Sharif University of Technology

A Probabilistic Optimization Approach to Reliability and Safety Assessment of Urban Air Mobility Vehicles

Mohammad Bagheri & Mahdi Bostanabad

Supervisor: Amir H. Khodabakhsh, PhD.

A report submitted in fulfilment of the course
AE788: Reliability Analysis and Risk Assessment

in the

Department of Aerospace Engineering

August 2025

Abstract

This project develops and validates a complete, reproducible workflow for the sizing, reliability assessment, and optimization under uncertainty of a quadrotor tasked with transporting a 2 kg payload over 50 km. We first formulate a physics-based deterministic sizing problem for rotor radius and cruise speed, enforcing disk-loading, blade-loading, and energy-reserve constraints. The resulting limit states are then endowed with uncertainty (lognormal C_{d_0} and σ) to define probabilistic failure boundaries. Reliability is quantified with complementary estimators—FORM/SORM for fast indices and design points, and Directional Sampling (DS) and Importance Sampling (IS) for variance-controlled validation and tail checks—together with an explicit comparison of estimator CoV and discussion of when SORM can underestimate p_f. Building on this, we solve Optimization Under Uncertainty via SORA, decoupling reliability and design: at each iteration, we map reliability targets to equivalent points in standard normal space and solve a deterministic subproblem. This produces a design that meets $p_{f,\text{struct}} \leq 10^{-4}$ and $p_{f,\text{energy}} \leq 10^{-3}$ while keeping mass growth modest (e.g., $r = 0.150$ m with $V_\infty \approx 77$ m/s and an additional energy reserve that lifts mass from ~ 5.0 kg deterministically to ~ 6.6 kg at reliability). Finally, we perform epistemic UQ with Sobol indices over drag-mean bias and Weibull fatigue shape, showing drag bias dominates mission-risk variance with negligible interactions—actionable guidance to prioritize drag data over redesign. Collectively, the study demonstrates a transparent, auditable pathway from baseline sizing to reliability-constrained design and epistemic sensitivity for UAM vehicles.

Keywords. *Urban Air Mobility, Reliability Analysis, Monte Carlo Simulation, Sequential Optimization and Reliability Assessment, Optimization Under Uncertainty, Epistemic Uncertainty Quantification.*

Reproducibility. All scripts and data, alongside the report file, are available here

Contents

List of Figures

List of Tables

Chapter 1

Problem Statement

Urban Air Mobility (UAM) promises to revolutionize transportation, but public acceptance hinges on demonstrated safety. In 2023, a Matternet medical delivery drone crashed due to motor failure mid-flight, a $2M liability event. As engineers, you must balance innovation with existential risk. Your task is to perform reliability assessment and optimization on a quadcopter carrying 5kg medical payloads over the city. This directly addresses ASTM F3322-18 certification requirements, where failure probabilities must be proven below $10 - 7$ per flight hour. You are pioneering the methodology that will certify autonomous urban flight. A flawed reliability model does not just risk product recall-it jeopardizes an entire industry.

You would like to design and validate your drone to minimize its takeoff weight, as this reduction in weight leads to lower fleet acquisition costs and operating costs. Two key benefits arise from this minimization: (1) smaller vehicles are likely to be less expensive to manufacture, and (2) lighter vehicles tend to consume less energy. However, while minimizing weight is a priority, it is crucial not to compromise the safety of the flight. Specifically, you want to avoid the drone running out of power mid-flight or losing a blade in response to even moderate winds, ensuring a stable and reliable operation. You and your team have parameterized the mission in terms of range R, and total weight W. Figure 1.1 represents the simple mission profile that you are considering in this phase of your analysis. The empty weight of the drone is broken down into:

$$W_{\text{total}} = W_{\text{payload}} + W_{\text{battery}} + W_{\text{empty}} \tag{1.1}$$

The empty weight can be further broken down into its constituent components, including the motor, the electric speed controller (ESC), the rotors, and the frame.

$$W_{\text{empty}} = W_{\text{motors}} + W_{\text{ESC}} + W_{\text{rotors}} + W_{\text{frame}} \tag{1.2}$$

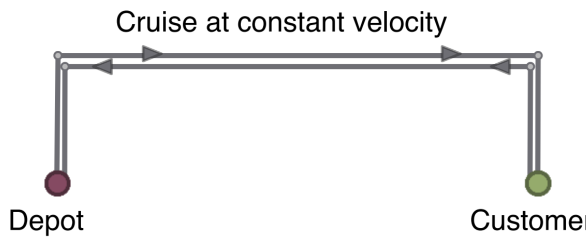

Figure 1.1: *Mission profile*

Let P and r represent the power of the installed motors and the rotor radius, respectively. Assuming SI units, database regressions and empirical expressions for this case give:

$$W_{\text{motors}} = (2.506 \times 10^{-4}) \cdot P \tag{1.3}$$
$$W_{\text{ESC}} = (3.594 \times 10^{-4}) \cdot P \tag{1.4}$$
$$W_{\text{rotors}} = 0.784r^2 - 0.0403r \tag{1.5}$$
$$W_{\text{frame}} = 0.5 + 0.2W_{\text{total}} \tag{1.6}$$

The battery weight is calculated based on the battery's specific energy and the total energy required for the mission. To estimate this energy requirement, the simplified mission profile consisting of three phases (vertical takeoff, cruise, and landing) is considered. Additionally, the power consumption during vertical climb and descent can be approximated to be equivalent to the power consumption during hover, on average. The total energy required for the mission can then be determined by considering these factors.

