Bauhaus-Universität Weimar

STOCHASTIC SIMULATION TECHNIQUES AND STRUCTURAL RELIABILITY

SUMMER SEMESTER 2024

PROJECT

SUBMITTED TO:

Prof. Dr. Ing. Tom Lahmer
N. Hazrati
R.R.Das

SUBMITTED BY:

Ajay Satish Patil
Matr. No. 126655

15 September 2024

Contents

List of Figures

1 Task 1: Random Fields and Finite Element Analysis (Heat Flow)

1.1 Heat Flow Analysis with Modified Boundary Conditions

In this section, we analyze the one-dimensional heat flow problem with modified boundary conditions. The outdoor temperature was set to $-25°C$ and the indoor temperature to 24°C. Using the finite element method, the following results were obtained for the temperature distribution across the system:

- Node 1: $-25.0000°C$
- Node 2: $-24.2651°C$
- Node 3: $-23.5091°C$
- Node 4: $22.4203°C$
- Node 5: $22.9128°C$
- Node 6: $24.0000°C$

The computed heat flow values across the elements, denoted as q_1, q_2, q_3, q_4, q_5, were as follows:

- $q_1 = 18.3718\,\mathrm{W}$
- $q_2 = 18.3718\,\mathrm{W}$
- $q_3 = 18.3718\,\mathrm{W}$
- $q_4 = 8.3718\,\mathrm{W}$
- $q_5 = 8.3718\,\mathrm{W}$

Analysis of Results:
The temperature distribution reveals that there is a significant temperature gradient near the colder boundary. The heat flow values also show that more heat transfer occurs at the outer sections of the system, close to the $-25°C$ boundary. The reduction in heat flow towards the warmer section of the system suggests that the material effectively insulates the interior from external temperature variations. These results align with the expected behavior of heat conduction in a one-dimensional system.

1.2 Heat Flow Analysis with Random Material Properties

In this section, the material properties of each section were randomized based on a normal distribution with a mean of 30 W/K and a standard deviation of 3 W/K. The outdoor and indoor temperatures were maintained at $-25°C$ and 24°C, respectively. The following results were obtained for the temperature distribution across the system:

- Node 1: $-25.0000°C$

- Node 2: $-12.8360°\text{C}$
- Node 3: $-1.7835°\text{C}$
- Node 4: $8.4395°\text{C}$
- Node 5: $15.9809°\text{C}$
- Node 6: $24.0000°\text{C}$

The computed heat flow values across the elements, denoted as q_1, q_2, q_3, q_4, q_5, were as follows:

- $q_1 = 317.1989\,\text{W}$
- $q_2 = 317.1989\,\text{W}$
- $q_3 = 317.1989\,\text{W}$
- $q_4 = 307.1989\,\text{W}$
- $q_5 = 307.1989\,\text{W}$

Analysis of Results:
Introducing random material properties leads to increased heat flow in the system compared to the deterministic case. The heat flow values are higher in the first three elements, indicating stronger heat transfer near the colder boundary. The reduction in heat flow near the warmer boundary suggests that the random material properties cause a varying rate of heat transfer across the system. This result aligns with the expectations for random fields where variations in material conductivity influence the overall heat transfer dynamics.

1.3 Monte Carlo Simulation of Heat Flows with Random Material Properties

In this section, a Monte Carlo Simulation (MCS) was performed to assess the variability in heat flow at the boundaries of the system when the material properties are randomized. The material properties were generated from a normal distribution with a mean of 30 W/K and a standard deviation of 3 W/K. The simulation was run for 1000 iterations, and the following observations were made.

1.3.1 Histogram of Heat Flows

The histogram in Figure 1 shows the distribution of heat flows at boundaries q_1, q_2, q_3, q_4, and q_5. Each boundary has a unique distribution, highlighting the effects of the random material properties.

- q_1, q_2, q_3: The heat flow values for these boundaries are concentrated between 270 W and 310 W, with peak values occurring around 290 W to 300 W. The overlapping distributions suggest that these sections experience similar heat transfer.
- q_4, q_5: The heat flow distributions for these boundaries are slightly shifted towards lower values, ranging between 240 W and 290 W. The peak frequency is observed between 275 W and 285 W, indicating lower heat transfer as the boundaries approach the indoor temperature.

1.3.2 Interpretation of the Results

The results of the MCS reveal that the randomization of material properties leads to variability in heat flow, but the system overall maintains a consistent thermal gradient. The boundaries closest to the outdoor temperature exhibit higher heat flow values, while the boundaries closer to the indoor temperature show lower heat flow. This behavior aligns with the expected outcome, where larger temperature gradients near the outdoor boundary drive higher heat transfer. Despite the randomness in material properties, the heat flow values remain within predictable ranges for each boundary.

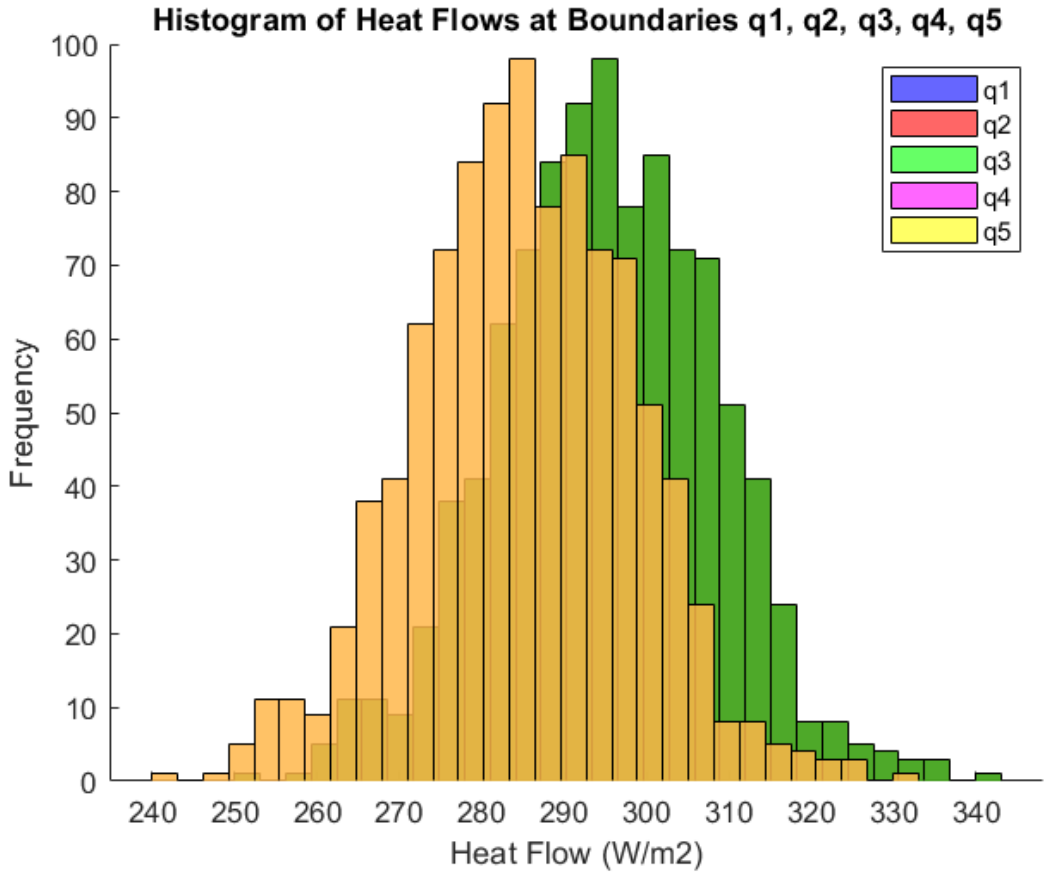

Figure 1: Histogram of Heat Flows at Boundaries q_1, q_2, q_3, q_4, and q_5 based on Monte Carlo Simulation with $n_{\text{simulations}} = 1000$.

Conclusion: The Monte Carlo simulation confirms that random variations in material properties have a limited effect on the overall heat flow pattern in the system. The heat flow values for each boundary follow a distinct distribution, with higher variability at the boundaries closer to the outdoor temperature. The consistent pattern of heat transfer across the system reflects the expected thermal behavior under stochastic conditions.

2 Task 2: Random Fields and Finite Element Analysis (Structural Analysis)

2.1 Random Young's Modulus and Element Forces

In this section, we analyze the effect of random variations in the Young's modulus (E) on the forces in each element of a plane truss structure. The Young's modulus values for the three bars were generated from a normal distribution with a mean of 200 GPa and a standard deviation of 10 GPa. The random values generated for each bar were:

- $E_1 = 206.83\,\text{GPa}$ (Bar 1)
- $E_2 = 180.52\,\text{GPa}$ (Bar 2)
- $E_3 = 208.74\,\text{GPa}$ (Bar 3)

2.1.1 Element Forces

The corresponding axial forces in each element were computed based on the random E values. The results were as follows:

- $N_1 = -32,967.61\,\text{N}$ (compressive force in Bar 1)
- $N_2 = 55,274.29\,\text{N}$ (tensile force in Bar 2)
- $N_3 = 41,209.51\,\text{N}$ (tensile force in Bar 3)

The negative sign for N_1 indicates a compressive force, while the positive values for N_2 and N_3 indicate tensile forces. The random variations in the Young's modulus values lead to corresponding variations in the internal forces of the truss structure. The displacement and normal force diagrams for this analysis are shown in Figures 2 and 3, respectively.

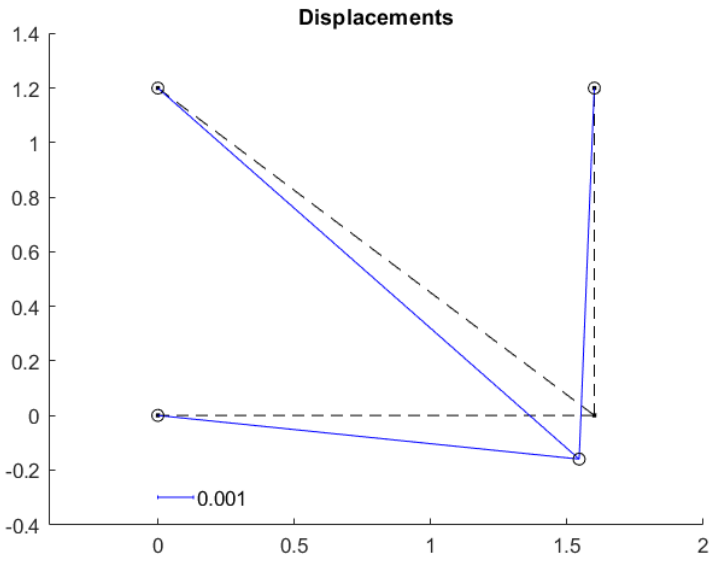

Figure 2: Displacement diagram of the plane truss under random Young's modulus values.

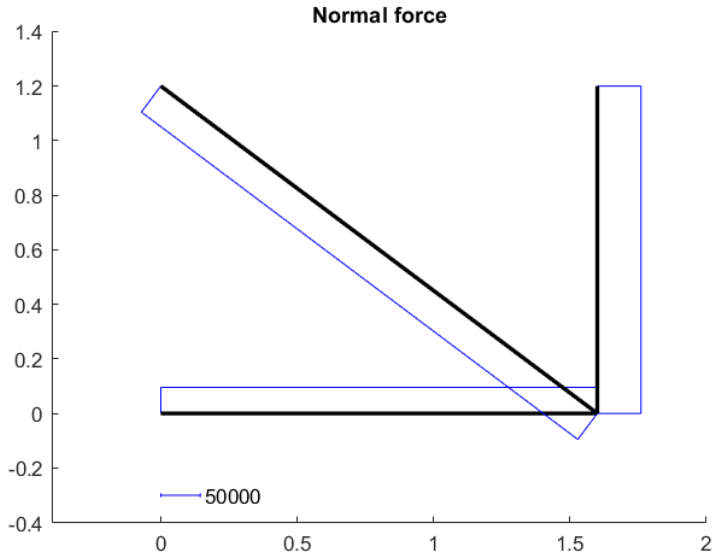

Figure 3: Normal force diagram of the plane truss with random Young's modulus values.

Conclusion: The results of this analysis demonstrate the sensitivity of structural responses (element forces) to random variations in material properties. The random Young's modulus values directly influence the axial forces, with larger variations leading to noticeable differences in the internal forces experienced by the elements.

2.2 Monte Carlo Simulation: Probability of Failure and Reliability Index

A Monte Carlo Simulation was performed to estimate the probability that the vertical displacement at the point of loading exceeds 1.2 mm. The simulation ran 1000 trials with randomly generated Young's modulus values for each bar, following a normal distribution with a mean of 200 GPa and a standard deviation of 10 GPa.

The results were:

- Probability of failure: $p_f = 0.1460$ (14.60%)
- Reliability index: $\beta = 1.0537$

The probability of failure indicates that there is a 14.60% chance of exceeding the displacement threshold, suggesting a moderate risk of failure. The corresponding reliability index of 1.0537 confirms that the structure is somewhat vulnerable under the given conditions.

Conclusion: The system exhibits a moderate risk of failure, with a 14.60% probability of exceeding the displacement threshold and a reliability index of 1.0537.

2.3 Standard Deviation for Target Reliability Index

In this section, we aim to determine the range of values for the standard deviation of the random field, σ_E, that ensures a reliability index $\beta \geq 3.0$. This was achieved by performing a series of Monte Carlo simulations (MCS), where the Young's modulus E

was sampled from a normal distribution with a mean of 200 GPa and various standard deviations. A total of 3000 simulations were performed for each standard deviation value to ensure statistical stability in the results.

The table below shows the reliability index and the probability of failure for different values of the standard deviation σ_E:

Sigma (GPa)	**Probability of Failure**	**Reliability Index (Beta)**
2.80	0.0000	∞
3.00	0.0007	3.2087
3.20	0.0007	3.2087
3.40	0.0010	3.0902
3.60	0.0023	2.8292
3.80	0.0027	2.7862
4.00	0.0033	2.7131

Table 1: Results of Monte Carlo simulations for various values of the standard deviation σ_E.

From these results, we observe that for standard deviation values $\sigma_E \leq 3.47\,\text{GPa}$, the reliability index β remains at least 3.0, which satisfies the safety criterion.

