import matplotlib.pyplot as plt

# Defining the Compute Parameters
N = 8196
ws = 100

b = 0.01

dw = ws/N
dt = 2*np.pi/ws

wvec = np.arange(-N/2, N/2, 1)*dw
tvec = np.arange(0, N, 1)*dt

fvec = np.linspace(-1, 1, N)
xvec = np.linspace(-0.1, 0.1, N)
yvec = np.linspace(-1, 1, N)

# Defining the PSD of the Forcing
w_1, w_2 = 10, 15
zeta_1, zeta_2 = 0.01, 0.005

w, w_j = sp.symbols('omega, omega_j', real=True)
zeta_j = sp.symbols('zeta_j', real=True)

S_FF = 6/(400+w**2)
S_FF_lambda = sp.lambdify(w, S_FF)
S_FF_num = np.vectorize(S_FF_lambda)

# Defining the FRF of the System
Hj = 1/((w_j**2-w**2)+1j*2*zeta_j*w_j*w)

# Defining the Auto-Correlation of Forcing
tau = sp.symbols('tau', real=True)
R_FF = sp.exp(-20*sp.Abs(tau)) * 6/40
R_FF_lambda = sp.lambdify(tau, R_FF)
R_FF_num = np.vectorize(R_FF_lambda)

# Defining the pdf of the Forcing
mu_f = 0
sigma2_f = sc.integrate.quad(S_FF_lambda, -np.inf, np.inf)[0]/(2*np.pi
                                                               )
f = sp.symbols('f', real=True)

p_F = 1/(sp.sqrt(2*sp.pi*sigma2_f))*sp.exp(-(f-mu_f)**2/(2*sigma2_f))
p_F_lambda = sp.lambdify(f, p_F)
p_F_num = np.vectorize(p_F_lambda)

# Computing the Auto-Correlation Matrix
```

```
rows = np.arange(0, N, 1).reshape(-1, 1)
cols = np.arange(0, N, 1).reshape(1, -1)

rows_ = np.repeat(rows, N, axis=1)
cols_ = np.repeat(cols, N, axis=0)
C_ind = np.abs(rows_ - cols_)

C = R_FF_num(C_ind*dt)
C[np.abs(C) < 1e-8] = 0

D = sc.linalg.cholesky(C)
```

```
# Defining the SDoF System
def sdof(y, t, w_j, zeta_j, f):
    y0dot = y[1]
    y1dot = f(t) - 2*zeta_j*w_j*y[1] - w_j**2*y[0]
    return np.array([y0dot, y1dot])

# Defining the PCE Functions
def hermite(n, xi):
    if n == 0:
        return 1
    elif n == 1:
        return xi
    elif n == 2:
        return xi**2 - 1
    else:
        return xi*hermite(n-1, xi) - (n-1)*hermite(n-2, xi)

def pdf_zeta(zeta):
    pdf = 1./np.sqrt(2*np.pi) * np.exp(-0.5*zeta**2)
    return pdf

def inverse_cdf_gaussian(x, mu=0, sigma2=1):
    return np.sqrt(2*sigma2)*special.erfinv(2*x-1) + mu
```

Part A

```
# Defining the PSD of the Responses
S_XX = S_FF*(Hj*sp.conjugate(Hj)).simplify()

S_X1X1 = S_XX.subs({w_j: w_1, zeta_j: zeta_1})
S_X1X1_lambda = sp.lambdify(w, S_X1X1)
S_X1X1_num = np.vectorize(S_X1X1_lambda)
mu_x1 = 0
sigma2_x1 = sc.integrate.quad(S_X1X1_lambda, -np.inf, np.inf)[0]/(2*np
                                        .pi)

S_X2X2 = S_XX.subs({w_j: w_2, zeta_j: zeta_2})
SX2X2_lambda = sp.lambdify(w, S_X2X2)
SX2X2_num = np.vectorize(SX2X2_lambda)
mu_x2 = 0
sigma2_x2 = sc.integrate.quad(SX2X2_lambda, -np.inf, np.inf)[0]/(2*np.
                                        pi)

s_X1X1_analytical = np.nan_to_num(S_X1X1_num(wvec), copy=False)
s_X2X2_analytical = np.nan_to_num(SX2X2_num(wvec), copy=False)
```

```
# Defining the pdf of the Responses
x = sp.symbols('x', real=True)
mu_x = sp.symbols('mu_x', real=True)
sigma_x = sp.symbols('sigma_x', real=True)

p_X = 1/(sp.sqrt(2*sp.pi)*sigma_x)*sp.exp(-(x-mu_x)**2/(2*sigma_x**2))


y, x_1, x_2 = sp.symbols('y, x_1, x_2', real=True)

p_X1 = p_X.subs({x: x_1, mu_x: mu_x1, sigma_x: sp.sqrt(sigma2_x1)})
p_X1_lambda = sp.lambdify(x_1, p_X1)
p_X1_num = np.vectorize(p_X1_lambda)

p_X2 = p_X.subs({x: x_2, mu_x: mu_x2, sigma_x: sp.sqrt(sigma2_x2)})
p_X2_lambda = sp.lambdify(x_2, p_X2)
p_X2_num = np.vectorize(p_X2_lambda)

p_X1_analytical = np.nan_to_num(p_X1_num(xvec), copy=False)
p_X2_analytical = np.nan_to_num(p_X2_num(xvec), copy=False)