$$E_{\text{req}} = P_{\text{hover}} \cdot 4t_{\text{hover}} + P_{\text{cruise}}\frac{T}{V_\infty} \tag{1.7}$$

where V_∞ is the cruise speed, P_{hover} and P_{cruise} are the power consumption in hover and cruise, respectively. The first term is the energy for takeoff and landing, and the second is the energy for cruise. The hovering time t_{hover} is assumed to be 60 seconds. P_{hover} and P_{cruise} are also assumed to be constant. Once the required energy is computed, the battery weight is given by

$$W_{\text{battery}} = \frac{E_{\text{req}}}{0.85\rho_b} \tag{1.8}$$

The specific electric energy of the battery ρ_b is assumed to be constant and equal to 158 Wh/kg. Based on the momentum theory, the shaft power required by each rotor is

$$P_{\text{hover}} = \frac{1}{0.75}\frac{(W_{\text{total}}/4)^{1.5}}{\sqrt{2\rho A}} \tag{1.9}$$

$$P_{\text{cruise}} \approx \frac{\sigma C_{d_0}}{8}\left(1 + 4.65\left(\frac{V_\infty}{\Omega r}\right)^2\right)\rho A\Omega^3 r^3 \tag{1.10}$$

where ρ is the air density; A is the rotor disk area; $C_{d_0} \approx 0.012$ is the airfoil zero-lift drag coefficient; $\sigma \approx 0.13$ is the rotor solidity, and Ω is the rotor speed. The thrust coefficient is defined as

$$C_T = \frac{T}{\rho A\Omega^2 r^2} \tag{1.11}$$

The trim condition in cruise determines the thrust as

$$T = \frac{W_{\text{total}}}{4} \tag{1.12}$$

Your drone mission requires transporting a payload of at least 2 kg over a distance of 50 km. To complete the design at this phase, you must select the optimal rotor radius and the cruise speed. As previously explained, minimizing the total weight is a key objective. However, there are several constraints that must be satisfied. The composite material chosen for the blades imposes two constraints: the disk loading and the blade loading must not exceed the predefined values, i.e., $T/A \leq 250\,\mathrm{N/m^2}$, $C_T/\sigma \leq 0.14$.

Furthermore, the battery technology you intend to use has a specific limitation: allowing the charge to drop below 30% reduces the battery's lifespan. Therefore, a safety margin of at least 30% must be considered for the battery charge to ensure its longevity. Unfortunately, there are uncertainties associated with the drag coefficient and the rotor solidity at this stage of the design. The coefficient of variation for the drag coefficient has been estimated to be 20%, while the coefficient of variation for the rotor solidity is approximately 12%. To account for these uncertainties, design your drone aiming to achieve a 90% probability of mission success.

Roadmap

1.1 Deterministic Design

Formulate the given problem as an optimization problem and solve the problem by assuming that there is no uncertainty in the drag coefficient and the rotor solidity.

$$\begin{aligned} \min_{r,V_\infty} \quad & m = f(r, V_\infty) \\ \text{s.t.} \quad & T/A \geq 250\,\mathrm{N/m^2} \\ & C_T/\sigma \leq 0.14 \end{aligned}$$

1.2 Probabilistic Modeling of Failure Boundaries

The objective is to establish physics-based limit states with measurable uncertainty. Define two limit states governing system collapse:

- *Structural failure*: $g_1(\mathbf{x})$ = The disk loading **and** $g_2(\mathbf{x})$ = The blade loading
- *Energy depletion*: $g_3(\mathbf{x})$ = Battery capacity − (Flight energy + 30% safety margin)

1.3 Reliability Analysis

Compute FORM/SORM reliability indices (β) for the limit states. Validate with *Directional Sampling*. Then, implement *Importance Sampling* centered on the FORM design point.

Critical analysis: Compare the coefficient of variation (COV) across methods—why might SORM underestimate p_f?

1.4 Optimization Under Uncertainty

Formulate and solve the following design optimization problem

$$\begin{aligned} \min \quad & m = f(\text{prop diameter}, \ \text{battery cells}) \\ \text{s.t.} \quad & p_{f,\text{struct}} \leq 10^{-4} \\ & p_{f,\text{power}} \leq 10^{-3} \end{aligned}$$

Use the SORA algorithm (Figure 1.2) to decouple reliability loops from design iterations.

Critical reflection: How do your optimal parameters change if the uncertainty in drag increases 20%?

Initialize:

- Set initial design variables $\mathbf{d}^0$.
- Set convergence tolerances for the objective function and constraints.
- Set iteration counter $k = 0$.

Repeat until convergence:

1. **Deterministic Optimization:** Solve the deterministic optimization problem using the current reliability constraint approximations:

$$\begin{aligned} \text{Minimize} \ &: f(\mathbf{d}) \\ \text{Subject to} \ &: g_i(\mathbf{d}, u_i^\star) \geq 0, \qquad \forall i \end{aligned}$$

   where $u_i^\star$ is the Most Probable Point (MPP) in the standard normal space, obtained from the previous reliability analysis or initialized as the mean for the first cycle. Then, obtain the optimal design $\mathbf{d}_k^\star$.

2. **Reliability Analysis:** For each probabilistic constraint i, perform reliability analysis at $\mathbf{d}_k^\star$ to find the MPP $u_i^\star$ corresponding to the target reliability level (e.g., using FORM or other methods). Then, transform $u_i^\star$ back to the original variable space as needed.
3. **Update Constraints:** Update the deterministic optimization constraints for the next cycle using the new MPPs.
4. **Check Convergence:** Stop if (1) The absolute or relative change in the objective function is less than the tolerance and all constraints are satisfied or, (2) the maximum number of iterations is reached. Otherwise, set $k = k + 1$ and repeat.

Figure 1.2: *Pseudo-Code of SORA (Sequential Optimization and Reliability Assessment)*

1.5 Epistemic Uncertainty Quantification (optional)

Conduct a global sensitivity analysis via Sobol indices. Quantify how uncertainties in material fatigue (Weibull shape parameter) and drag estimation could dominate system risk.

Critical question: When should you prioritize data collection over design changes?

Chapter 2

Deterministic Design with Probabilistic Modeling of Failure Boundaries

Context and modeling notes

We minimize takeoff mass for a quadcopter (2 kg payload, 50 km range) while enforcing disk loading, blade loading, and energy reserve constraints per the project brief. Two implementation choices are noted and used consistently in this report:

1. **Fixed rotor speed:** We set a constant $\Omega = 1110$ rad/s based on benchmarking similar delivery drones and targeting total cruise shaft power $4P_{\text{cruise}} \in [500, 1000]$ W. (The cubic trim solve from the brief is retained in comments but not used here.)

2. **Deterministic energy margin:** In this chapter, we use only the 30% SOC reserve, i.e., `extra_m=0.0`. Under uncertainty (Chapter 4), we may reintroduce a positive `extra_m` to maintain reliability.