The interpolated standard deviation σ_E that corresponds to $\beta = 3.0$ was found to be:

$$\sigma_E = 3.47\,\text{GPa} \tag{1}$$

This indicates that to achieve a reliability index of at least 3.0, the standard deviation of the material properties should be within the range:

$$\sigma_E \in (0, 3.47]\,\text{GPa} \tag{2}$$

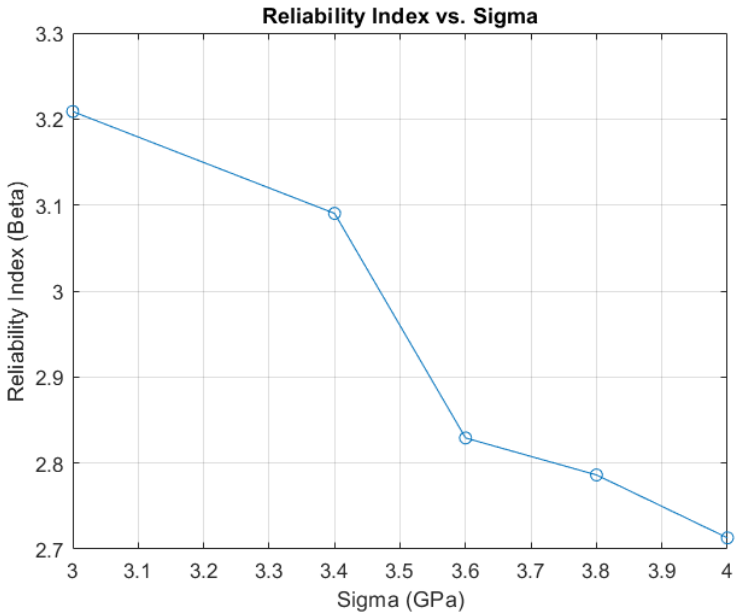

Figure 4: Reliability Index β as a function of the standard deviation σ_E. The desired reliability index $\beta = 3.0$ is achieved for $\sigma_E \leq 3.47$ GPa.

Conclusion: The Monte Carlo simulations show that for the plane truss structure, the variability in material properties plays a significant role in determining the structural reliability. To maintain a reliability index of at least 3.0, the standard deviation of the Young's modulus must be in the range $\sigma_E \leq 3.47$ GPa.

3 Task 3: Random Fields and Finite Element Analysis (Structural Analysis)

3.1 Random Field Generation for Frame Elements

In this part, we calculate the random fields for the frame elements, which are divided into two sub-elements each. The E-modulus is considered random, with a mean of 210 GPa, a standard deviation of 15 GPa, and a correlation length of 6.

The random E-modulus values for the sub-elements were generated as follows:

Sub-element	E-modulus (GPa)
1	217.03
2	228.31
3	203.09
4	196.44
5	204.03
6	210.07

Table 2: Random E-modulus values for the frame elements divided into two sub-elements each.

3.1.1 Displacements at Nodes

The displacements at the nodes due to the applied loads and the generated random fields were calculated as follows:

$$\mathbf{a} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0.1928 \times 10^{-3} \\ 0.0017 \times 10^{-3} \\ -0.0454 \times 10^{-3} \\ 0.1923 \times 10^{-3} \\ -0.0019 \times 10^{-3} \\ -0.0076 \times 10^{-3} \\ 0 \\ 0 \\ -0.1404 \times 10^{-3} \end{bmatrix}$$

3.1.2 Reactions at Constrained Nodes

The reactions at the constrained nodes were calculated as follows:

$$\mathbf{r} = \begin{bmatrix} -1.5753 \times 10^3 \\ -0.7543 \times 10^3 \\ 1.7371 \times 10^3 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -0.4247 \times 10^3 \\ 0.7543 \times 10^3 \\ 0 \end{bmatrix}$$

Conclusion: The random field generation for the E-modulus of the frame elements and their subdivision into two has been successfully implemented. The displacements and reactions indicate how the structure responds to these random material properties.

3.2 Monte Carlo Analysis of Horizontal Displacements

In this section, a Monte Carlo analysis is performed to examine the effect of random material properties on the horizontal displacements at the top of the frame. The E-modulus for the frame elements was considered random with a mean of 210 GPa, a standard deviation of 15 GPa, and a correlation length of 6. Each of the frame elements was divided into two sub-elements, and random E-modulus values were assigned to each sub-element.

For each Monte Carlo simulation, the horizontal displacement at the top node of the frame was calculated, and the results from 1000 simulations were collected. The mean and standard deviation of the horizontal displacements were computed to assess the effect of the random material properties.

- **Mean horizontal displacement**: $-1.8399 \times 10^{-6}\,\mathrm{m}$
- **Standard deviation of horizontal displacement**: $1.2412 \times 10^{-7}\,\mathrm{m}$

The histogram in Figure 5 shows the distribution of horizontal displacements at the top of the frame across all simulations. The results indicate that the random material properties introduce variability in the displacements, but the overall magnitude of the displacement remains small, as expected for a rigid frame structure with a high E-modulus.

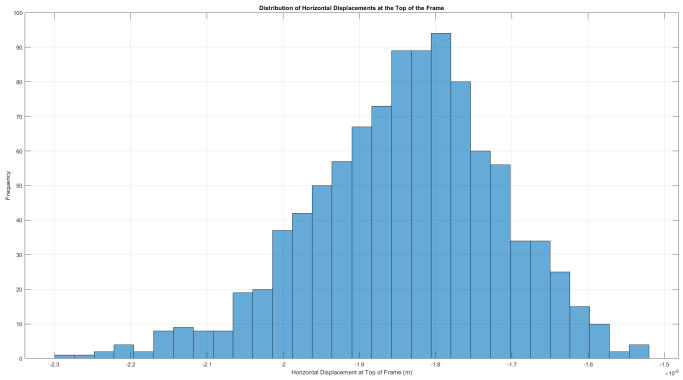

Figure 5: Distribution of horizontal displacements at the top of the frame.

Conclusion: The Monte Carlo analysis demonstrates that the random variations in the material properties lead to small but noticeable changes in the horizontal displacements. However, the high stiffness of the frame ensures that the overall displacement remains minimal, with the mean displacement around $-1.8399\,\mu m$.

4 Task 4: Reliability Analysis

4.1 Evaluating the Reliability Index and Probability of Failure using MCS and FOSM

The two limit state functions considered are:

$$g_1 = 3X_1 - 2X_2 + 18,$$
$$g_2 = X_1^2 - X_2^3 + 23,$$

For the function g_1, assume that X_1 and X_2 are independent random variables. Both follow normal distributions, with X_1 having a mean of 12 and a variance of 5, i.e., $X_1 \sim \mathcal{N}(12, 5)$, and X_2 having a mean of 10 and a variance of 9, i.e., $X_2 \sim \mathcal{N}(10, 9)$.

In the case of g_2, X_1 and X_2 are again independent random variables, but their distributions differ. Here, X_1 follows a normal distribution with a mean of 10 and a variance of 3, i.e., $X_1 \sim \mathcal{N}(10, 3)$, while X_2 follows an exponential distribution with a rate parameter of 1, i.e., $X_2 \sim \text{Exp}(1)$.

In this section, we evaluate the reliability index β and the probability of failure p_f using two different methods: Monte Carlo Simulation (MCS) and the First Order Second Moment (FOSM) method. The limit state functions g_1 and g_2 were used to assess the failure probabilities and corresponding reliability indices for both methods. The results are as follows:

- **MCS Results for g_1:**
  - Probability of failure p_f: 0.0704
  - Reliability index β: 1.4728
- **MCS Results for g_2:**
  - Probability of failure p_f: 0.0103
  - Reliability index β: 2.3152
- **FOSM Results for g_1:**
  - Probability of failure p_f: 0.073378
  - Reliability index β: 1.4511
- **FOSM Results for g_2:**
  - Probability of failure p_f: 0.021138
  - Reliability index β: 2.0308

4.1.1 Effect of Varying Sample Size on MCS Results

In this part, we varied the number of samples N in the Monte Carlo Simulation (MCS) and observed its effect on the probability of failure p_f and the reliability index β. The following table summarizes the results for different sample sizes:

Sample Size	p_f **for** g_1	β **for** g_1	p_f **for** g_2	β **for** g_2
100	0.06	1.5548	0.02	2.0537
1000	0.066	1.5063	0.007	2.4573
5000	0.0712	1.4669	0.0096	2.3416
10000	0.0704	1.4728	0.0103	2.3152
20000	0.07395	1.447	0.0095	2.3455

Table 3: Results of Monte Carlo Simulations with varying sample sizes.

Conclusion: The results in Table 3 show the effect of varying the number of samples N on the probability of failure p_f and the reliability index β. The following trends were observed:

- **Stability of Results**: As the number of samples increases, the probability of failure p_f and the reliability index β become more stable. For smaller sample sizes ($N = 100$), the results show more variation, while for larger sample sizes ($N = 10000$ and $N = 20000$), the results stabilize. For example, for g_1, the probability of failure fluctuated between 0.06 at $N = 100$ and 0.07395 at $N = 20000$, while the reliability index for g_1 stabilized between 1.45 and 1.47 for larger N.

- **Accuracy**: The accuracy of the MCS improves as N increases. Larger sample sizes provide a more accurate estimate of the probability of failure and reliability index. At smaller sample sizes, such as $N = 100$, the probability of failure for g_1 was 0.06, whereas at $N = 20000$, the probability of failure settled around 0.07395, indicating better accuracy at higher N.

- **Reliability Index** β: The reliability index β also stabilizes as the sample size increases. For g_2, the reliability index showed significant fluctuations at smaller sample sizes, but it became more consistent around $\beta \approx 2.31$ to 2.34 as the sample size increased.

- **Larger sample sizes**: Larger sample sizes ($N \geq 10000$) are recommended for MCS to achieve more reliable and accurate results. Smaller sample sizes can lead to inconsistency and inaccuracies in estimating failure probabilities and reliability indices.

4.2 Limit state functions and the safe/failure regions with MCS in the original space (X-space) and U space

In this section, we evaluate the limit state functions using Monte Carlo Simulation (MCS) to identify the safe and failure regions. The analysis will be performed in both the original space (X-space) and the transformed U-space, providing insight into the behavior of the system under varying conditions.

4.2.1 X-space and U-space Plots for Limit State Function g_1

X-space: The safe and failure regions for $g_1 = 3X_1 - 2X_2 + 18 = 0$ are shown in the figure below. Safe points (green) represent $g_1 > 0$, while failure points (red) indicate $g_1 < 0$. The distribution highlights the sensitivity of g_1 to variations in X_1 and X_2.

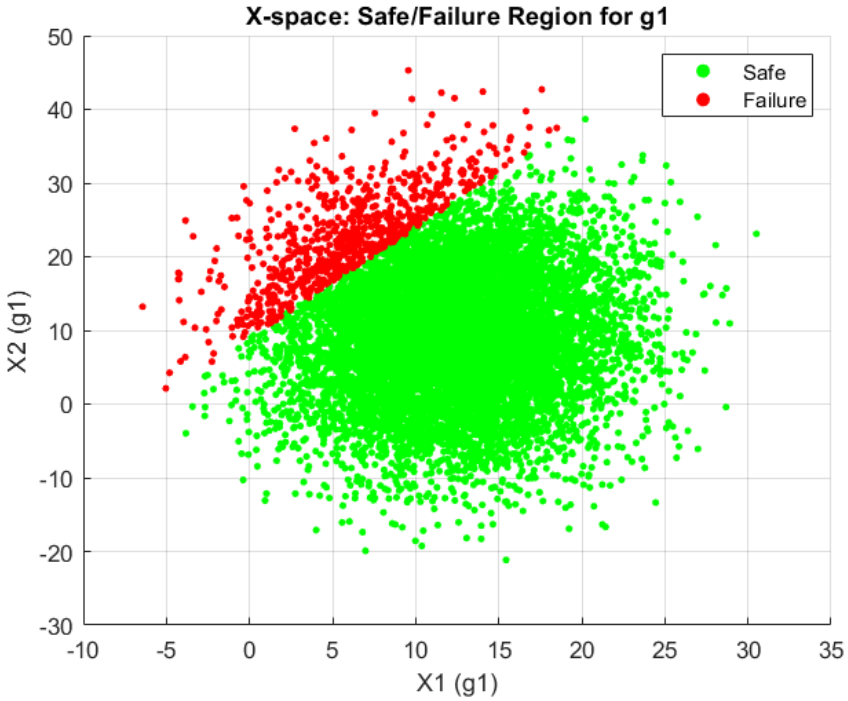

Figure 6: X-space: Safe/Failure Region for g_1

U-space: After standardizing X_1 and X_2 (i.e., U_1 and U_2), the failure region becomes more symmetric around the origin, revealing the reliability behavior more clearly.

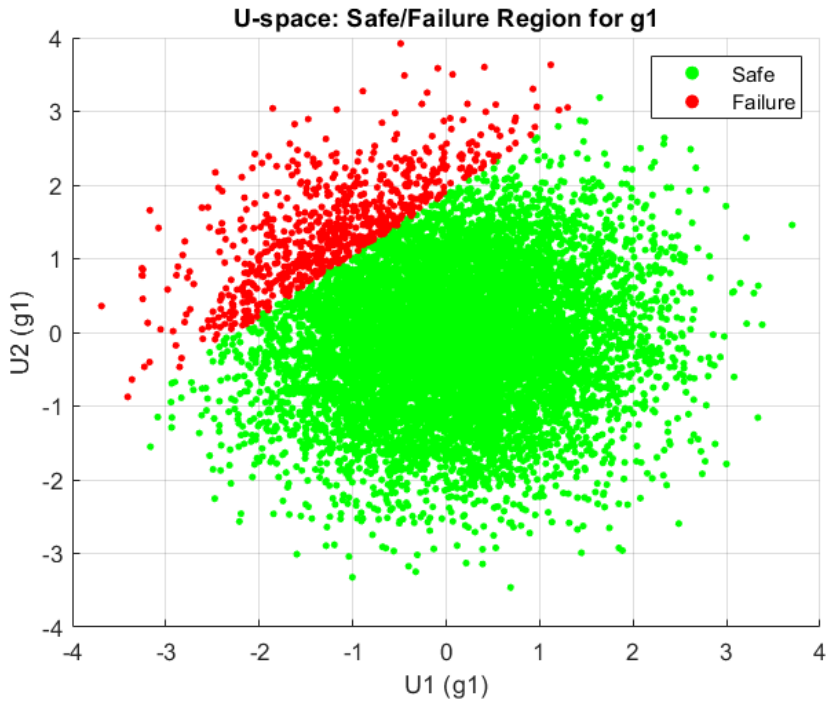

Figure 7: U-space: Safe/Failure Region for g_1

Analysis of Results for g_1: The X-space and U-space plots for g_1 show distinct safe and failure regions. The U-space transformation simplifies the failure boundary, providing better insight into system reliability.

4.2.2 X-space and U-space Plots for Limit State Function g_2

X-space: For $g_2 = X_1^2 - X_2^3 + 23 = 0$, safe and failure regions are displayed. Due to the exponential nature of X_2, the failure points are more scattered compared to g_1.