# Defining the pdf of Clearance
def get_p_Y_analytical(b):
    p_Y = sp.integrate(p_X1*p_X2.subs({x_2: y-b+x_1}), (x_1, -sp.oo,
                                                        sp.oo))
    p_Y_lambda = sp.lambdify(y, p_Y)
    return p_Y_lambda

def collision_analytical(b, p_X1, p_X2):
    p_Y = sp.integrate(p_X1*p_X2.subs({x_2: y-b+x_1}), (x_1, -sp.oo,
                                                        sp.oo))
    p_Y_lambda = sp.lambdify(y, p_Y)
    p_Y_num = np.vectorize(p_Y_lambda)
    return p_Y_num(0)
```

Part B

```
white_noise = sc.stats.norm.rvs(size=N)

# Generating the Forcing
f_numerical = np.real(np.dot(D, white_noise))

# Estimating the PSD of the Forcing
s_FF_numerical = np.fft.fft(f_numerical)**2 / (2*N+1)
```

Part C

```
# Generating the Response
ft = lambda t: f_numerical[int(t/dt)]

x1 = sc.integrate.odeint(sdof, np.array([0, 0]), tvec, args=(w_1,
                                                             zeta_1, ft))
x2 = sc.integrate.odeint(sdof, np.array([0, 0]), tvec, args=(w_2,
                                                             zeta_2, ft))

x1_numerical = x1[:, 0]
```

```
x2_numerical = x2[:, 0]

y_numerical = b - x1_numerical + x2_numerical

def collision_numerical(b, x1_numerical, x2_numerical):
    y_generated = b - x1_numerical + x2_numerical
    bins, counts = plot_histogram(y_generated, bins=100)
    if bins[0] < 0 and 0 < bins[-1]:
        return np.interp(0, bins[:-1], counts)
    else:
        return 0
```

Part D

```
xi = np.random.normal(0, 1, size=N)

# Computing PCE for the Forcing
f0_int = lambda u: inverse_cdf_gaussian(u, 0, sigma2_f) * hermite(0,
                                        inverse_cdf_gaussian(u))
norm0_int = lambda u: hermite(0, u)**2 * pdf_zeta(u)
norm0 = sc.integrate.quad(norm0_int, -np.inf, np.inf)[0]
f0 = sc.integrate.quad(f0_int, 0, 1)[0]/norm0


f1_int = lambda u: inverse_cdf_gaussian(u, 0, sigma2_f) * hermite(1,
                                        inverse_cdf_gaussian(u))
norm1_int = lambda u: hermite(1, u)**2 * pdf_zeta(u)
norm1 = sc.integrate.quad(norm1_int, -np.inf, np.inf)[0]
f1 = sc.integrate.quad(f1_int, 0, 1)[0]/norm1

f_pce = f0 + f1 * hermite(1,  xi)

# Computing PCE for the Response
F0 = lambda t: 0
F1 = lambda t: f_pce[int(t/dt)]

x1_0 = sc.integrate.odeint(sdof, np.array([0, 0]), tvec, args=(w_1,
                                        zeta_1, F0))[:, 0]
x1_1 = sc.integrate.odeint(sdof, np.array([0, 0]), tvec, args=(w_1,
                                        zeta_1, F1))[:, 0]

x2_0 = sc.integrate.odeint(sdof, np.array([0, 0]), tvec, args=(w_2,
                                        zeta_2, F0))[:, 0]
x2_1 = sc.integrate.odeint(sdof, np.array([0, 0]), tvec, args=(w_2,
                                        zeta_2, F1))[:, 0]

x1_pce = x1_0 + x1_1  * hermite(1, xi)
x2_pce = x2_0 + x2_1  * hermite(1, xi)

y_pce = b - x1_pce + x2_pce

def collision_pce(b, x1_pce, x2_pce):
    y_pce = b - x1_pce + x2_pce
    bins, counts = plot_histogram(y_pce, bins=100)
    if bins[0] < 0 and 0 < bins[-1]:
        return np.interp(0, bins[:-1], counts)
    else:
        return 0
```

Part E

```
    bvec = np.linspace(0.01, 0.15, 50)

    # Computing the probability of collision
    p_Y_analytical = np.array([collision_analytical(b, p_X1, p_X2) for b
                                        in bvec])
    p_Y_numerical = np.array([collision_numerical(b, x1_numerical,
                                        x2_numerical) for b in bvec])
    p_Y_pce = np.array([collision_pce(b, x1_pce, x2_pce) for b in bvec])
```