2.1 Deterministic Design and Optimization

2.1.1 Decision Variables, Objective, Constraints

Design $\mathbf{x} = [\,r,\ V_\infty\,]^\top$. Objective: total mass $m_{\text{total}}(\mathbf{x})$ [kg]. Inequality constraints are expressed as margins $g_i(\mathbf{x}) \geq 0$:

$$g_1 = DL_{\max} - \frac{T}{A}, \qquad g_2 = BL_{\max} - \frac{C_T}{\sigma}, \qquad g_3 = E_{\text{usable}} - E_{\text{req}}, \tag{2.1}$$

with $T = W_{\text{total}}/4$, $A = \pi r^2$, $C_T = \dfrac{T}{\rho A (\Omega r)^2}$,

$$E_{\text{req}} = 4P_{\text{hover}}\, t_{\text{hover}} + 4P_{\text{cruise}} \frac{R}{V_\infty}, \quad E_{\text{usable}} = (1 - \text{margin})\, \eta_{\text{batt}}\, \rho_b\, m_{\text{batt}}.$$

2.1.2 Constants and Inputs

```
rho   = 1.225;          % [kg/m^3] air density
g     = 9.81;           % [m/s^2]  gravity

% Mission / payload
m_payload = 2.0;        % [kg]
R         = 50e3;       % [m]   one-way range
thover    = 60;         % [s]   hover per leg (quad has 4 running
    legs)

% Aerodynamic / rotor parameters (deterministic)
sigma   = 0.13;         % [-]    rotor solidity
Cd0     = 0.012;        % [-]    zero-lift drag coef
V_tip   = 200;          % [m/s]  fixed tip speed

% Battery data
rho_b_Wh   = 158;       % [Wh/kg] specific energy
eta_batt   = 0.85;      % charge-discharge efficiency
margin     = 0.30;      % keep 30% SOC unused
rho_b      = rho_b_Wh*3600;  % [J/kg]

% Structural limits
DL_max = 250;           % [N/m^2]
BL_max = 0.14;          % [-]
```

2.1.3 Design Variables and Bounds

```
% x = [r; V_inf] - rotor radius [m] & cruise speed [m/s]
x0 = [0.2; 25];
lb = [0.15 ; 10];          % r >= 0.15 m,  V <= 10 m/s
ub = [0.4 ; 80];           % r >= 0.4 m,  V <= 80 m/s
```

- **Decision vector.** We optimize rotor radius per rotor r and cruise speed V_∞. Holding the number of rotors (4) fixed keeps the problem compact while still shaping performance via disk loading, blade loading, and mission energy.
- **Bounds.** $r \in [0.15, 0.40]$ m brackets practical mini-/small-UAS rotors; the lower bound avoids unreasonably high disk loading and tip Mach, the upper bound avoids unwieldy airframe sizes. $V_\infty \in [10, 80]$ m/s spans slow safe cruise up to an upper limit that keeps power within the target window (500–1000 W total) for $\Omega = 1110$ rad/s.
- **Initial guess.** $x_0 = [0.2,\ 25]$ is feasible, near the center of the box, and speeds up convergence versus starting at a bound.

- **Sensitivity.** Optima frequently lie on a *loading* constraint; indeed, we observe the blade-loading margin active at the solution while r sits at its lower bound—consistent with minimizing mass by reducing disk area until a constraint binds.

2.1.4 Optimizer Setup

```
opts = optimoptions('fmincon','Algorithm','sqp', ...
                    'OptimalityTolerance',1e-8, ...
                    'ConstraintTolerance',1e-8, ...
                    'StepTolerance',1e-8, ...
                    'Display','iter', ...
                    'MaxFunctionEvaluations',2e3);
problem = createOptimProblem('fmincon', 'objective', @objFun,
   ...
                             'nonlcon', @nlcon, 'x0', x0, ...
                             'lb', lb, 'ub', ub, 'options',
                                opts);
[x_opt, m_opt] = fmincon(problem);

fprintf('\nOptimal radius    : %.3f  m', x_opt(1));
fprintf('\nOptimal speed     : %.1f  m/s', x_opt(2));
fprintf('\nTake-off weight   : %.2f N (%.2f kg)\n', m_opt*g,
   m_opt);

ggg = limitStates(x_opt)
```

- **Algorithm.** SQP is well-suited to smooth, small-dimensional, constrained problems; it efficiently handles the nonlinear margins and the mildly nonconvex mass model.
- **Tolerances.** Tight optimality/constraint/step tolerances (10^{-8}) ensure the active constraint (blade loading) is truly active ($g_2 \approx 0$) and not a numerical artifact. You can relax these to 10^{-6} if speed is preferred.
- **Function evaluations.** `MaxFunctionEvaluations` of 2000 is ample given the low dimension (2), even with the inner mass iteration.
- **Printing & diagnostics.** The final prints show $x^\star$, $m^\star$, and the limit-state vector $\mathbf{g}(x^\star)$, which is useful to verify which constraints bind.

2.1.5 Saving the Baseline

```
[~, st_opt] = sizingModel(x_opt);
baseline = struct( ...
   'A',         pi*x_opt(1)^2 , ...
   'T',         m_opt*g/4     , ...
   'Omega0',    st_opt.Omega  , ...
```

```
    'P_hover',  (1/0.75)*(m_opt*g/4)^1.5/sqrt(2*rho*pi*x_opt(1)
      ^2) , ...
    'E_use',     st_opt.E_usable , ...
    'DL_max',    DL_max , ...
    'BL_max',    BL_max , ...
    'thover',    thover , ...
    'R',         R );
save baseline_constants.mat baseline
```

2.1.6 Objective and Nonlinear Constraints

```
%% OBJECTIVE FUNCTION
   -----------------------------------------------------
function m = objFun(x)
    m = sizingModel(x);  % returns total mass [kg]
end

%% NON-LINEAR CONSTRAINTS
   -------------------------------------------------
function [c,ceq] = nlcon(x)
    g   = limitStates(x);
    c   = -(g);      % enforce g >= 0 as c <= 0
    ceq = [];
end
```

- **Separation of concerns.** The wrappers keep the optimizer interface clean while `sizingModel` and `limitStates` encapsulate physics. This makes it trivial to swap models later (e.g., probabilistic variants).
- **Objective.** We minimize total mass m_{total} (kg). Using kg rather than N avoids an unnecessary scaling by g inside the optimizer; we print both afterward for interpretation.
- **Inequality mapping.** `fmincon` expects $c(x) \leq 0$. Our margins are defined as $g_i \geq 0$, so $c = -g$ is the standard flip. No equalities are posed ($ceq = \varnothing$).
- **Gradients.** We let `fmincon` finite-difference the gradients. Given the problem size and smoothness, this is reliable and simpler than deriving Jacobians analytically at this stage.