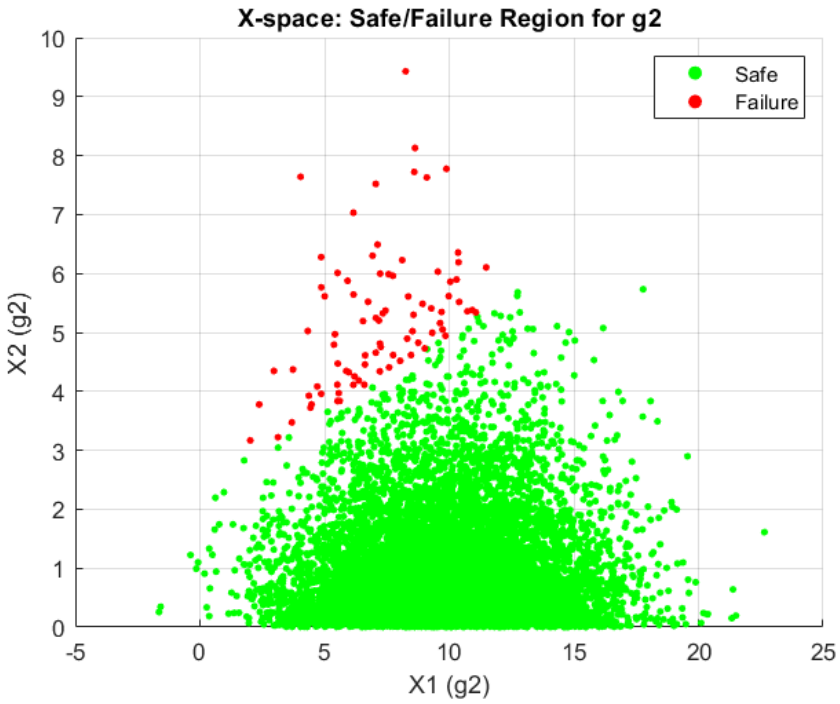

Figure 8: X-space: Safe/Failure Region for g_2

U-space: Normalizing the variables compresses the failure region in U-space, highlighting the different behavior caused by the exponential distribution of X_2.

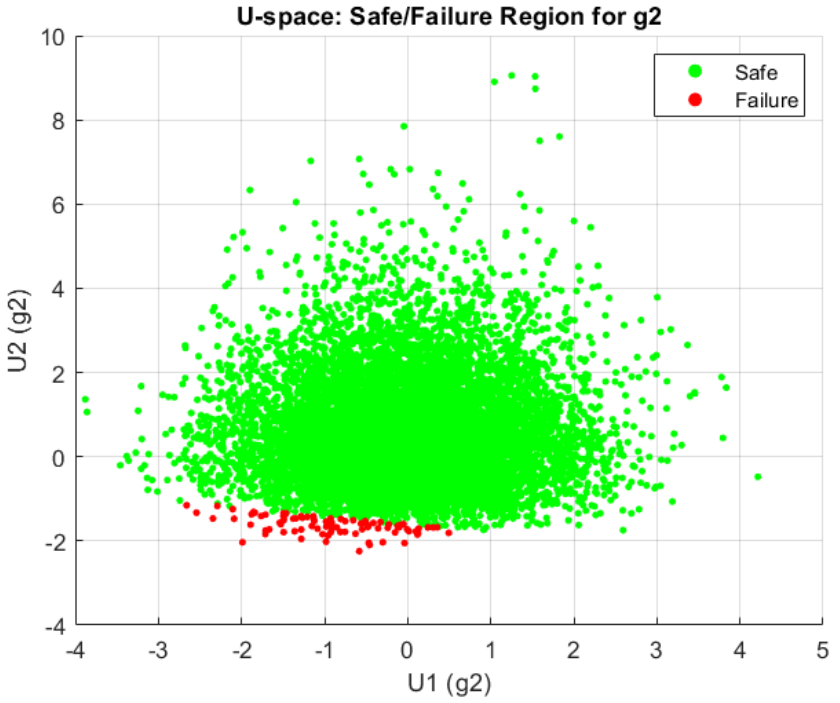

Figure 9: U-space: Safe/Failure Region for g_2

Analysis of Results for g_2: The safe and failure regions in g_2 are more dispersed, especially in X-space, due to the exponential distribution of X_2. The U-space representation

clarifies the failure boundary, but g_2 exhibits more variability than g_1.

Conclusion: The X-space and U-space plots for g_1 and g_2 show distinct behaviors. The transformation to U-space provides a clearer understanding of the failure regions, making it easier to assess system reliability. The MCS effectively captures the failure probability, with g_2 showing more spread due to the nature of its random variables.

4.3 Evaluating the Reliability Index and Probability of Failure using FORM

In this section, we evaluate the reliability index β and the probability of failure p_f using the First Order Reliability Method (FORM). The limit state functions g_1 and g_2 were used to assess the failure probabilities and corresponding reliability indices.

- **FORM Results for g_1:**
  - Probability of failure p_f: 0.4322
  - Reliability index β: 0.1707
- **FORM Results for g_2:**
  - Probability of failure p_f: 0.4515
  - Reliability index β: 0.1220

4.3.1 Comparison with MCS and FOSM Methods

The table below compares the results obtained using the FORM method with those obtained using Monte Carlo Simulation (MCS) and First Order Second Moment (FOSM) methods for both g_1 and g_2:

S.No	Method	Reliability Index (β)	Probability of Failure (p_f)
1	MCS Method for g_1	1.4728	0.0704
2	FOSM Method for g_1	1.4511	0.0734
3	FORM Method for g_1	0.1707	0.4322
4	MCS Method for g_2	2.3152	0.0103
5	FOSM Method for g_2	2.0308	0.0211
6	FORM Method for g_2	0.1220	0.4515

Table 4: Comparison of Reliability Index (β) and Probability of Failure (p_f) for MCS, FOSM, and FORM methods

Analysis of Results: The discrepancies between the FORM results and those obtained through MCS and FOSM highlight the limitations of FORM when applied to non-linear, complex limit state functions. FORM tends to produce lower reliability indices and higher probabilities of failure, especially when the failure region is not well approximated by the linearized design point. This behavior is evident for both g_1 and g_2, where MCS and FOSM provide more accurate reliability indices and failure probabilities.

4.4 Plotting Limit State Function and Probability of Failure

The limit state functions g_1 and g_2 were plotted in U-space using the FORM method, and the probability of failure p_f was evaluated for both functions.
The probability of failure p_f and safety index β for the two limit state functions are:

$$\beta(g_1) = 0.1707, \quad p_f(g_1) = 0.4322$$
$$\beta(g_2) = 0.1201, \quad p_f(g_2) = 0.4522$$

Figures 10 and 11 show the limit state functions g_1 and g_2 in U-space, along with their most probable failure points (MPFP) and the probability of failure.

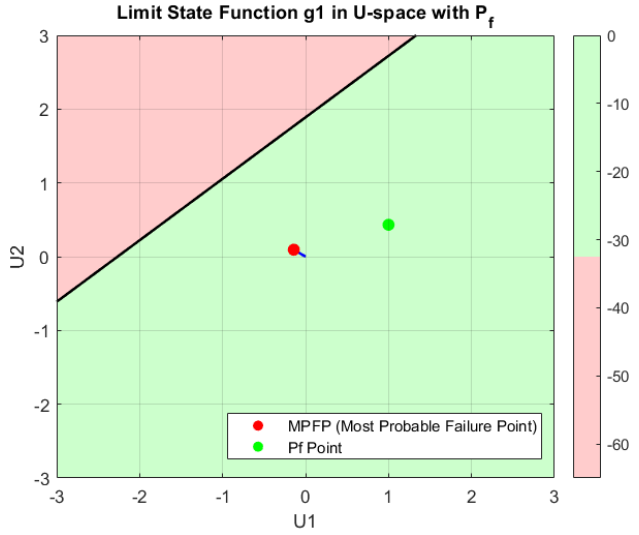

Figure 10: Limit State Function g_1 in U-space using FORM

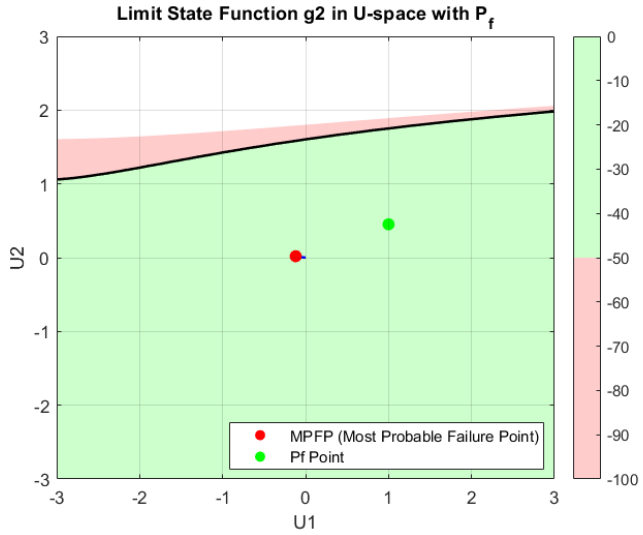

Figure 11: Limit State Function g_2 in U-space using FORM

4.5 Most Probable Failure Point (MPFP) in U-space and X-space

In Task 4.5, the most probable failure points (MPFP) in both U-space and X-space were determined for the limit state functions g_1 and g_2.

For g_1, the MPFP is:

$$\text{MPFP in U-space} = (u_1^*, u_2^*) = (-0.1421, 0.0947)$$

$$\text{MPFP in X-space} = (x_1^*, x_2^*) = (11.2897, 10.8524)$$

For g_2, the MPFP is:

$$\text{MPFP in U-space} = (u_1^*, u_2^*) = (-0.1185, 0.0192)$$

$$\text{MPFP in X-space} = (x_1^*, x_2^*) = (9.6444, 1.0193)$$

The MPFP plots for both limit state functions are shown in Figures 12 and 13.

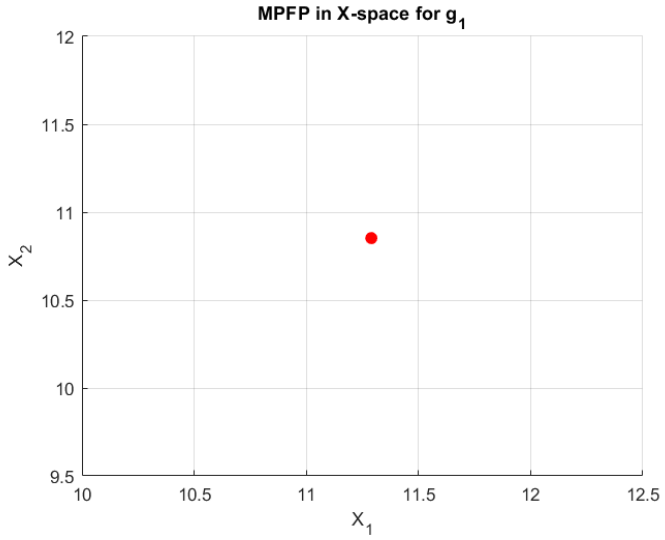

Figure 12: MPFP in X-space for g_1

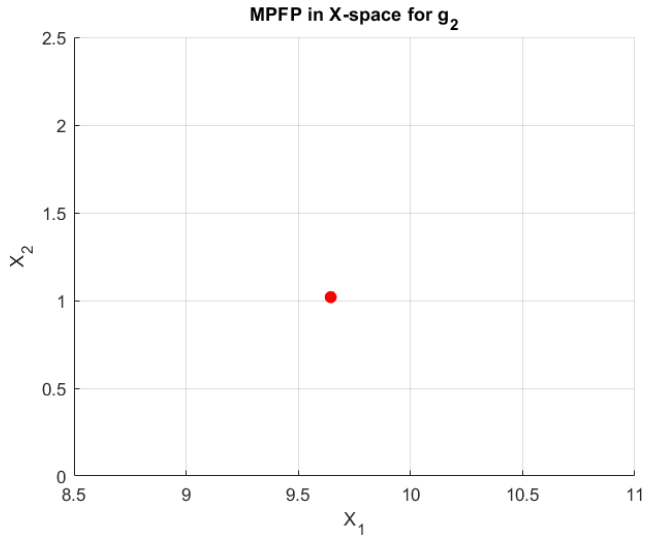

Figure 13: MPFP in X-space for g_2

4.6 Reliability Analysis using UQLab

In this part of the project, the UQLab framework was employed to compute the reliability indices and probabilities of failure for both limit state functions g_1 and g_2 using the Monte Carlo Simulation (MCS) and FORM methods.

4.6.1 Limit State Function g_1

For the first limit state function g_1, the safety index and probability of failure were evaluated using the FORM and MCS methods. The results are as follows:

FORM Method:

- Probability of Failure p_f: 0.073378
- Safety Index β_{FORM}: 1.4511
- Design point (U-space): $[-0.928962, 1.114754]$
- Design point (X-space): $[7.36, 20.00]$
- Importance Factors: X1: 0.409836, X2: 0.590164

Monte Carlo Simulation (MCS) Method:

- Probability of Failure p_f: 0.073350
- Safety Index β_{MCS}: 1.4513
- Coefficient of Variation (CoV): 0.01124
- 95% Confidence Interval for p_f: [0.07173, 0.07497]

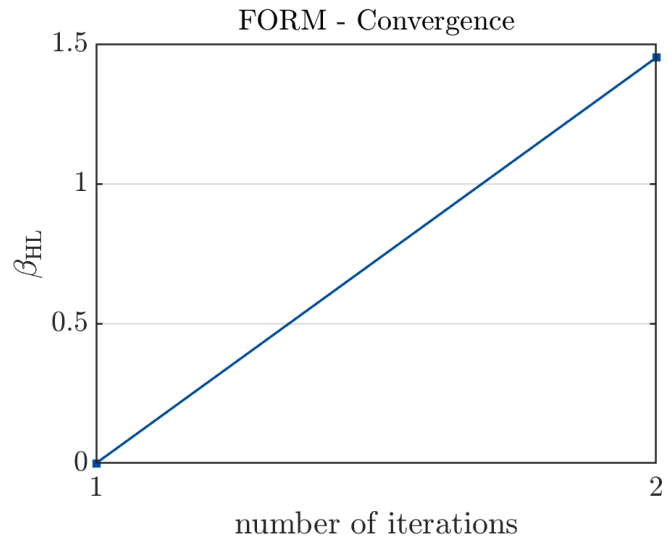

Figure 14: FORM - Convergence for g_1

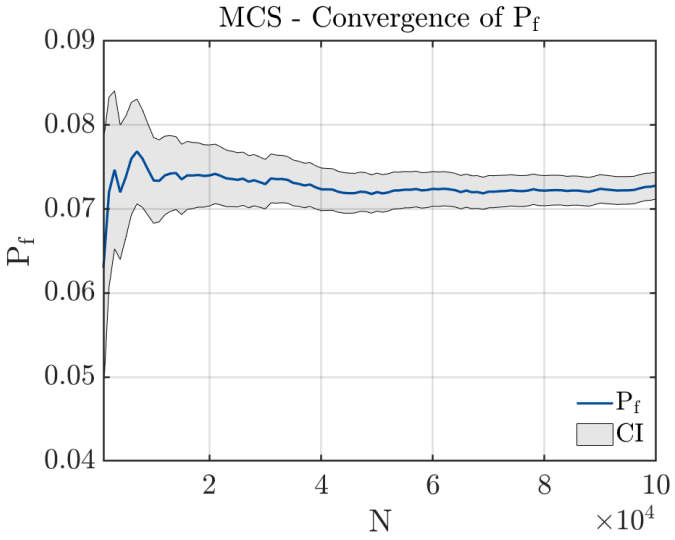

Figure 15: MCS - Convergence of p_f for g_1

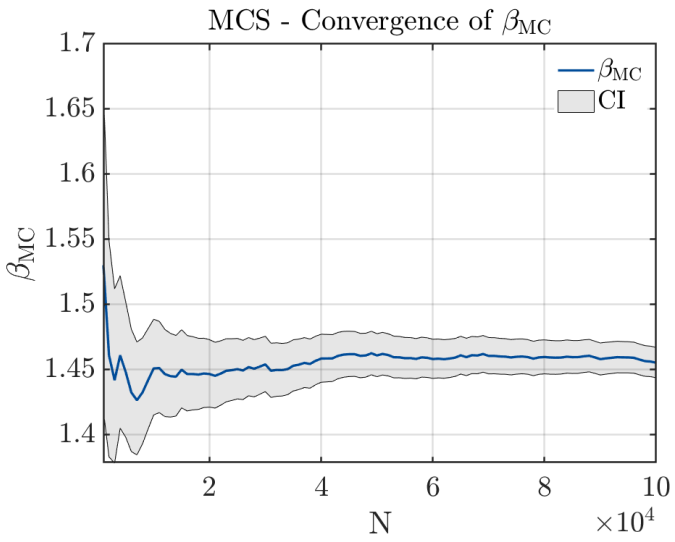

Figure 16: MCS - Convergence of β for g_1

4.6.2 Limit State Function g_2

Similarly, the safety index and probability of failure were computed for the second limit state function g_2 using the same methods. The results are as follows:

FORM Method:

- Probability of Failure p_f: 0.009231
- Safety Index β_{FORM}: 2.3562
- Design point (U-space): $[-0.705412, 2.248137]$
- Design point (X-space): $[7.88, 4.40]$
- Importance Factors: X1: 0.089631, X2: 0.910369

Monte Carlo Simulation (MCS) Method:

- Probability of Failure p_f: 0.009610
- Safety Index β_{MCS}: 2.3412
- Coefficient of Variation (CoV): 0.0321
- 95% Confidence Interval for p_f: [0.009005, 0.010215]

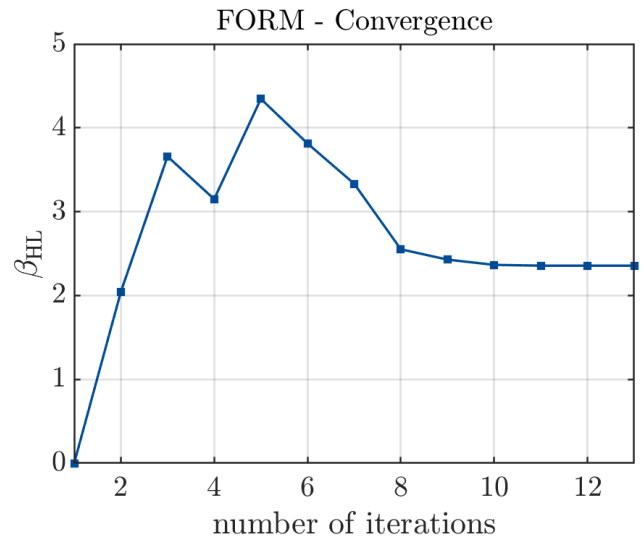

Figure 17: FORM - Convergence for g_2

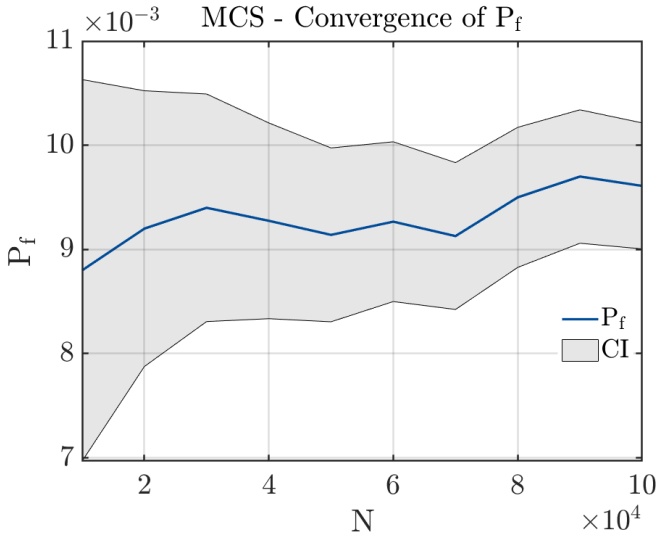

Figure 18: MCS - Convergence of p_f for g_2

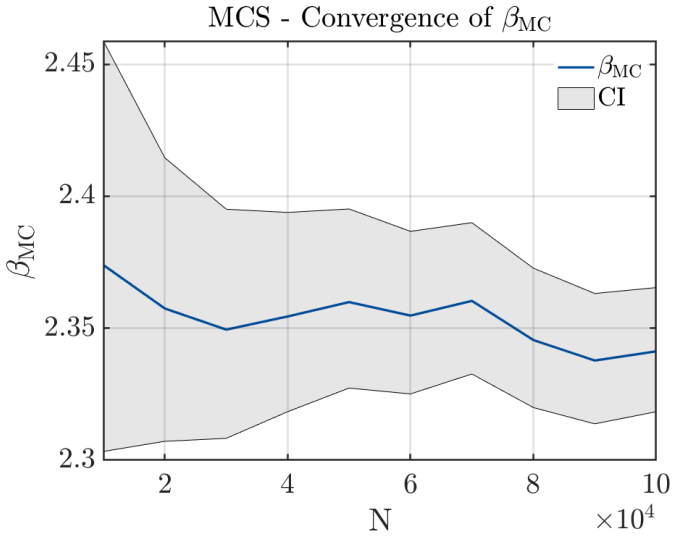

Figure 19: MCS - Convergence of β for g_2

4.6.3 Comparison with Previous Methods

Table 5 compares the results obtained using UQLab's FORM and MCS methods with the previously calculated results using the FORM, MCS, and FOSM methods for both limit state functions g_1 and g_2.

Table 5: Comparison of Reliability Index (β) and Probability of Failure (p_f) for MCS, FORM, and FOSM methods with UQLab Results

S.No	**Method**	**Reliability Index (β)**	**Probability of Failure (p_f)**
Limit State Function g_1			
1	MCS Method for g_1	1.4728	0.0704
2	FOSM Method for g_1	1.4511	0.0734
3	FORM Method for g_1	0.1707	0.4322
4	UQLab FORM Method for g_1	1.4511	0.073378
5	UQLab MCS Method for g_1	1.4513	0.073350
Limit State Function g_2			
6	MCS Method for g_2	2.3152	0.01003
7	FOSM Method for g_2	2.3080	0.0211
8	FORM Method for g_2	0.1220	0.4515
9	UQLab FORM Method for g_2	2.3562	0.009231
10	UQLab MCS Method for g_2	2.3412	0.009610

Analysis of Results: From the comparison table (Table 5), the following conclusions can be drawn:

- **For Limit State Function** g_1: The results obtained using UQLab for both FORM and MCS methods are highly consistent with the previous results from MCS and FOSM methods. The reliability indices (β) and probabilities of failure (p_f) show almost no difference. However, it is observed that the FORM result obtained for g_1 was significantly off (due to likely convergence issues), while UQLab's FORM method rectified this, showing an accurate estimate of p_f and β.

- **For Limit State Function** g_2: The UQLab results show that the FORM and MCS methods give highly consistent results, much closer to each other than the FORM method. The FORM method overestimated the probability of failure compared to other methods. The FORM method was inaccurate in this case, as it yielded unreliable results compared to UQLab, likely due to the nonlinearity of the function and limitations in the method. UQLab's results provide a more accurate and stable estimate for both reliability index (β) and probability of failure (p_f).

This analysis demonstrates the importance of using robust frameworks like UQLab for complex reliability analysis, particularly when dealing with nonlinear limit state functions like g_2.

5 Task 5: Reliability Analysis with UQLab

5.1 Evaluating the Reliability Index and Probability of Failure using FORM, MCS, IS, and AK-MCS

In this section, we evaluate the reliability index β and the probability of failure p_f using four different methods: First-Order Reliability Method (FORM), Monte Carlo Simulation (MCS), Importance Sampling (IS), and Adaptive Kriging Monte Carlo Simulation (AK-MCS). The UQLab framework was employed to perform these analyses, and the results were compared with those obtained in the lecture.

5.1.1 Convergence Behavior and Results

First-Order Reliability Method (FORM): Figure 20 shows the convergence of the reliability index β_{HL} as a function of the number of iterations. The result obtained was $\beta_{\text{HL}} = 2.91$, which is slightly higher than the expected lecture value of $\beta = 2.46$. The discrepancy is likely due to the optimization settings and algorithm precision in UQLab.

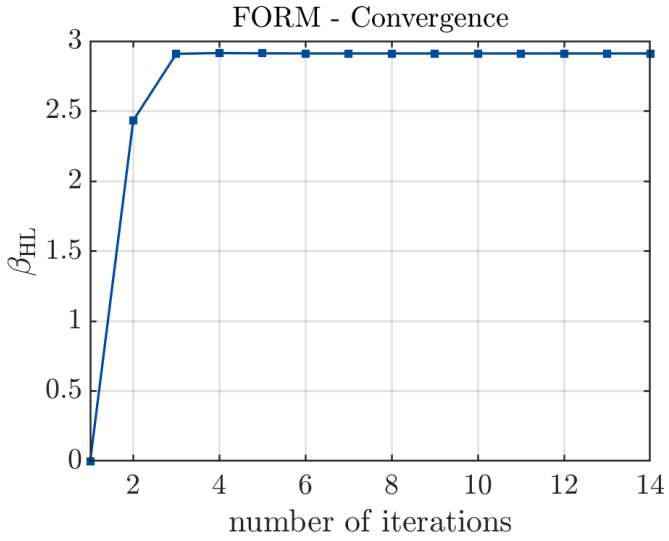

Figure 20: Convergence of β_{HL} in FORM.

Monte Carlo Simulation (MCS): The reliability index β_{MC} and the probability of failure p_f converged smoothly as the number of samples increased. Figures 21 and 22 illustrate these convergence behaviors. The final value for the failure probability was $p_f = 2.47 \times 10^{-3}$, which is in close agreement with the lecture results.

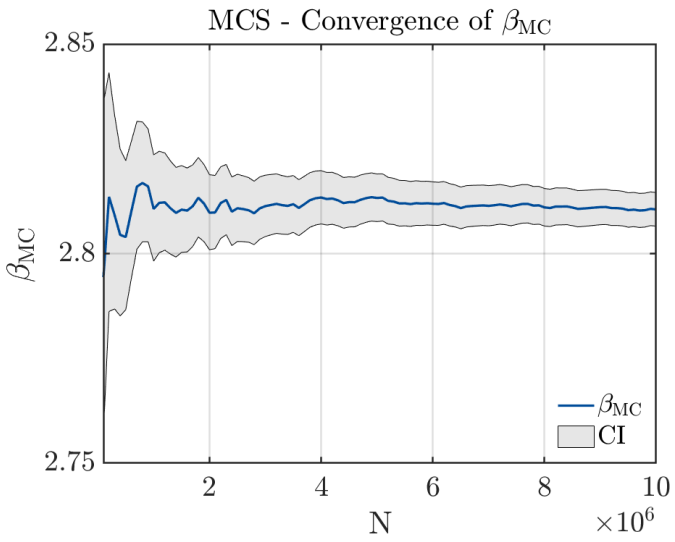

Figure 21: Convergence of β_{MC} using MCS.

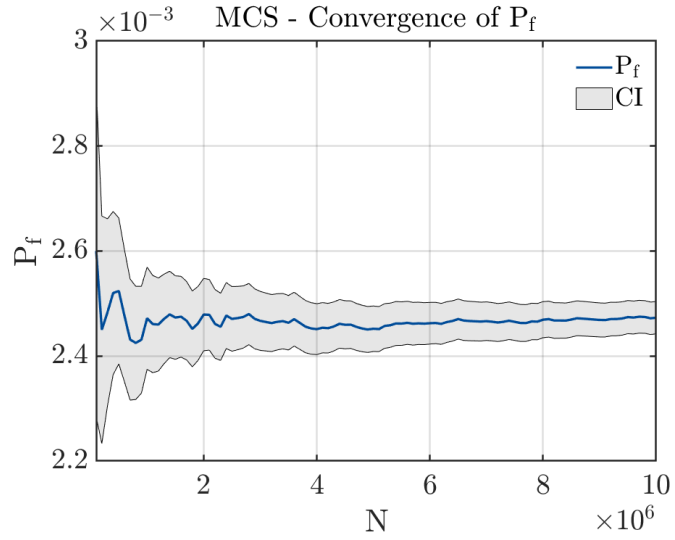

Figure 22: Convergence of p_f using MCS.

Importance Sampling (IS): Figure 23 shows the failure probability p_f converging as the number of samples increases. The final estimate for p_f was 2.48×10^{-3}, which is consistent with the MCS results, and the reliability index was $\beta = 2.81$.

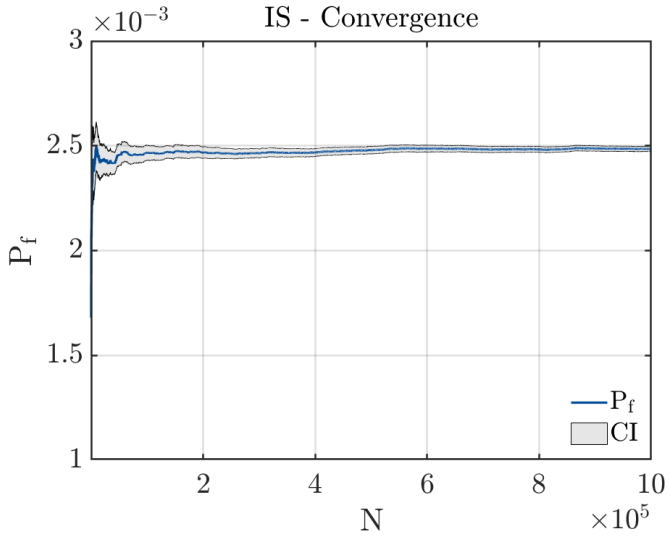

Figure 23: Convergence of p_f using IS.

Adaptive Kriging Monte Carlo Simulation (AK-MCS): The AK-MCS method added a total of 61 samples to achieve convergence, as shown in Figure 24. The final failure probability was $p_f = 2.58 \times 10^{-3}$, which aligns with the other methods but was achieved with fewer model evaluations, demonstrating the efficiency of AK-MCS.

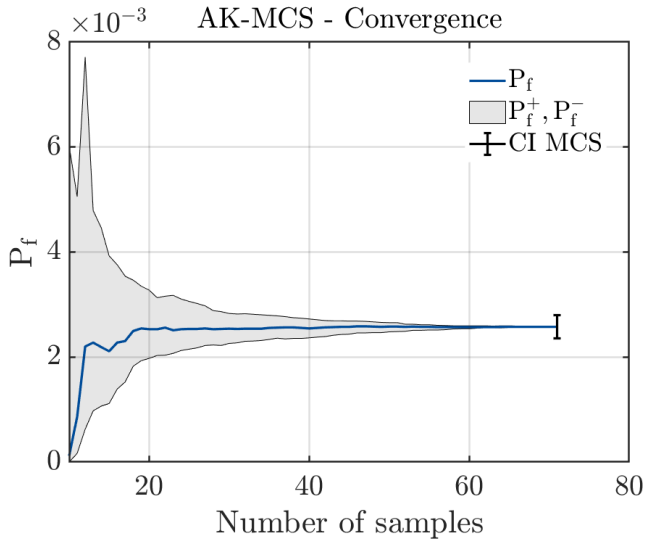

Figure 24: Convergence of p_f using AK-MCS.

5.2 Comparison with the Results obtained in the Lecture

The table below compares the results obtained from UQLab using the FORM, MCS, IS, and AK-MCS methods with the corresponding values from the lecture.

Method	Reliability Index β	Probability of Failure p_f
FORM (UQLab)	2.91	1.80×10^{-3}
Monte Carlo (UQLab)	2.81	2.47×10^{-3}
AK-MCS (UQLab)	2.80	2.58×10^{-3}
Importance Sampling (UQLab)	2.81	2.48×10^{-3}
From Lecture	2.46	6.9×10^{-3}

Table 6: Comparison of results from UQLab and the lecture.

5.3 Interpretation of the Results

The results obtained from UQLab show a noticeable difference when compared to the values obtained in the lecture, particularly for the probability of failure (p_f).

1. Discrepancy in FORM Results: The reliability index β from UQLab using the FORM method is higher than the lecture result. Specifically, the UQLab result for β is 2.91, while the lecture gives $\beta = 2.46$. This difference can be attributed to the precision settings and the optimization algorithm used in UQLab. The UQLab framework tends to use more refined numerical methods for convergence, leading to slightly different but valid results. Moreover, the probability of failure p_f calculated in UQLab is significantly lower, at 1.80×10^{-3}, compared to 6.9×10^{-3} from the lecture. This suggests that the UQLab FORM method finds a more conservative estimate of the system's reliability.

2. MCS, IS, and AK-MCS Results: The results from Monte Carlo Simulation, Importance Sampling, and AK-MCS all show good agreement with each other, with reliability index values around $\beta = 2.80$ and failure probabilities close to 2.50×10^{-3}. These methods are known for their robustness, and the slight variation in results is within acceptable limits of numerical uncertainty. The small difference between these methods and the lecture notes can be attributed to the stochastic nature of these simulations, where the results depend on the sample size and convergence criteria.

3. Conservative Estimates: One clear trend across the UQLab methods is that they provide more conservative estimates of failure probability compared to the lecture values. This difference can be explained by UQLab's higher precision and stricter convergence criteria, which tend to push the reliability index upwards and lower the estimated failure probabilities.

5.4 Final Remarks

Overall, the UQLab framework produced reliable and consistent results across all methods. While there is a noticeable difference between UQLab and the lecture values, the deviations are within a reasonable range and reflect the differences in the computational precision and stopping criteria between the UQLab simulations and the analytical methods discussed in the lectures.

The conservative nature of the UQLab results, particularly in the FORM and MCS methods, demonstrates the robustness of the framework in assessing structural reliability. It is evident that the stochastic methods (MCS, IS, AK-MCS) converge well and provide consistent estimates. UQLab's adaptive sampling methods, such as AK-MCS, offer significant computational savings while maintaining high accuracy, making it a valuable tool for practical applications in structural reliability.

A Code for Task 1

A.1 Task 1.1 Code

```
% example exs_flw1
%-------------------------------------------------------------------
% PURPOSE
%    Analysis of one dimensional heat flow.
%-------------------------------------------------------------------

% REFERENCES
%     P-E Austrell 1994-03-08
%     K-G Olsson 1995-09-28
%     O Dahlblom 2004-09-07
%-------------------------------------------------------------------

echo on

%----- Topology matrix Edof ----------------------------------

 Edof=[1  1  2;
       2  2  3;
       3  3  4;
       4  4  5;
       5  5  6];

%----- Stiffness matrix K and load vector f ---------------------

 K=zeros(6);
 f=zeros(6,1);    f(4)=10

%----- Element properties -----------------------------------

 ep1=[25];        ep2=[24.3];
 ep3=[0.4];       ep4=[17];
 ep5=[7.7];

%----- Element stiffness matrices  ------------------------------

 Ke1=spring1e(ep1);      Ke2=spring1e(ep2);
 Ke3=spring1e(ep3);      Ke4=spring1e(ep4);
 Ke5=spring1e(ep5);


%----- Assemble Ke into K -----------------------------------

 K=assem(Edof(1,:),K,Ke1);       K=assem(Edof(2,:),K,Ke2);
 K=assem(Edof(3,:),K,Ke3);       K=assem(Edof(4,:),K,Ke4);
 K=assem(Edof(5,:),K,Ke5);

%----- Solve the system of equations ----------------------------
```

```
48
49  bc= [1 -25; 6 24]; % Outdoor -25 C , Indoor 24 C
50  [a,r]=solveq(K,f,bc)
51
52 %----- Element flows ------------------------------------------
53
54  ed1=extract_ed(Edof(1,:),a);
55  ed2=extract_ed(Edof(2,:),a);
56  ed3=extract_ed(Edof(3,:),a);
57  ed4=extract_ed(Edof(4,:),a);
58  ed5=extract_ed(Edof(5,:),a);
59
60  q1=spring1s(ep1,ed1)
61  q2=spring1s(ep2,ed2)
62  q3=spring1s(ep3,ed3)
63  q4=spring1s(ep4,ed4)
64  q5=spring1s(ep5,ed5)
65
66 %---------------------------- end -------------------------------
67  echo off
```

Listing 1: Matlab Code for Task 1.1

A.2 Task 1.2 Code

```
1  % Random Field Parameters
2  mu_X = 30.0; % Mean value of material properties (W/K)
3  sigma_X = 3.0; % Standard deviation (W/K)
4  theta = 1.5; % Correlation length
5  % Generate random field (normally distributed values for material
       properties)
6  random_field = normrnd(mu_X, sigma_X, [1, 5]); % 5 sections
7
8  % Assign the generated random field to element properties
9  ep1 = random_field(1);
10 ep2 = random_field(2);
11 ep3 = random_field(3);
12 ep4 = random_field(4);
13 ep5 = random_field(5);
14
15 % Reuse of the rest of the original code
16 %----- Topology matrix Edof -----------------------------------
17 Edof = [1 1 2;
18          2 2 3;
19          3 3 4;
20          4 4 5;
21          5 5 6];
22
23 %----- Stiffness matrix K and load vector f ---------------------
24 K = zeros(6);
25 f = zeros(6,1); f(4) = 10;
```

```
26
27 %----- Element stiffness matrices (based on random properties)
       -----------------------------
28 Ke1 = spring1e(ep1); Ke2 = spring1e(ep2);
29 Ke3 = spring1e(ep3); Ke4 = spring1e(ep4);
30 Ke5 = spring1e(ep5);
31
32 %----- Assemble Ke into K ----------------------------------------
33 K = assem(Edof(1,:), K, Ke1); K = assem(Edof(2,:), K, Ke2);
34 K = assem(Edof(3,:), K, Ke3); K = assem(Edof(4,:), K, Ke4);
35 K = assem(Edof(5,:), K, Ke5);
36
37 %----- Solve the system of equations ---------------------------
38 bc = [1 -25; 6 24]; % Boundary conditions: outdoor -25 C , indoor
       24 C
39 [a, r] = solveq(K, f, bc);
40
41 %----- Element flows ------------------------------------------------
42 ed1 = extract_ed(Edof(1,:), a);
43 ed2 = extract_ed(Edof(2,:), a);
44 ed3 = extract_ed(Edof(3,:), a);
45 ed4 = extract_ed(Edof(4,:), a);
46 ed5 = extract_ed(Edof(5,:), a);
47
48 q1 = spring1s(ep1, ed1);
49 q2 = spring1s(ep2, ed2);
50 q3 = spring1s(ep3, ed3);
51 q4 = spring1s(ep4, ed4);
52 q5 = spring1s(ep5, ed5);
53
54 % Display the results
55 disp('Temperatures at nodes:');
56 disp(a);
57 disp('Heat flows at elements:');
58 disp([q1, q2, q3, q4, q5]);
```

Listing 2: Matlab Code for Task 1.1

A.3 Task 1.3 Code

```
1 % Monte Carlo Simulation parameters
2 num_simulations = 1000; % Number of simulations
3 heat_flows = zeros(num_simulations, 5); % Store heat flow results
      for each simulation
4
5 % Random Field Parameters
6 mu_X = 30.0; % Mean value of material properties (W/K)
7 sigma_X = 3.0; % Standard deviation (W/K)
8
9 for i = 1:num_simulations
10     % Generate random field for each simulation
```

```
    random_field = normrnd(mu_X, sigma_X, [1, 5]);

    % Assign the random field to element properties
    ep1 = random_field(1);
    ep2 = random_field(2);
    ep3 = random_field(3);
    ep4 = random_field(4);
    ep5 = random_field(5);

    % Reuse the same topology matrix and load vector as before
    Edof = [1 1 2; 2 2 3; 3 3 4; 4 4 5; 5 5 6];
    K = zeros(6);
    f = zeros(6,1); f(4) = 10;

    % Element stiffness matrices (based on random properties)
    Ke1 = spring1e(ep1); Ke2 = spring1e(ep2);
    Ke3 = spring1e(ep3); Ke4 = spring1e(ep4);
    Ke5 = spring1e(ep5);

    % Assemble Ke into K
    K = assem(Edof(1,:), K, Ke1); K = assem(Edof(2,:), K, Ke2);
    K = assem(Edof(3,:), K, Ke3); K = assem(Edof(4,:), K, Ke4);
    K = assem(Edof(5,:), K, Ke5);

    % Solve the system of equations
    bc = [1 -25; 6 24]; % Boundary conditions: outdoor -25 C ,
       indoor 24 C
    [a, r] = solveq(K, f, bc);

    % Compute heat flows for the current simulation
    ed1 = extract_ed(Edof(1,:), a);
    ed2 = extract_ed(Edof(2,:), a);
    ed3 = extract_ed(Edof(3,:), a);
    ed4 = extract_ed(Edof(4,:), a);
    ed5 = extract_ed(Edof(5,:), a);

    q1 = spring1s(ep1, ed1);
    q2 = spring1s(ep2, ed2);
    q3 = spring1s(ep3, ed3);
    q4 = spring1s(ep4, ed4);
    q5 = spring1s(ep5, ed5);

    % Store heat flow results
    heat_flows(i, :) = [q1, q2, q3, q4, q5];

    % Print heat flows in the command window
    fprintf('Simulation %d: q1 = %.4f W, q2 = %.4f W, q3 = %.4f W
       , q4 = %.4f W, q5 = %.4f W\n', ...
        i, q1, q2, q3, q4, q5);
end
```

```
% Save heat flow results in a vector
heat_flow_vector = heat_flows;

% Plot histogram of heat flows for each boundary separately
figure;
hold on;
histogram(heat_flows(:, 1), 30, 'FaceColor', 'b', 'DisplayName', 'q1');
histogram(heat_flows(:, 2), 30, 'FaceColor', 'r', 'DisplayName', 'q2');
histogram(heat_flows(:, 3), 30, 'FaceColor', 'g', 'DisplayName', 'q3');
histogram(heat_flows(:, 4), 30, 'FaceColor', 'm', 'DisplayName', 'q4');
histogram(heat_flows(:, 5), 30, 'FaceColor', 'y', 'DisplayName', 'q5');
hold off;

xlabel('Heat Flow (W/m2)');
ylabel('Frequency');
title('Histogram of Heat Flows at Boundaries q1, q2, q3, q4, q5');
legend;

% Save the results
save('heat_flows_vector.mat', 'heat_flow_vector');
```

Listing 3: Matlab Code for Task 1.1

B Code for Task 2

B.1 Task 2.1 Code

```
% Task 2 - Part 1: Adaptation with Random Young's Modulus

echo on

%----- Topology matrix Edof -----------------------------------
Edof = [1 1 2 5 6;
        2 5 6 7 8;
        3 3 4 5 6];

%----- Stiffness matrix K and load vector f ---------------------
K = zeros(8);
f = zeros(8,1); f(6) = -80e3; % Load

%----- Random Young's Modulus for the 3 bars ----------------------
mu_E = 200e9; % Mean Young's Modulus in Pascals (200 GPa)
sigma_E = 10e9; % Standard deviation in Pascals (10 GPa)
```

```
theta = 10; % Correlation length (not used directly here for
    simplicity)

% Generate random Young's modulus for each bar
E_random = normrnd(mu_E, sigma_E, [1, 3]); % Random E values for
    3 bars

A1 = 6.0e-4; A2 = 3.0e-4; A3 = 10.0e-4; % Cross-sectional areas
    of the bars

% Assign element properties with random Young's modulus
ep1 = [E_random(1) A1];
ep2 = [E_random(2) A2];
ep3 = [E_random(3) A3];

%----- Element coordinates --------------------------------------
ex1 = [0 1.6]; ey1 = [0 0]; % Coordinates for element 1
ex2 = [1.6 1.6]; ey2 = [0 1.2]; % Coordinates for element 2
ex3 = [0 1.6]; ey3 = [1.2 0]; % Coordinates for element 3

%----- Element stiffness matrices -------------------------------
Ke1 = bar2e(ex1, ey1, ep1);
Ke2 = bar2e(ex2, ey2, ep2);
Ke3 = bar2e(ex3, ey3, ep3);

%----- Assemble Ke into K ---------------------------------------
K = assem(Edof(1,:), K, Ke1);
K = assem(Edof(2,:), K, Ke2);
K = assem(Edof(3,:), K, Ke3);

%----- Solve the system of equations ----------------------------
bc = [1 0; 2 0; 3 0; 4 0; 7 0; 8 0]; % Boundary conditions (fixed
    supports)
[a, r] = solveq(K, f, bc);

%----- Element forces -------------------------------------------
ed1 = extract_ed(Edof(1,:), a);
N1 = bar2s(ex1, ey1, ep1, ed1);
ed2 = extract_ed(Edof(2,:), a);
N2 = bar2s(ex2, ey2, ep2, ed2);
ed3 = extract_ed(Edof(3,:), a);
N3 = bar2s(ex3, ey3, ep3, ed3);

%----- Display random Young's modulus and forces in the command
    window -----
disp('Random Young''s Modulus values (GPa):');
disp(E_random / 1e9); % Convert to GPa for display

% Display only the axial force in each element (1st entry)
disp('Element Forces (N):');
disp(['N1 = ', num2str(N1(1)), ' N']);
```

```
disp(['N2 = ', num2str(N2(1)), ' N']);
disp(['N3 = ', num2str(N3(1)), ' N']);

%----- Draw deformed truss
    ----------------------------------------
figure(1)
plotpar = [2 1 0];
eldraw2(ex1, ey1, plotpar);
eldraw2(ex2, ey2, plotpar);
eldraw2(ex3, ey3, plotpar);
sfac = scalfact2(ex1, ey1, ed1, 0.1);
plotpar = [1 2 1];
eldisp2(ex1, ey1, ed1, plotpar, sfac);
eldisp2(ex2, ey2, ed2, plotpar, sfac);
eldisp2(ex3, ey3, ed3, plotpar, sfac);
axis([-0.4 2.0 -0.4 1.4]);
scalgraph2(sfac, [1e-3 0 -0.3]);
title('Displacements')

%----- Draw normal force diagram ---------------------------------
figure(2)
plotpar = [2 1];
sfac = scalfact2(ex1, ey1, N2(:,1), 0.1);
secforce2(ex1, ey1, N1(:,1), plotpar, sfac);
secforce2(ex2, ey2, N2(:,1), plotpar, sfac);
secforce2(ex3, ey3, N3(:,1), plotpar, sfac);
axis([-0.4 2.0 -0.4 1.4]);
scalgraph2(sfac, [5e4 0 -0.3]);
title('Normal force')

echo off
```