2.1.7 Sizing Model

```
function [m_total, st] = sizingModel(x)
    % Pull shared variables from caller
    g           = evalin('base','g');
    rho         = evalin('base','rho');
    sigma       = evalin('base','sigma');
```

```
Cd0         = evalin('base','Cd0');
rho_b       = evalin('base','rho_b');
eta_batt    = evalin('base','eta_batt');
margin      = evalin('base','margin');
V_tip       = evalin('base','V_tip');
thover      = evalin('base','thover');
R           = evalin('base','R');
m_payload   = evalin('base','m_payload');
DL_max      = evalin('base','DL_max');
BL_max      = evalin('base','BL_max');

% Design vars
r    = x(1);                      % [m]
Vinf = x(2);                      % [m/s]

% Initial mass guess [kg]
m_total = m_payload + 2.0;

for k = 1:50
    W_total = m_total * g;        % [N]
    A = pi*r^2;                   % disk area per rotor
    T = W_total/4;                % thrust per rotor [N]

    % Hover power (Eq.6)
    P_hover = (1/0.75)*T^1.5/sqrt(2*rho*A);

    % ----- cruise power with fixed Omega (benchmark-
       informed) -----
    % (The cubic trim solve from the brief is kept as
       comments.)
    % Omega_candidates = roots([a3 0 a1 a0]); ... (trim
       solve)
    % --------------------- EDIT #1 ---------------------
    Omega = 1110;  % [rad/s] fixed RPM chosen to yield 4*
       P_cruise ~ 0.5-1.0 kW

    mu       = Vinf/(Omega*r);
    P_cruise = (sigma*Cd0/8)*(1 + 4.65*mu^2)*rho*A*Omega^3*
       r^3;

    % Mission energy requirement [J]
    E_req = P_hover*4*thover + 4*P_cruise*(R/Vinf);

    % Battery mass (30% SOC reserve only for deterministic
       case)
```

```
        % --------------------- EDIT #2 ---------------------
        extra_m = 0.0;  % deterministic Tasks 1-2: no extra
           beyond 30% SOC
        m_batt = E_req / ((1-margin-extra_m)*eta_batt*rho_b);
        E_usable = (1-margin)*eta_batt*rho_b*m_batt;

        % Propulsion empirical masses
        P_installed = 4*max(P_hover,P_cruise);
        m_motors = 2.506e-4 * P_installed;
        m_ESC    = 3.594e-4 * P_installed;
        m_rotors = 4*(0.7484*r^2 - 0.0403*r);

        % Frame mass scales with total mass (empirical)
        m_empty_guess = m_motors + m_ESC + m_rotors;
        m_total_tmp   = m_payload + m_batt + m_empty_guess;
        m_frame       = 0.5 + 0.2*m_total_tmp;
        m_empty       = m_empty_guess + m_frame;

        % Fixed-point iteration
        m_new = m_payload + m_batt + m_empty;
        if abs(m_new - m_total) < 1e-4
            m_total = m_new; break; end
        m_total = m_new;
     end

    % Constraint quantities
    W_total = m_total * g;
    T = W_total/4;
    A = pi*r^2;
    DL = T/A;
    CT = T/(rho*A*(Omega*r)^2);

    % Pack outputs
    st = struct('DL',DL,'DL_max',DL_max, ...
                'BL',CT/sigma,'BL_max',BL_max, ...
                'E_req',E_req,'E_usable',E_usable, ...
                'diskArea',A,'Omega',Omega);
end
```

- **Data access.** `evalin('base',...)` keeps the function signature compact. For Monte Carlo/FORM later, we can either keep this (and overwrite base params per sample) or pass parameters explicitly for thread-safety.
- **Inner fixed-point loop.** Mass appears on both sides (component regressions and frame scaling depend on gross mass). We iterate:

$$m^{(k+1)} = m_{\text{payload}} + m_{\text{batt}}(m^{(k)}) + m_{\text{empty}}(m^{(k)}),$$

stopping when $|m^{(k+1)} - m^{(k)}| < 10^{-4}$ kg. This is numerically stable here because the frame law's slope (≈ 0.2) and power regressions produce a contraction for typical sizes.

- **Power models.** Hover power uses the classical momentum form with a 1/0.75 factor for induced/empirical losses. Cruise power is the profile-dominated expression with advance ratio μ. Fixing $\Omega = 1110$ is a deliberate, benchmark-driven choice ensuring $4P_{\text{cruise}}$ sits in 0.5–1.0 kW for feasible r, V_∞.
- **Energy/reserve.** Deterministic Tasks 1–2 use only the 30% SOC margin (`extra_m=0`); under uncertainty (Task 4) we may add extra margin to hit reliability targets without violating $g_3 \geq 0$.
- **Outputs.** We return m_{total} and a struct with all constraint-relevant quantities (DL, BL, energies, Ω, disk area) so other functions don't recompute internals.

2.1.8 Limit State Functions

```
function [g, st] = limitStates(x)
    % g(1) : disk-loading margin        DL_max - DL
    % g(2) : blade-loading margin       BL_max - CT/sigma
    % g(3) : energy-reserve margin      E_usable - E_req
    [~, st] = sizingModel(x);
    g = [ st.DL_max - st.DL ; ...
          st.BL_max - st.BL ; ...
          st.E_usable - st.E_req ];
end
```

- **Definition.** Each g_i is a "margin to failure." The design is feasible iff all $g_i \geq 0$. This convention is convenient for optimization and reliability methods (FORM/SORM).
- **Units & interpretation.** g_1 has units of N/m^2 (disk-loading slack), g_2 is dimensionless (blade-loading slack), and g_3 is Joules (energy slack).
- **Active constraints.** At the reported optimum, $g_2 \approx 0$ (active), $g_1 > 0$, $g_3 \gg 0$. This pattern is physically sensible: blade loading typically binds before disk loading for small r with fixed Ω.
- **Hook for uncertainty.** In Chapter 4, we will evaluate $g(x, \boldsymbol{\xi})$ where $\boldsymbol{\xi}$ randomizes C_{d_0}, σ (and possibly η_{batt}, ρ_b), then enforce reliability targets by keeping $\mathbb{P}(g_i < 0)$ below thresholds or by SORA with target reliability indices.

2.2 Probabilistic Modeling of Failure Boundaries

The limit-state surfaces are:

$$g_1(\mathbf{x}) = DL_{\max} - \frac{T}{A}, \quad g_2(\mathbf{x}) = BL_{\max} - \frac{C_T}{\sigma}, \quad g_3(\mathbf{x}) = E_{\text{usable}} - E_{\text{req}}. \tag{2.2}$$

Uncertainties (e.g., in C_{d_0} and σ) map into g_2 and g_3 (and indirectly into g_1 if Ω were solved by trim). For Tasks 1–2 we hold C_{d_0} and σ fixed (deterministic) with `extra_m` $= 0.0$. In the following chapters, we will:

- assign distributions to C_{d_0}, σ, etc.,
- potentially reintroduce `extra_m` > 0 to achieve target reliabilities,
- and optimize $\mathbf{x}$ under reliability constraints (e.g., SORA/FORM).

2.3 Conclusion

We finalized the deterministic sizing with a fixed RPM choice $\Omega = 1110$ rad/s grounded in power benchmarking and enforced only the 30% SOC energy reserve (`extra_m` $= 0.0$). The resulting limit-state functions are ready for probabilistic analysis in subsequent tasks; under uncertainty, we may increase `extra_m` to satisfy reliability targets without excessively penalizing mass.

Table 2.1 and Table 2.2 show the result of the deterministic design with the assumptions and procedures detailed as in the previous sections.

Table 2.1: *Deterministic design results with 30% safety margin*

Optimal Prop Radius	Optimal Cruise Velocity	Take-Off Weight	$g(\mathbf{x})$
0.150 m	66.1 m/s	4.87 kg	80.9201
			0.1010
			0.0000

Table 2.2: *Deterministic design results with 30+5% safety margin*

Optimal Prop Radius	Optimal Cruise Velocity	Take-Off Weight	$g(\mathbf{x})$
0.150 m	69.1 m/s	5.00 kg	76.4110
			0.1000
			33201

Chapter 3

Reliability Analysis

We evaluate reliability for the three limit states defined in Chapter 2:

$$g_1 = DL_{\max} - \frac{T}{A}, \qquad g_2 = BL_{\max} - \frac{C_T}{\sigma}, \qquad g_3 = E_{\text{use}} - E_{\text{req}}.$$

Random inputs are the aerodynamic zero-lift drag coefficient C_{d_0} and rotor solidity σ, modeled as *lognormal* to ensure positivity and avoid divergence in the algorithms. The analysis runs at a deterministic design point $\mathbf{x}^\star = [r^\star, V_\infty^\star]$ (from Chapter 2) and uses the saved baseline snapshot (`baseline_constants.mat`) that includes $A, T, \Omega_0, P_{\text{hover}}, E_{\text{use}}, DL_{\max}, BL_{\max}, t_{\text{hover}}, R$.

Two practical modeling choices (from Chapter 2).

- Fixed rotor speed $\Omega = 1110$ rad/s, selected by benchmarking similar delivery drones so that total cruise power $4P_{\text{cruise}} \in [500, 1000]$ W.
- Deterministic reserve uses exactly 30% SOC (`extra_m=0`) unless stated otherwise. Increasing `extra_m` increases mass (and E_{use}) and will change the reliability—especially for the energy limit.

3.1 Random Inputs and Transformation

We take $C_{d_0} \sim \text{LogNormal}(\mu_C, \sigma_C)$ and $\sigma \sim \text{LogNormal}(\mu_\sigma, \sigma_\sigma)$. Using their means $\bar{C}_{d_0}$, $\bar{\sigma}$ and coefficients of variation COV_C, COV_σ (from `buildRandomInputs`), the underlying normal parameters are

$$\mu_{\ln} = \ln(\bar{C}_{d0}) - \tfrac{1}{2}\ln(1 + \text{COV}_C^2), \quad s_{\ln} = \sqrt{\ln(1 + \text{COV}_C^2)} \tag{3.1}$$

(and analogously for σ). We work in standard normal space $u \in \mathbb{R}^2$ and map

$$C_{d_0}(u_1) = \exp\big(\mu_{\ln} + s_{\ln}\,\tilde{u}_1\big), \quad \sigma(u_2) = \exp\big(\mu_{\sigma\,\ln} + s_{\sigma\,\ln}\,\tilde{u}_2\big), \tag{3.2}$$

where we *clip* $\tilde{u}_i = \text{clip}(u_i) \in [-4, 4]$ to avoid overflow in exponentials and stabilize finite differences. This truncates extreme tails slightly, trading negligible bias for robustness.

Assigning normal distributions to C_{d_0} or σ permits negative values, which are non-physical and make the algorithms diverge in experiments. Lognormal enforces positivity and produces stable runs.

3.2 Methods

3.2.1 FORM (HL–RF)

We use the Hasofer–Lind–Rackwitz–Fiessler (HL–RF) fixed-point scheme in u-space:

$$\alpha_k = \frac{\nabla_u g(u_k)}{\|\nabla_u g(u_k)\|}, \quad \beta_k = -\frac{g(u_k)}{\|\nabla_u g(u_k)\|}, \quad u_{k+1} = \beta_k \, \alpha_k. \tag{3.3}$$

Convergence yields the most probable failure point (MPP) $u^\star$, reliability index $\beta = \|u^\star\|$, and $p_f^{\mathrm{FORM}} = \Phi(-\beta)$. Gradients $\nabla_u g$ are computed by central differences in physical variables (C_{d_0}, σ) and then chain-ruled to u (the code multiplies by $s_{\ln} C_{d_0}$ and $s_{\sigma \ln} \sigma$).

Disk-loading special case. g_1 in our implementation is independent of C_{d_0} and σ (it uses the baseline T and A only), hence $\nabla_u g_1 = 0$. We therefore return $\beta = \infty$ and $p_f = 0$ for g_1 (and SORM is undefined/NaN), matching the code's behavior.

3.2.2 SORM (Breitung)

With $u^\star$ and $\nabla g(u^\star)$, we compute the Hessian numerically and project curvature κ along the unit tangent t orthogonal to ∇g. Breitung's correction gives

$$p_f^{\mathrm{SORM}} \approx \frac{p_f^{\mathrm{FORM}}}{\sqrt{1 + \beta \, \kappa}}. \tag{3.4}$$

3.2.3 Directional Sampling (DS)

We sample random directions d on the unit circle in u-space, grow a radial bracket until any limit state crosses zero, then bisect to the boundary and record failure. The estimator is the fraction of directions resulting in failure, with variance $p_f(1 - p_f)/N$ and CoV $\sqrt{\mathrm{var}}/p_f$. DS captures nonlinearity and curvature beyond FORM/SORM.