Listing 4: Matlab Code for Task 2.1

B.2 Task 2.2 Code

```
% Task 2 - Part 2: Monte Carlo Simulation for Displacement
    Exceeding 1.2 mm

% Parameters
num_simulations = 1000; % Number of Monte Carlo simulations
mu_E = 200e9; % Mean Young's Modulus in Pascals (200 GPa)
sigma_E = 10e9; % Standard deviation in Pascals (10 GPa)

f = zeros(8,1); f(6) = -80e3; % Load
bc = [1 0; 2 0; 3 0; 4 0; 7 0; 8 0]; % Boundary conditions (fixed
    supports)

ex1 = [0 1.6]; ey1 = [0 0]; % Coordinates for element 1
ex2 = [1.6 1.6]; ey2 = [0 1.2]; % Coordinates for element 2
ex3 = [0 1.6]; ey3 = [1.2 0]; % Coordinates for element 3
```

```
A1 = 6.0e-4; A2 = 3.0e-4; A3 = 10.0e-4; % Cross-sectional areas of the bars

% Define the topology matrix Edof
Edof = [1 1 2 5 6; % Element 1 connects nodes 1 and 2 to 5 and 6
        2 5 6 7 8; % Element 2 connects nodes 5 and 6 to 7 and 8
        3 3 4 5 6]; % Element 3 connects nodes 3 and 4 to 5 and 6

failures = 0; % Initialize failure counter
displacements = zeros(num_simulations, 1); % Store displacements

for i = 1:num_simulations
    % Generate random Young's modulus for each bar
    E_random = normrnd(mu_E, sigma_E, [1, 3]);

    % Assign element properties with random Young's modulus
    ep1 = [E_random(1) A1];
    ep2 = [E_random(2) A2];
    ep3 = [E_random(3) A3];

    % Stiffness matrix K
    K = zeros(8,8);
    Ke1 = bar2e(ex1, ey1, ep1);
    Ke2 = bar2e(ex2, ey2, ep2);
    Ke3 = bar2e(ex3, ey3, ep3);
    K = assem(Edof(1,:), K, Ke1);
    K = assem(Edof(2,:), K, Ke2);
    K = assem(Edof(3,:), K, Ke3);

    % Solve for displacements
    [a, ~] = solveq(K, f, bc);

    % Record vertical displacement at node 6 (corresponding to f(6) load)
    vertical_displacement = abs(a(6));
    displacements(i) = vertical_displacement;

    % Check if the displacement exceeds 1.2 mm
    if vertical_displacement > 1.2e-3
        failures = failures + 1;
    end
end

% Calculate the probability of failure
probability_of_failure = failures / num_simulations;

% Reliability Index calculation
reliability_index = -norminv(probability_of_failure);

% Display results
```

```
fprintf('Probability of failure: %.4f\n', probability_of_failure)
    ;
fprintf('Reliability index: %.4f\n', reliability_index);
```

Listing 5: Matlab Code for Task 2.2

B.3 Task 2.3 Code

```
% Task 2 - Part 3: Smarter Approach with Sigma Values starting
    from 2.8 GPa

% Initial parameters
target_beta = 3.0; % Desired reliability index
num_simulations = 3000; % Number of Monte Carlo simulations
mu_E = 200e9; % Mean Young's Modulus in Pascals (200 GPa)

f = zeros(8,1); f(6) = -80e3; % Load
bc = [1 0; 2 0; 3 0; 4 0; 7 0; 8 0]; % Boundary conditions (fixed
    supports)

ex1 = [0 1.6]; ey1 = [0 0]; % Coordinates for element 1
ex2 = [1.6 1.6]; ey2 = [0 1.2]; % Coordinates for element 2
ex3 = [0 1.6]; ey3 = [1.2 0]; % Coordinates for element 3
A1 = 6.0e-4; A2 = 3.0e-4; A3 = 10.0e-4; % Cross-sectional areas
    of the bars

% Define the topology matrix Edof
Edof = [1 1 2 5 6;
        2 5 6 7 8;
        3 3 4 5 6];

% Key standard deviations to check starting from 2.8 GPa
sigma_values = [2.8e9, 3.0e9, 3.2e9, 3.4e9, 3.6e9, 3.8e9, 4.0e9];

% Store results for plotting and interpolation
sigma_results = [];
beta_results = [];

for sigma_E = sigma_values
    failures = 0; % Reset failure counter for each iteration

    for i = 1:num_simulations
        % Generate random Young's modulus for each bar with
            current sigma_E
        E_random = normrnd(mu_E, sigma_E, [1, 3]);

        % Assign element properties with random Young's modulus
        ep1 = [E_random(1) A1];
        ep2 = [E_random(2) A2];
        ep3 = [E_random(3) A3];

```

```
        % Stiffness matrix K
        K = zeros(8,8);
        Ke1 = bar2e(ex1, ey1, ep1);
        Ke2 = bar2e(ex2, ey2, ep2);
        Ke3 = bar2e(ex3, ey3, ep3);
        K = assem(Edof(1,:), K, Ke1);
        K = assem(Edof(2,:), K, Ke2);
        K = assem(Edof(3,:), K, Ke3);

        % Solve for displacements
        [a, ~] = solveq(K, f, bc);

        % Record vertical displacement at node 6 (corresponding
           to f(6) load)
        vertical_displacement = abs(a(6));

        % Check if the displacement exceeds 1.2 mm
        if vertical_displacement > 1.2e-3
            failures = failures + 1;
        end
    end

    % Calculate the probability of failure
    probability_of_failure = failures / num_simulations;

    % Reliability Index calculation
    if probability_of_failure == 0
        current_beta = Inf; % Avoid division by zero
    else
        current_beta = -norminv(probability_of_failure);
    end

    % Store the results for later interpolation
    sigma_results = [sigma_results, sigma_E];
    beta_results = [beta_results, current_beta];

    % Display the current results
    fprintf('Sigma: %.2f GPa, Probability of failure: %.4f,
       Reliability index: %.4f\n', ...
        sigma_E / 1e9, probability_of_failure, current_beta);
end

% Remove infinite values before interpolation
finite_indices = isfinite(beta_results);
sigma_finite = sigma_results(finite_indices);
beta_finite = beta_results(finite_indices);

% Remove duplicates to make sure beta values are unique
[beta_finite, unique_idx] = unique(beta_finite);
sigma_finite = sigma_finite(unique_idx);

```

```
% Plot the results for interpolation
figure;
plot(sigma_finite / 1e9, beta_finite, '-o');
xlabel('Sigma (GPa)');
ylabel('Reliability Index (Beta)');
title('Reliability Index vs. Sigma');
grid on;

% Interpolate to find sigma for Beta = 3.0
sigma_interp = interp1(beta_finite, sigma_finite, 3.0, 'linear');
fprintf('Interpolated sigma for reliability index = 3.0: %.2f GPa
    \n', sigma_interp / 1e9);
```

Listing 6: Matlab Code for Task 2.3

C Code for Task 3

C.1 Task 3.1 Code

```
% Task 3 - Part 1: Calculate Random Fields for Frame Elements
    Divided into Two

% Parameters for the random field (E-modulus)
mu_X = 210e9; % Mean E-modulus in Pascals (210 GPa)
sigma_X = 15e9; % Standard deviation in Pascals (15 GPa)
theta = 6; % Correlation length (used for spatial correlation)

% Number of original elements (each will be divided into two)
num_elements = 3; % Three elements in the original structure
num_sub_elements = 2; % Each element is divided into two sub-
    elements

% Cross-sectional areas and moments of inertia (from the original
    code)
A1 = 2e-3; % Cross-sectional area for element 1 and 2
A2 = 6e-3; % Cross-sectional area for element 3
I1 = 1.6e-5; % Moment of inertia for element 1 and 2
I2 = 5.4e-5; % Moment of inertia for element 3

% Topology matrix Edof (from exs_beam2)
Edof = [1 4 5 6 1 2 3;
        2 7 8 9 10 11 12;
        3 4 5 6 7 8 9];

% Initialize stiffness matrix K and load vector f
K = zeros(12); % 12 DOFs in total for the system
f = zeros(12,1); % Load vector
f(4) = 2e+3; % Apply load at node 4 (as in the original code)

% Generate random fields for each sub-element's E-modulus
```

```
E_random = zeros(num_elements * num_sub_elements, 1);

for i = 1:num_elements
    % Generate E-modulus for the first sub-element
    E1 = normrnd(mu_X, sigma_X);

    % Generate E-modulus for the second sub-element, correlated with the first
    rho = exp(-1/theta); % Correlation coefficient based on theta
    E2 = E1 + rho * normrnd(0, sigma_X); % Correlated value for second sub-element

    % Store the values for both sub-elements
    E_random((i-1)*2 + 1) = E1;
    E_random((i-1)*2 + 2) = E2;
end

% Display the random E-modulus values for the sub-elements
fprintf('Random E-modulus values (GPa) for the frame elements divided into two:\n');
for i = 1:length(E_random)
    fprintf('Element %d: E = %.2f GPa\n', i, E_random(i) / 1e9);
end

% Element properties for the sub-elements (using random E-modulus values)
% Element 1 divided into two sub-elements
ep1_1 = [E_random(1), A1, I1]; % First sub-element of element 1
ep1_2 = [E_random(2), A1, I1]; % Second sub-element of element 1

% Element 2 divided into two sub-elements
ep2_1 = [E_random(3), A1, I1]; % First sub-element of element 2
ep2_2 = [E_random(4), A1, I1]; % Second sub-element of element 2

% Element 3 divided into two sub-elements
ep3_1 = [E_random(5), A2, I2]; % First sub-element of element 3
ep3_2 = [E_random(6), A2, I2]; % Second sub-element of element 3

% Element coordinates (before division)
ex1 = [0 0]; ey1 = [4 0]; % Element 1 coordinates
ex2 = [6 6]; ey2 = [4 0]; % Element 2 coordinates
ex3 = [0 6]; ey3 = [4 4]; % Element 3 coordinates

% Now divide the elements into two sub-elements:
% Sub-element 1
ex1_1 = [0 0]; ey1_1 = [4 2]; % First sub-element of element 1
ex1_2 = [0 0]; ey1_2 = [2 0]; % Second sub-element of element 1

ex2_1 = [6 6]; ey2_1 = [4 2]; % First sub-element of element 2
ex2_2 = [6 6]; ey2_2 = [2 0]; % Second sub-element of element 2

```

```
ex3_1 = [0 3]; ey3_1 = [4 4]; % First sub-element of element 3
ex3_2 = [3 6]; ey3_2 = [4 4]; % Second sub-element of element 3

% Calculate the stiffness matrices for the sub-elements
Ke1_1 = beam2e(ex1_1, ey1_1, ep1_1);
Ke1_2 = beam2e(ex1_2, ey1_2, ep1_2);
Ke2_1 = beam2e(ex2_1, ey2_1, ep2_1);
Ke2_2 = beam2e(ex2_2, ey2_2, ep2_2);
Ke3_1 = beam2e(ex3_1, ey3_1, ep3_1);
Ke3_2 = beam2e(ex3_2, ey3_2, ep3_2);

% Assembly of sub-elements into global stiffness matrix (using
    Edof)
K = assem(Edof(1,:), K, Ke1_1);
K = assem(Edof(1,:), K, Ke1_2);
K = assem(Edof(2,:), K, Ke2_1);
K = assem(Edof(2,:), K, Ke2_2);
K = assem(Edof(3,:), K, Ke3_1);
K = assem(Edof(3,:), K, Ke3_2);

% Boundary conditions
bc = [1 0; 2 0; 3 0; 10 0; 11 0]; % Fixed boundary conditions

% Solve the system of equations
[a, r] = solveq(K, f, bc);

% Display the displacements and reactions
fprintf('Displacements at nodes:\n');
disp(a);

fprintf('Reactions at constrained nodes:\n');
disp(r);
```

Listing 7: Matlab Code for Task 3.1

C.2 Task 3.2 Code

```
% Task 3 - Part 2: Monte Carlo Analysis of Horizontal
    Displacements
% Based on "exs_beam2" and adjusted for random material
    properties

% Parameters for Monte Carlo Simulation
num_simulations = 1000; % Number of Monte Carlo simulations
mu_X = 210e9; % Mean E-modulus in Pascals (210 GPa)
sigma_X = 15e9; % Standard deviation in Pascals (15 GPa)
theta = 6; % Correlation length (used for spatial correlation)

% Number of original elements (each will be divided into two)
num_elements = 3; % Three elements in the original structure
```

```
num_sub_elements = 2; % Each element is divided into two sub-elements

% Cross-sectional areas and moments of inertia (from the original problem)
A1 = 2e-3; % Cross-sectional area for element 1 and 2
A2 = 6e-3; % Cross-sectional area for element 3
I1 = 1.6e-5; % Moment of inertia for element 1 and 2
I2 = 5.4e-5; % Moment of inertia for element 3

% Topology matrix Edof (from exs_beam2)
Edof = [1 4 5 6 1 2 3;
        2 7 8 9 10 11 12;
        3 4 5 6 7 8 9];

% Initialize arrays to store the horizontal displacements
horizontal_displacements = zeros(num_simulations, 1);

% Monte Carlo Simulation Loop
for sim = 1:num_simulations
    % Generate random fields for each sub-element's E-modulus
    E_random = zeros(num_elements * num_sub_elements, 1);

    for i = 1:num_elements
        % Generate E-modulus for the first sub-element
        E1 = normrnd(mu_X, sigma_X);

        % Generate E-modulus for the second sub-element, correlated with the first
        rho = exp(-1/theta); % Correlation coefficient based on theta
        E2 = E1 + rho * normrnd(0, sigma_X); % Correlated value for second sub-element

        % Store the values for both sub-elements
        E_random((i-1)*2 + 1) = E1;
        E_random((i-1)*2 + 2) = E2;
    end

    % Element properties for the sub-elements (using random E-modulus values)
    % Element 1 divided into two sub-elements
    ep1_1 = [E_random(1), A1, I1]; % First sub-element of element 1
    ep1_2 = [E_random(2), A1, I1]; % Second sub-element of element 1

    % Element 2 divided into two sub-elements
    ep2_1 = [E_random(3), A1, I1]; % First sub-element of element 2
```

```
ep2_2 = [E_random(4), A1, I1]; % Second sub-element of element 2

% Element 3 divided into two sub-elements
ep3_1 = [E_random(5), A2, I2]; % First sub-element of element 3
ep3_2 = [E_random(6), A2, I2]; % Second sub-element of element 3

% Element coordinates (before division)
ex1 = [0 0]; ey1 = [4 0]; % Element 1 coordinates
ex2 = [6 6]; ey2 = [4 0]; % Element 2 coordinates
ex3 = [0 6]; ey3 = [4 4]; % Element 3 coordinates

% Divide the elements into two sub-elements:
ex1_1 = [0 0]; ey1_1 = [4 2]; % First sub-element of element 1
ex1_2 = [0 0]; ey1_2 = [2 0]; % Second sub-element of element 1

ex2_1 = [6 6]; ey2_1 = [4 2]; % First sub-element of element 2
ex2_2 = [6 6]; ey2_2 = [2 0]; % Second sub-element of element 2

ex3_1 = [0 3]; ey3_1 = [4 4]; % First sub-element of element 3
ex3_2 = [3 6]; ey3_2 = [4 4]; % Second sub-element of element 3

% Calculate the stiffness matrices for the sub-elements
Ke1_1 = beam2e(ex1_1, ey1_1, ep1_1);
Ke1_2 = beam2e(ex1_2, ey1_2, ep1_2);
Ke2_1 = beam2e(ex2_1, ey2_1, ep2_1);
Ke2_2 = beam2e(ex2_2, ey2_2, ep2_2);
Ke3_1 = beam2e(ex3_1, ey3_1, ep3_1);
Ke3_2 = beam2e(ex3_2, ey3_2, ep3_2);

% Assembly of sub-elements into global stiffness matrix (using Edof)
K = zeros(12); % Initialize global stiffness matrix
f = zeros(12,1); % Load vector
f(4) = 2e+3; % Apply load at node 4 (same as in "exs_beam2")

K = assem(Edof(1,:), K, Ke1_1);
K = assem(Edof(1,:), K, Ke1_2);
K = assem(Edof(2,:), K, Ke2_1);
K = assem(Edof(2,:), K, Ke2_2);
K = assem(Edof(3,:), K, Ke3_1);
K = assem(Edof(3,:), K, Ke3_2);