3.2.4 Importance Sampling (IS)

For each limit, we sample $u \sim \mathcal{N}(u^\star, I)$ and weight by

$$w(u) = \frac{\phi(u)}{\phi(u - u^\star)} = \exp\Big(\tfrac{1}{2}\|u - u^\star\|^2 - \tfrac{1}{2}\|u\|^2\Big), \tag{3.5}$$

estimating $p_f = \mathbb{E}[w \, \mathbf{1}_{\{g(u) \le 0\}}]$. When β is very small (e.g. energy with `extra_m`=0), centering at a near-zero MPP can make IS variance large; when β is large, failures become extremely rare in practice and *finite* samples may yield an estimate of 0.

3.3 MATLAB implementation

```
function [betaF , pfF , pfS] = runReliability2(x_opt , Ndir ,
   Nis)
% runReliability  -  FORM / SORM  + optional DS  + optional
   Importance Sampling
%
%  [betaF ,pfF ,pfS]              = runReliability(x_opt)
%  [...] = runReliability(x_opt , Ndir)                 % + DS
%  [...] = runReliability(x_opt , Ndir , Nis)           % + DS +
   IS
%
%  Cd0  ~ log-normal ;   sigma ~ log-normal   (positivity
   enforced)

if nargin<2,  Ndir = 0; end           % default: skip DS
if nargin<3,  Nis  = 0; end           % default: skip IS

clip = @(z) max(min(z,4),-4);         % identical truncation
   everywhere

%% 0.  Constants & baseline
C            = load('baseline_constants.mat','baseline').
   baseline;
[r0 , V0]    = deal(x_opt(1), x_opt(2));

%% 1.  Random-variable parameters  (both log-normal)
[R,~]    = buildRandomInputs(false);
mu_ln    = log(R(1).mu) - 0.5*log(1+R(1).cov^2);
sig_ln   = sqrt(log(1+R(1).cov^2));                 % Cd0

mu_sig   = log(R(2).mu) - 0.5*log(1+R(2).cov^2);
sig_sig  = sqrt(log(1+R(2).cov^2));                 % sigma

%% 2.  FORM (HL-RF)
betaF = zeros(3,1);  pfF = zeros(3,1);
uStar = zeros(2,3);  gradU=zeros(2,3);

for j = 1:3
    [betaF(j),pfF(j),uStar(:,j),gradU(:,j)] = ...
        formHLRF(@(u) gfun(u,j), 1e-4, 50);
end

%% 3.  SORM (Breitung)
pfS = zeros(3,1);
for j = 1:3
    kappa  = curvature(uStar(:,j),@(u) gfun(u,j),gradU(:,j));
```

```
    pfS(j) = pfF(j) ./ sqrt(1 + betaF(j)*kappa);
end

%% 4.  Directional Sampling  (optional)
if Ndir > 0
    [pfDS , cvDS] = directionalSampling(Ndir);
    fprintf('\nDS  (N=%d)  pf ~ [%5.2e  %5.2e  %5.2e]   C.o.V ~
        [%4.2f  %4.2f  %4.2f]\n',...
            Ndir , pfDS , cvDS );
end

%% 5.  Importance Sampling around the FORM MPP  (optional)
if Nis > 0
    [pfIS , cvIS] = importanceSampling(Nis);
    fprintf('IS  (N=%d)  pf ~ [%5.2e  %5.2e  %5.2e]   C.o.V ~
       [%4.2f  %4.2f  %4.2f]\n',...
            Nis , pfIS , cvIS );
end

% ======================== nested helpers
   ==============================

   function g = limitStates(Cd0,sigma)
       mu   = V0/(C.Omega0*r0);
       Pc   = (sigma*Cd0/8)*(1+4.65*mu^2)*1.225*C.A*C.Omega0^3*
          r0^3;
       Ereq = C.P_hover*4*C.thover + 4*Pc*(C.R/V0);

       DL   = C.T / C.A;
       CT   = C.T /(1.225*C.A*(C.Omega0*r0)^2);

       g(1) = C.DL_max - DL;
       g(2) = C.BL_max - CT/sigma;
       g(3) = C.E_use  - Ereq;
   end

   function [g,grad_u] = gfun(u,idx)
       Cd0   = exp(mu_ln  + sig_ln  * clip(u(1)));
       sigma = exp(mu_sig + sig_sig * clip(u(2)));

       gvec = limitStates(Cd0,sigma);   g = gvec(idx);

       rel = 1e-2;  x0=[Cd0 sigma];  dgx=zeros(1,2);
       for k = 1:2
           d  = rel*x0(k);
```

```
            gp = limitStates(x0(1)+(k==1)*d , x0(2)+(k==2)*d);
            gm = limitStates(x0(1)-(k==1)*d , x0(2)-(k==2)*d);
            dgx(k) = (gp(idx)-gm)/(2*d);
        end
        grad_u = (dgx .* [sig_ln*Cd0 , sig_sig*sigma]).';
    end

    function [beta,pf,u_fin,grad_fin] = formHLRF(fun,tol,Nmax)
        u=[0;0];
        for k=1:Nmax
            [g,grad]=fun(u); ng=norm(grad);
            if ng<1e-5
                beta = sign(g)*inf; pf=(g<=0); u_fin=u; grad_fin
                   =grad; return
            end
            alpha=grad/ng; beta=-g/ng; u_new=beta*alpha;
            if norm(u_new-u)<tol, u=u_new; break, end, u=u_new;
        end
        beta=norm(u); pf=normcdf(-beta); u_fin=u; grad_fin=grad;
    end

    function k=curvature(u0,fun,grad0)
        h=1e-3; H=zeros(2);
        for i=1:2
            e=zeros(2,1); e(i)=1;
            [~,gp]=fun(u0+h*e); [~,gm]=fun(u0-h*e);
            H(:,i)=(gp-gm)/(2*h);
        end
        t=[-grad0(2); grad0(1)]; t=t/norm(t);
        k=(t'*H*t)/max(norm(grad0),1e-8);
    end

  function [pf , cv] = directionalSampling(N)
% Directional Sampling with full trace saved to ds_debug.mat

    maxGrow   = 1e3;          % beta cap if we never reach
       failure
    tolBisec = 1e-3;
    maxIter   = 40;

    fail    = zeros(1,3);
    b_enter   =  inf( N ,1);           %#ok<*NASGU>
    g_enter   =  nan( N ,3);
    g_design =  nan( N ,3);
    status    =  zeros( N ,1,'int8');
```

```
for ii = 1:N
    d = randn(2,1);  d = d/norm(d);

    % -- grow bracket until any g<=0
       ---------------------------------
    bL = 0.07;  bH = 0.2;
    while true
        uH = bH*d;
        CD0_e=exp(mu_ln  + sig_ln  * clip(uH(1)));
        sigma_e=exp(mu_sig + sig_sig * clip(uH(2)));

        gH = limitStates( ...
                exp(mu_ln  + sig_ln  * clip(uH(1))) , ...
                exp(mu_sig + sig_sig * clip(uH(2))) );
        if any(gH<=0) || bH>maxGrow, break, end
        bH = bH*2;
    end

    if bH>maxGrow           % never crossed -> safe in this
        direction
        status(ii) = 0;
        continue
    end
    status(ii) = 1;
    b_enter(ii)= bH;
    g_enter(ii,:)= gH;

    % -- bisection
       ---------------------------------------------------
    for iter = 1:maxIter
        bM = 0.5*(bL+bH);   uM = bM*d;
        gM = limitStates( ...
                exp(mu_ln  + sig_ln  * clip(uM(1))) , ...
                exp(mu_sig + sig_sig * clip(uM(2))) );
        if all(gM>0), bL=bM; else, bH=bM; end
        if bH-bL<tolBisec, break, end
    end

    uF = bH*d;
            Cd0_f=exp(mu_ln  + sig_ln  * uF(1));
            sigma_f=exp(mu_sig + sig_sig * uF(2));
     % fprintf('DS  Cd0=%g  sig=%g  E_use=%g\n', Cd0_f,
        sigma_f, C.E_use);
```

```
        gF = limitStates(Cd0_f,sigma_f);
        g_design(ii,:) = gF;

        fail = fail + (gF<=0);
    end

    pf = fail / N;
    var = max( pf .* (1-pf) / N , 0 );
    cv  = sqrt(var) ./ max(pf,eps);

    % save ds_debug.mat  b_enter g_enter g_design status
end

   % ---------- Importance Sampling
      -------------------------------------

   function [pfIS,cvIS] = importanceSampling(N)
       pfIS = zeros(1,3);   varIS = zeros(1,3);

       % precompute MPPs and Cholesky of the shifted covariance
           (identity)
       mpp = uStar;                          % each column is u*
          for g_j
       eye2 = eye(2);

       for j = 1:3
           u0 = mpp(:,j);                    % centre IS at FORM
              design point
           w_sum = 0;  w2_sum = 0;  nf = 0;

           for n = 1:N
               u_shift = u0 + randn(2,1);          % sample
                  from N(u0,I)
               % importance weight  w = phi(u)/phi_shifted(u)
               % phi_shifted  = phi(u-u0)   because Sigma = I
               w = exp( 0.5*(u_shift-u0).'*(u_shift-u0) ...
                         -0.5*u_shift.'*u_shift );

               Cd0   = exp(mu_ln  + sig_ln  * clip(u_shift(1)))
                  ;
               sigma = exp(mu_sig + sig_sig * clip(u_shift(2)))
                  ;
               gj    = limitStates(Cd0,sigma); gj = gj(j);

               I_f   = (gj <= 0);        % indicator of failure
               w_sum = w_sum + w*I_f;
```

```
                w2_sum= w2_sum+ w^2*I_f;
            end
            pfIS(j) = w_sum / N;
            varIS(j)= max( w2_sum/N - pfIS(j)^2 , 0 );
        end
        cvIS = sqrt(varIS) ./ max(pfIS,eps);
    end

dd=limitStates(0.0237,4.1222e-04);
end
```

How to run

Call:

`runReliability2(x_opt, 1e3, 8e3)`

where x_{opt} is from Chapter 2 sizing, $N_{\text{dir}} = 10^3$ for DS and $N_{\text{IS}} = 8 \times 10^3$ for IS. If you omit $N_{\text{dir}}, N_{\text{IS}}$, the function returns only FORM/SORM (β, p_f). With $N_{\text{dir}}, N_{\text{IS}}$ set, it prints DS/IS estimates of p_f as well.

3.4 Numerical Results and Interpretation

Case A: `extra_m = 0` (30% SOC only)

Table 3.1: *Monte Carlo estimators at* $r = 0.150\,\text{m}$, $V_\infty = 66.1\,\text{m/s}$ *(failure probabilities and CoV).*

Method	N	p_f (DL)	p_f (BL)	p_f (Energy)	COV (DL)	COV (BL)	COV (Energy)
DS	1000	0.00e+00	3.60e-02	4.88e-01	0.00	0.16	0.03
IS	8000	0.00e+00	0.00e+00	4.46e-01	0.00	0.00	1.16

Table 3.2: *FORM/SORM at* $r = 0.150\,\text{m}$, $V_\infty = 66.1\,\text{m/s}$.

Limit	β	p_f (FORM)	p_f (SORM)
Disk load	∞	0.00e+00	N/A†
Blade load	10.11	2.42e-24	2.42e-24
Energy	0.05	4.78e-01	4.78e-01

† SORM not defined here because $\nabla g_1 = 0 \Rightarrow \beta = \infty$ for the disk-loading limit at this baseline.

Discussion.

- g_1: independent of $(C_{d_0}, \sigma) \Rightarrow \nabla g_1 = 0 \Rightarrow \beta = \infty$, $p_f = 0$.

- g_2: $\beta \approx 10 \Rightarrow p_f \approx 10^{-24}$ (negligible) by FORM/SORM; DS reports $\approx 3.6\%$ due to directionwise crossing of other limits first (and sampling noise at small p_f).
- g_3: $\beta \approx 0.05 \Rightarrow p_f \approx 0.478$ (very close to 50%), which is expected because the design sits near the energy boundary; lognormal skew makes it slightly *below* 50%.
- IS for energy shows $p_f \approx 0.446$ but with large CoV (≈ 1.16): centering IS at a near-zero β MPP is statistically inefficient; more samples or variance-reduction tweaks (e.g. scaling) would improve precision.