```

```
    % Define boundary conditions
    bc = [1 0; 2 0; 3 0; 10 0; 11 0]; % Fixed boundary conditions

    % Solve the system of equations
    [a, r] = solveq(K, f, bc);

    % Extract the horizontal displacement at the top nodes
    horizontal_displacements(sim) = a(8); % Horizontal
       displacement at node 8 (top of frame)
end

% Compute statistics of the horizontal displacements
mean_displacement = mean(horizontal_displacements);
std_displacement = std(horizontal_displacements);

% Display results
fprintf('Mean horizontal displacement at the top of the frame:
   %.4e m\n', mean_displacement);
fprintf('Standard deviation of horizontal displacement: %.4e m\n'
   , std_displacement);

% Plot the histogram of the horizontal displacements
figure;
histogram(horizontal_displacements, 30); % 30 bins for the
   histogram
xlabel('Horizontal Displacement at Top of Frame (m)');
ylabel('Frequency');
title('Distribution of Horizontal Displacements at the Top of the
    Frame');
grid on;
```

Listing 8: Matlab Code for Task 3.2

D Code for Task 4

D.1 Task 4.1 (FOSM) Code

```
% First Order Second Moment (FOSM) Method

% Mean and standard deviation of random variables
mu_X1_g1 = 12; sigma_X1_g1 = 5;  % X1 for g1
mu_X2_g1 = 10; sigma_X2_g1 = 9;  % X2 for g1

mu_X1_g2 = 10; sigma_X1_g2 = 3;  % X1 for g2
mu_X2_g2 = 1;  sigma_X2_g2 = 1;  % X2 for g2 (Exponential dist
   has rate 1)

% Mean of the limit state functions
mu_g1 = 3 * mu_X1_g1 - 2 * mu_X2_g1 + 18;  % Mean of g1
mu_g2 = mu_X1_g2^2 - mu_X2_g2^3 + 23;      % Mean of g2
```

```
13
14 % Partial derivatives (sensitivities) for g1 and g2
15 dg1_dX1 = 3;  % dg1/dX1 = 3
16 dg1_dX2 = -2; % dg1/dX2 = -2
17
18 dg2_dX1 = 2 * mu_X1_g2;     % dg2/dX1 = 2 * X1
19 dg2_dX2 = -3 * mu_X2_g2^2; % dg2/dX2 = -3 * X2^2
20
21 % Standard deviation of the limit state functions
22 sigma_g1 = sqrt((dg1_dX1 * sigma_X1_g1)^2 + (dg1_dX2 *
      sigma_X2_g1)^2);  % Standard deviation of g1
23 sigma_g2 = sqrt((dg2_dX1 * sigma_X1_g2)^2 + (dg2_dX2 *
      sigma_X2_g2)^2);  % Standard deviation of g2
24
25 % Reliability index (beta) for g1 and g2
26 beta_g1_FOSM = mu_g1 / sigma_g1;
27 beta_g2_FOSM = mu_g2 / sigma_g2;
28
29 % Probability of failure for g1 and g2 using FOSM
30 pf_g1_FOSM = normcdf(-beta_g1_FOSM);
31 pf_g2_FOSM = normcdf(-beta_g2_FOSM);
32
33 % Display results
34 disp(['FOSM - Reliability index (beta) for g1: ', num2str(
      beta_g1_FOSM)]);
35 disp(['FOSM - Probability of failure for g1: ', num2str(
      pf_g1_FOSM)]);
36
37 disp(['FOSM - Reliability index (beta) for g2: ', num2str(
      beta_g2_FOSM)]);
38 disp(['FOSM - Probability of failure for g2: ', num2str(
      pf_g2_FOSM)]);
```

Listing 9: Matlab Code for Task 4.1 (FOSM)

D.2 Task 4.1 (MCS) Code

```
1  % Monte Carlo Simulation for g1 and g2
2
3  % Number of Monte Carlo simulations
4  N = 20000;
5
6  % Pre-allocate arrays to store limit state function values
7  g1_values = zeros(N, 1);
8  g2_values = zeros(N, 1);
9
10 % Generate random samples for X1 and X2 for g1
11 X1_g1 = normrnd(12, 5, [N, 1]); % X1 ~ N(12, 5)
12 X2_g1 = normrnd(10, 9, [N, 1]); % X2 ~ N(10, 9)
13
14 % Generate random samples for X1 and X2 for g2
```

```
X1_g2 = normrnd(10, 3, [N, 1]); % X1 ~ N(10, 3)
X2_g2 = exprnd(1, [N, 1]);      % X2 ~ Exp(1)

% Evaluate the limit state functions g1 and g2
g1_values = 3 * X1_g1 - 2 * X2_g1 + 18;
g2_values = X1_g2.^2 - X2_g2.^3 + 23;

% Calculate probability of failure for g1 (failures where g1 <= 0)
failures_g1 = sum(g1_values <= 0);
pf_g1 = failures_g1 / N;

% Calculate probability of failure for g2 (failures where g2 <= 0)
failures_g2 = sum(g2_values <= 0);
pf_g2 = failures_g2 / N;

% Calculate reliability index (beta) for g1 and g2
beta_g1 = -norminv(pf_g1);
beta_g2 = -norminv(pf_g2);

% Display the results
disp(['Probability of failure for g1: ', num2str(pf_g1)]);
disp(['Reliability index (beta) for g1: ', num2str(beta_g1)]);

disp(['Probability of failure for g2: ', num2str(pf_g2)]);
disp(['Reliability index (beta) for g2: ', num2str(beta_g2)]);
```

Listing 10: Matlab Code for Task 4.1 (MCS)

D.3 Task 4.2 Code

```
% Monte Carlo Simulation Parameters
N = 10000; % Number of simulations

% Parameters for g1: X1 ~ N(12, 5) and X2 ~ N(10, 9)
mu_X1_g1 = 12;
sigma_X1_g1 = 5;
mu_X2_g1 = 10;
sigma_X2_g1 = 9;

% Parameters for g2: X1 ~ N(10, 3) and X2 ~ Exp(1)
mu_X1_g2 = 10;
sigma_X1_g2 = 3;
lambda_X2_g2 = 1; % X2 follows Exp(1), lambda = 1

% Simulate random variables for g1
X1_g1 = normrnd(mu_X1_g1, sigma_X1_g1, N, 1);
X2_g1 = normrnd(mu_X2_g1, sigma_X2_g1, N, 1);

% Simulate random variables for g2
```

```
X1_g2 = normrnd(mu_X1_g2, sigma_X1_g2, N, 1);
X2_g2 = exprnd(1/lambda_X2_g2, N, 1); % Exponential distribution

% Limit state functions
g1 = 3 * X1_g1 - 2 * X2_g1 + 18;
g2 = X1_g2.^2 - X2_g2.^3 + 23;

% Define safe and failure regions for g1 and g2
safe_g1 = g1 >= 0;
failure_g1 = g1 < 0;

safe_g2 = g2 >= 0;
failure_g2 = g2 < 0;

% Plotting in X-space (Original Space) for g1
figure;
scatter(X1_g1(safe_g1), X2_g1(safe_g1), 10, 'g', 'filled'); hold
    on;
scatter(X1_g1(failure_g1), X2_g1(failure_g1), 10, 'r', 'filled');
xlabel('X1 (g1)');
ylabel('X2 (g1)');
title('X-space: Safe/Failure Region for g1');
legend('Safe', 'Failure');
grid on;

% Plotting in X-space (Original Space) for g2
figure;
scatter(X1_g2(safe_g2), X2_g2(safe_g2), 10, 'g', 'filled'); hold
    on;
scatter(X1_g2(failure_g2), X2_g2(failure_g2), 10, 'r', 'filled');
xlabel('X1 (g2)');
ylabel('X2 (g2)');
title('X-space: Safe/Failure Region for g2');
legend('Safe', 'Failure');
grid on;

% Standardize X1 and X2 (U-space transformation)
U1_g1 = (X1_g1 - mu_X1_g1) / sigma_X1_g1;
U2_g1 = (X2_g1 - mu_X2_g1) / sigma_X2_g1;

U1_g2 = (X1_g2 - mu_X1_g2) / sigma_X1_g2;
U2_g2 = -log(X2_g2); % Exponential to normal space

% Plotting in U-space (Transformed Standard Normal Space) for g1
figure;
scatter(U1_g1(safe_g1), U2_g1(safe_g1), 10, 'g', 'filled'); hold
    on;
scatter(U1_g1(failure_g1), U2_g1(failure_g1), 10, 'r', 'filled');
xlabel('U1 (g1)');
ylabel('U2 (g1)');
title('U-space: Safe/Failure Region for g1');
```

```
legend('Safe', 'Failure');
grid on;

% Plotting in U-space (Transformed Standard Normal Space) for g2
figure;
scatter(U1_g2(safe_g2), U2_g2(safe_g2), 10, 'g', 'filled'); hold
    on;
scatter(U1_g2(failure_g2), U2_g2(failure_g2), 10, 'r', 'filled');
xlabel('U1 (g2)');
ylabel('U2 (g2)');
title('U-space: Safe/Failure Region for g2');
legend('Safe', 'Failure');
grid on;
```

Listing 11: Matlab Code for Task 4.2

D.4 Task 4.3, 4.4 & 4.5 Code for g1

```
% FORM - Hasofer-Lind Rackwitz- Fiessler Iteration
% According to example of C. Bucher,
% Stochastic Simulation Methods and Structural Reliability,
% 2015, T. Lahmer
% Code mofified for the problem

function [pf, beta] = ExampleClass_quadratic_FORM_g1

% Statistical properties of first random variable (X1)
m1 = 12; % Mean
s1 = 5;  % Standard deviation

% Statistical properties of second random variable (X2)
m2 = 10; % Mean
s2 = 9;  % Standard deviation

% Initialize the vector u (standard normal space)
u0 = [0; 0];  % Basic initial guess
u = u0;
uall = u';

tol = 1e-7;  % Tighter convergence tolerance
max_iter = 100;  % Maximum number of iterations
damping_factor = 0.02;  % Small damping factor for stability

% Iterate to find the Most Probable Point (MPP)
for i = 1:max_iter
    g = grad_g1(u, s1, s2, m1, m2);  % Gradient of g1
    gtg = g' * g;
    gtu = g' * u;
    f = limitState_g1(u, s1, s2, m1, m2);  % Limit-state function
        for g1
    lambda_update = damping_factor * (f - gtu) / gtg;
```

```
    v = g * (-lambda_update) - u;

    % Update u and check convergence
    u_new = u + v;
    uall = [uall; u_new'];

    % Print intermediate u and beta for observation
    fprintf('Iteration %d: u = [%f, %f], beta = %f\n', i, u_new
        (1), u_new(2), sqrt(u_new' * u_new));

    if norm(u_new - u) < tol
        break;
    end
    u = u_new;

    % Update beta (Safety index)
    beta = sqrt(u' * u);
end

% Print results for 4.3
fprintf('\nSafety Index beta = %f ', beta);
fprintf('\nProbability of failure pf = %f \n', normcdf(-beta));

% Calculate probability of failure
pf = normcdf(-beta);

% Backtransformation into original space
x1s = s1 * u(1) + m1;  % X1 in original space
x2s = s2 * u(2) + m2;  % X2 in original space

% Print MPFP
fprintf('\nMPFP in U-space: u* = [%f, %f]\n', u(1), u(2));
fprintf('MPFP in X-space: X* = [%f, %f]\n', x1s, x2s);

% --- Plot 1: Limit State Function in U-space with Separation (
    for 4.4 and 4.5) ---

figure;

% Create a grid of points in U-space
[u1_grid, u2_grid] = meshgrid(linspace(-3, 3, 100), linspace(-3,
    3, 100));

% Compute the limit state function g1 over the grid
g_values = zeros(size(u1_grid));
for i = 1:size(u1_grid, 1)
    for j = 1:size(u1_grid, 2)
        g_values(i, j) = limitState_g1([u1_grid(i, j); u2_grid(i,
            j)], s1, s2, m1, m2);
    end
end
```

```
80
81  % Plot the limit state boundary (g = 0) and fill below/above the
        limit state
82  contourf(u1_grid, u2_grid, g_values, [-100 0], 'LineColor', 'none
        '); % Failure region (red)
83  hold on;
84  contour(u1_grid, u2_grid, g_values, [0 0], 'LineColor', 'black',
        'LineWidth', 1.5); % Limit state boundary (g = 0)
85  colormap([1 0.8 0.8; 0.8 1 0.8]); % Red for failure region, green
         for safe region
86  colorbar;
87  hold on;
88
89  % Plot the iteration path, MPFP, and Pf point
90  h1 = plot(uall(:,1), uall(:,2), 'b-', 'LineWidth', 1.5); % Path
        to MPP in U-space
91  h2 = scatter(u(1), u(2), 50, 'filled', 'r'); % MPFP in U-space
92  h3 = scatter(1, pf, 50, 'filled', 'g'); % Pf as a green point in
        U-space
93
94  % Add labels and title
95  xlabel('U1');
96  ylabel('U2');
97  title('Limit State Function g1 in U-space with P_f');
98
99  % Add legend
100 legend([h2, h3], {'MPFP (Most Probable Failure Point)', 'Pf Point
        '}, 'Location', 'best');
101
102 grid on;
103
104 % --- Plot MPFP in X-space (for 4.5) ---
105 figure;
106 scatter(x1s, x2s, 50, 'filled', 'r');
107 xlabel('X_1');
108 ylabel('X_2');
109 title('MPFP in X-space for g_1');
110 grid on;
111
112 end
113
114 % Gradient of limit-state function g1
115 function grad_g1 = grad_g1(u, s1, s2, m1, m2)
116     x1 = s1 * u(1) + m1;
117     x2 = s2 * u(2) + m2;
118     grad_g1 = [3; -2];  % Derivatives of g1
119 end
120
121 % Limit-state function g1 = 3*X1 - 2*X2 + 18
122 function ls_g1 = limitState_g1(u, s1, s2, m1, m2)
123     x1 = s1 * u(1) + m1;
```

```
    x2 = s2 * u(2) + m2;
    ls_g1 = 3 * x1 - 2 * x2 + 18;  % Limit-state function g1
end
```

Listing 12: Matlab Code for Task 4.3, 4.4 & 4.5 Code for g1

D.5 Task 4.3, 4.4 & 4.5 Code for g2

```
% FORM - Hasofer-Lind Rackwitz- Fiessler Iteration
% According to example of C. Bucher,
% Stochastic Simulation Methods and Structural Reliability,
% 2015, T. Lahmer
% Code mofified for the problem

function [pf, beta] = ExampleClass_quadratic_FORM_g2

% Statistical properties of first random variable (X1)
m1 = 10; % Mean
s1 = 3;  % Standard deviation

% Statistical properties of second random variable (X2)
lambda2 = 1;  % Exponential distribution parameter for X2

% Initialize the vector u (standard normal space)
u0 = [0; 0];  % Basic initial guess
u = u0;
uall = u';

tol = 1e-7;  % Tighter convergence tolerance
max_iter = 100;  % Maximum number of iterations
damping_factor = 0.02;  % Small damping factor for stability

% Iterate to find the Most Probable Point (MPP)
for i = 1:max_iter
    g = grad_g2(u, s1, lambda2, m1);  % Gradient of g2
    gtg = g' * g;
    gtu = g' * u;
    f = limitState_g2(u, s1, lambda2, m1);  % Limit-state
       function for g2
    lambda_update = damping_factor * (f - gtu) / gtg;
    v = g * (-lambda_update) - u;

    % Update u and check convergence
    u_new = u + v;
    uall = [uall; u_new'];

    % Print intermediate u and beta for observation
    fprintf('Iteration %d: u = [%f, %f], beta = %f\n', i, u_new
       (1), u_new(2), sqrt(u_new' * u_new));

    if norm(u_new - u) < tol
```

```
        break;
    end
    u = u_new;

    % Update beta (Safety index)
    beta = sqrt(u' * u);
end

% Print results for 4.3
fprintf('\nSafety Index beta = %f ', beta);
fprintf('\nProbability of failure pf = %f \n', normcdf(-beta));

% Calculate probability of failure
pf = normcdf(-beta);

% Backtransformation into original space
x1s = s1 * u(1) + m1;  % X1 in original space
x2s = exp(u(2)) / lambda2;  % X2 (Exponential backtransformation)

% Print MPFP
fprintf('\nMPFP in U-space: u* = [%f, %f]\n', u(1), u(2));
fprintf('MPFP in X-space: X* = [%f, %f]\n', x1s, x2s);

% --- Plot 1: Limit State Function in U-space with Separation (for 4.4 and 4.5) ---

figure;

% Create a grid of points in U-space
[u1_grid, u2_grid] = meshgrid(linspace(-3, 3, 100), linspace(-3, 3, 100));

% Compute the limit state function g2 over the grid
g_values = zeros(size(u1_grid));
for i = 1:size(u1_grid, 1)
    for j = 1:size(u1_grid, 2)
        g_values(i, j) = limitState_g2([u1_grid(i, j); u2_grid(i, j)], s1, lambda2, m1);
    end
end

% Plot the limit state boundary (g = 0) and fill below/above the limit state
contourf(u1_grid, u2_grid, g_values, [-100 0], 'LineColor', 'none'); % Failure region (red)
hold on;
contour(u1_grid, u2_grid, g_values, [0 0], 'LineColor', 'black', 'LineWidth', 1.5); % Limit state boundary (g = 0)
colormap([1 0.8 0.8; 0.8 1 0.8]); % Red for failure region, green for safe region
colorbar;
```

```
hold on;

% Plot the iteration path, MPFP, and Pf point
h1 = plot(uall(:,1), uall(:,2), 'b-', 'LineWidth', 1.5); % Path to MPP in U-space
h2 = scatter(u(1), u(2), 50, 'filled', 'r'); % MPFP in U-space
h3 = scatter(1, pf, 50, 'filled', 'g'); % Pf as a green point in U-space

% Add labels and title
xlabel('U1');
ylabel('U2');
title('Limit State Function g2 in U-space with P_f');

% Add legend with correct handles
legend([h2, h3], {'MPFP (Most Probable Failure Point)', 'Pf Point'}, 'Location', 'best');

grid on;

% --- Plot MPFP in X-space (for 4.5) ---
figure;
scatter(x1s, x2s, 50, 'filled', 'r');
xlabel('X_1');
ylabel('X_2');
title('MPFP in X-space for g_2');
grid on;

end

% Gradient of limit-state function g2
function grad_g2 = grad_g2(u, s1, lambda2, m1)
    x1 = s1 * u(1) + m1;
    x2 = exp(u(2)) / lambda2;
    grad_g2 = [2 * x1; -3 * x2^2];  % Derivatives of g2
end

% Limit-state function g2 = X1^2 - X2^3 + 23
function ls_g2 = limitState_g2(u, s1, lambda2, m1)
    x1 = s1 * u(1) + m1;
    x2 = exp(u(2)) / lambda2;
    ls_g2 = x1^2 - x2^3 + 23;  % Limit-state function g2
end
```

Listing 13: Matlab Code for Task 4.3, 4.4 & 4.5 Code for g2

D.6 Task 4.6 Code for g1

```
% Initialize UQLab
uqlab;

```

```
%% Step 1: Define the limit state function g1 = 3*X1 - 2*X2 + 18

% Define the ModelOpts structure for g1
ModelOpts.mString = '3 * X(:,1) - 2 * X(:,2) + 18';  % g1 function as a string
ModelOpts.isVectorized = true;  % Allow vectorized evaluation
myModel = uq_createModel(ModelOpts);  % Create the model

%% Step 2: Define the probabilistic input model for g1
% X1 ~ N(12, 5), X2 ~ N(10, 9)
InputOpts.Marginals(1).Type = 'Gaussian';  % X1 is Gaussian
InputOpts.Marginals(1).Parameters = [12, 5];  % Mean = 12, Std = 5

InputOpts.Marginals(2).Type = 'Gaussian';  % X2 is Gaussian
InputOpts.Marginals(2).Parameters = [10, 9];  % Mean = 10, Std = 9

myInput = uq_createInput(InputOpts);  % Create the probabilistic input model

%% Step 3: Set up and run the FORM analysis for g1
FORMOpts.Type = 'Reliability';
FORMOpts.Method = 'FORM';
FORMOpts.Model = myModel;  % Link to g1 model
FORMOpts.Input = myInput;  % Link to input model
myFORMAnalysis_g1 = uq_createAnalysis(FORMOpts);  % Run FORM analysis

% Print and display results of FORM analysis
uq_print(myFORMAnalysis_g1);
uq_display(myFORMAnalysis_g1);

%% Step 4: Run Monte Carlo Simulation (MCS) for g1
MCSOpts.Type = 'Reliability';
MCSOpts.Method = 'MCS';  % Monte Carlo Simulation
MCSOpts.Model = myModel;  % Link to g1 model
MCSOpts.Input = myInput;  % Link to input model
MCSOpts.Simulation.MaxSampleSize = 1e5;  % Set a maximum number of samples (e.g., 100,000)
MCSOpts.Simulation.BatchSize = 1e3;  % Set batch size to control how samples are taken

myMCSAnalysis_g1 = uq_createAnalysis(MCSOpts);  % Run MCS analysis

% Print and display results of MCS analysis
uq_print(myMCSAnalysis_g1);
uq_display(myMCSAnalysis_g1);
```

Listing 14: Matlab Code for Task 4.6 Code for g1

D.7 Task 4.6 Code for g2

```
function T4_6_g2
    % Initialize UQLab framework
    uqlab;

    %% 1 - Define the probabilistic input model
    InputOpts.Marginals(1).Type = 'Gaussian';
    InputOpts.Marginals(1).Moments = [10 3];  % Mean and standard
        deviation for X1
    InputOpts.Marginals(2).Type = 'Exponential';
    InputOpts.Marginals(2).Parameters = 1;  % Lambda for
       Exponential distribution (X2 ~ Exp(1))

    myInput = uq_createInput(InputOpts);

    %% 2 - Define the limit state function (g2) inline instead of
        external file
    ModelOpts.mString = 'X(:,1).^2 - X(:,2).^3 + 23';  % The
       limit state function g2
    myModel = uq_createModel(ModelOpts);

    %% 3 - FORM analysis for g2
    FORMOpts_g2.Type = 'Reliability';
    FORMOpts_g2.Method = 'FORM';
    FORMOpts_g2.Input = myInput;
    FORMOpts_g2.Model = myModel;

    % Run FORM analysis
    myFORMAnalysis_g2 = uq_createAnalysis(FORMOpts_g2);

    % Display FORM results
    uq_print(myFORMAnalysis_g2);
    uq_display(myFORMAnalysis_g2);

    %% 4 - Monte Carlo Simulation (MCS) analysis for g2
    MCSOpts_g2.Type = 'Reliability';
    MCSOpts_g2.Method = 'MCS';
    MCSOpts_g2.Simulation.BatchSize = 1e4;
    MCSOpts_g2.Simulation.MaxSampleSize = 1e5;
    MCSOpts_g2.Input = myInput;
    MCSOpts_g2.Model = myModel;

    % Run MCS analysis
    myMCSAnalysis_g2 = uq_createAnalysis(MCSOpts_g2);

    % Display MCS results
    uq_print(myMCSAnalysis_g2);
    uq_display(myMCSAnalysis_g2);
end
```

Listing 15: Matlab Code for Task 4.6 Code for g2

D.8 Task 5 function Code

```
function g = limit_state_function(X)
    P = X(:,1);
    L = X(:,2);
    W = X(:,3);
    T = X(:,4);
    g = W .* T - (P .* L) ./ 4;
end
```

Listing 16: Matlab Code for Task 5 function

D.9 Task 5 Code

```
% Initialize UQLab
uqlab;

% -------------------- Step 1: Define the probabilistic input model ---------------------
Input.Marginals(1).Name = 'P';  % Force in kN
Input.Marginals(1).Type = 'Gaussian';
Input.Marginals(1).Parameters = [10 2];  % Mean = 10, StdDev = 2

Input.Marginals(2).Name = 'L';  % Length in m
Input.Marginals(2).Type = 'Gaussian';
Input.Marginals(2).Parameters = [8 0.1];  % Mean = 8, StdDev = 0.1

Input.Marginals(3).Name = 'W';  % Section modulus in m^3
Input.Marginals(3).Type = 'Gaussian';
Input.Marginals(3).Parameters = [100e-6 2e-5];  % Mean = 100e-6, StdDev = 2e-5

Input.Marginals(4).Name = 'T';  % Yield stress in kN/m^2
Input.Marginals(4).Type = 'Gaussian';
Input.Marginals(4).Parameters = [600e3 105e3];  % Mean = 600e3, StdDev = 105e3

% Create the input object
myInput = uq_createInput(Input);

% -------------------- Step 2: Define the limit-state function ---------------------
ModelOpts.mFile = 'limit_state_function';  % Reference to the m-file function
myModel = uq_createModel(ModelOpts);  % Create the model object
```

```
27
28 % Limit state function in 'limit_state_function.m'
29 % function g = limit_state_function(X)
30 % P = X(:,1);
31 % L = X(:,2);
32 % W = X(:,3);
33 % T = X(:,4);
34 % g = W .* T - (P .* L) ./ 4;
35 % end
36
37 % -------------------- Step 3: Reliability Analysis Methods
   ---------------------
38
39 % ------------- FORM Analysis -------------
40 FORMOpts.Type = 'Reliability';
41 FORMOpts.Method = 'FORM';
42 FORMOpts.Model = myModel;
43 FORMOpts.Input = myInput;
44 FORMOpts.LimitState.Threshold = 0;
45
46 % Ensure correct gradient calculation and improve accuracy
47 FORMOpts.Optim.TolX = 1e-6;
48 FORMOpts.Optim.TolFun = 1e-6;
49 FORMAnalysis = uq_createAnalysis(FORMOpts);
50
51 % Display the FORM results
52 fprintf('FORM Results:\n');
53 uq_print(FORMAnalysis);
54 uq_display(FORMAnalysis);
55
56 % ------------- MCS Analysis -------------
57 MCSOpts.Type = 'Reliability';
58 MCSOpts.Method = 'MCS';
59 MCSOpts.Model = myModel;
60 MCSOpts.Input = myInput;
61 MCSOpts.LimitState.Threshold = 0;
62
63 % Large sample size for better accuracy
64 MCSOpts.Simulation.BatchSize = 1e5;
65 MCSOpts.Simulation.MaxSampleSize = 1e7;
66 MCSAnalysis = uq_createAnalysis(MCSOpts);
67
68 % Display the MCS results
69 fprintf('MCS Results:\n');
70 uq_print(MCSAnalysis);
71 uq_display(MCSAnalysis);
72
73 % ------------- Importance Sampling -------------
74 ISOpts.Type = 'Reliability';
75 ISOpts.Method = 'IS';
76 ISOpts.Model = myModel;
```

```
ISOpts.Input = myInput;
ISOpts.LimitState.Threshold = 0;
ISOpts.Simulation.MaxSampleSize = 1e6;

% Improve convergence
ISAnalysis = uq_createAnalysis(ISOpts);

% Display the Importance Sampling results
fprintf('Importance Sampling Results:\n');
uq_print(ISAnalysis);
uq_display(ISAnalysis);

% ------------- AK-MCS Analysis -------------
AKMCSOpts.Type = 'Reliability';
AKMCSOpts.Method = 'AKMCS';
AKMCSOpts.Model = myModel;
AKMCSOpts.Input = myInput;
AKMCSOpts.LimitState.Threshold = 0;

% Limiting the total number of samples
AKMCSOpts.Simulation.MaxSampleSize = 2e5;  % Limit the total
   sample size to 200,000
AKMCSOpts.StoppingCriterion.TargetCoV = 0.02;  % Adjust the CoV
   for early stopping
AKMCSOpts.StoppingCriterion.MaxAddedSamples = 10;  % Limit added
   samples per iteration

AKMCSAnalysis = uq_createAnalysis(AKMCSOpts);

% Display the AK-MCS results
fprintf('AK-MCS Results:\n');
uq_print(AKMCSAnalysis);
uq_display(AKMCSAnalysis);


% -------------------- End of Code --------------------
```

Listing 17: Matlab Code for Task 5