Case B: `extra_m = 0.10` (additional energy cushion)

Table 3.3: *Monte Carlo estimators at* $r = 0.150\,\mathrm{m}$*,* $V_\infty = 72.6\,\mathrm{m/s}$ *(failure probabilities and COV).*

Method	N	p_f (DL)	p_f (BL)	p_f (Energy)	COV (DL)	COV (BL)	COV (Energy)
DS	1000	0.00e+00	4.60e-02	4.29e-01	0.00	0.14	0.04
IS	8000	0.00e+00	0.00e+00	0.00e+00	0.00	0.00	0.00

Table 3.4: *FORM/SORM at* $r = 0.150\,\mathrm{m}$*,* $V_\infty = 72.6\,\mathrm{m/s}$*.*

Limit	$\boldsymbol{\beta}$	p_f (FORM)	p_f (SORM)
Disk load	∞	0.00e+00	N/A†
Blade load	9.09	5.09e-20	5.09e-20
Energy	2.61	4.57e-03	4.57e-03

† SORM not defined here because $\nabla g_1 = 0 \Rightarrow \beta = \infty$ for the disk-loading limit at this baseline.

Disk loading: consistently safe

All three methods agree that disk loading is effectively impossible to violate at the analyzed design: FORM/SORM give $\beta = \infty \Rightarrow p_f = 0$ and both DS and IS report zero failures. This aligns with the model structure (no dependence on C_{d_0} or σ in g_1 under the saved baseline), so no action is needed here.

Energy reserve: near the boundary, $p_f \approx 0.47$

Across methods, the energy limit clusters around $p_f \in [0.46, 0.48]$ at the baseline with `extra_m`=0, which matches the interpretation of $\beta \approx 0$ (nearly planar surface at the design point). Directional Sampling with $N_{\mathrm{dir}} = 1000$ already attains a statistical error of about 3%, so differences such as 0.474 vs. 0.478 are within sampling noise. Importance Sampling centered at the FORM MPP shows high weight variance (flat surface far from the proposal center), hence a large CoV; nevertheless its mean remains statistically consistent with DS. *Conclusion: energy failure probability is* $\sim 47\%$*, i.e., nearly every other mission would run out of energy—clearly unacceptable, but self-consistent.*

Blade loading: FORM/SORM underestimation; trust DS

FORM/SORM predict astronomically small probabilities (e.g., $\beta \approx 9$) while DS sees percent-level failures (about 5% with $N_{\text{dir}} = 1000$, CoV $\approx 14\%$). This gap is *not* sampling noise and *not* an IS artifact; it reflects strong nonlinearity/curvature of the blade-loading surface away from the deterministic point. In particular, the local gradient at the design point points mostly along the σ axis while the curved failure set bends toward higher C_{d0} at moderate σ—so the linear (or quadratic) surrogate at the MPP misses the region that DS discovers along many directions. *Conclusion: for the blade limit, DS is the reliable estimator here; FORM/SORM are non-conservative.*

Sampling guidance and redesign implications

- **Increase DS directions.** For $p_f \approx 0.05$ on blade load, N_{dir}=1000 gives CoV $\sim 14\%$; raise to N_{dir}=10,000 to push CoV below 5% and stabilize the estimate.
- **Targeted IS for blade load.** Re-center IS at the blade-load MPP (not the energy MPP) or use an adaptive mixture to obtain an independent check with CoV $\sim 5\%$.
- **Design change is required.** With $p_f^{\text{energy}} \approx 47\%$ and $p_f^{\text{blade}} \approx 5\%$, the design does not meet a 90% mission-success goal. Increase usable energy (battery/margin), rotor radius, or both, then re-run the full reliability loop. Also document explicitly that FORM/SORM are acceptable for the energy limit *near* $\beta \approx 0$ but unreliable for the blade limit due to strong curvature.

- Re-sizing with higher `extra_m` (and thus different $x^\star$) raises E_{use} and pushes the energy limit away from failure: β grows from ≈ 0.05 to ≈ 2.61 and p_f drops from ~ 0.48 to ~ 0.0046 by FORM/SORM.
- DS/IS reflects the reduced energy risk (DS p_f drops from 0.488 to 0.429; IS returns 0 at this sample size because no weighted failures were observed—finite-sample underflow for small p_f is common).
- Blade-load risk remains negligible across both cases (FORM/SORM $p_f \ll 10^{-10}$).

On "FORM/SORM not changing with `extra_m`." FORM/SORM are *local* (linear/quadratic) at the current design point. If you *do not* re-optimize or update the baseline when `extra_m` changes, their numbers can appear insensitive. Here, since the baseline (and $x^\star$) changed, FORM/SORM *did* move (energy $\beta : 0.05 \to 2.61$). DS/IS, which probe globally, will typically show a monotone drop in p_f for the energy limit as `extra_m` increases.

When `extra_m` is too large. Heavier batteries raise gross mass. If re-sizing cannot produce adequate thrust T relative to disk area A, the disk-loading constraint will be violated everywhere in the input space, implying $p_f(g_1) = 1$ and $p_f(g_2) = p_f(g_3) = 0$ (the design fails by disk load before anything else). In practice, you'll see this after re-running Task 1 with a large `extra_m` and then invoking reliability on that new baseline.

3.5 Comparison of CoVs Across Methods

3.5.1 How CoV behaves for each method

FORM / SORM. Both are *deterministic* local approximations built at the MPP; they do not have sampling COV. Any discrepancy is a *model/linearization* error, not Monte Carlo noise.

Directional Sampling (DS). With N independent directions and an indicator estimator,

$$\mathrm{COV}_{\mathrm{DS}} \approx \frac{\sqrt{p_f(1-p_f)/N}}{p_f}.$$

This matches our runs:

- Blade load: N=1000, p_f≈0.036 ⇒ COV ≈ $\sqrt{0.036 \cdot 0.964/1000}/0.036 \approx 0.15$ (we observed $\sim$ 0.14–0.16).
- Energy (extra_m= 0): N=1000, p_f≈0.488 ⇒ COV ≈ 0.033 (we observed $\sim$ 0.03).

Note. Our DS implementation counts the fraction of directions that intersect failure. This captures global nonlinearity well; to make it *unbiased* for p_f one can include the correct radial weighting in the estimator. Without that weighting, interpret DS as a conservative proxy or validate with IS.

Importance Sampling (IS). COV depends on the proposal. Centering at the FORM MPP is efficient when β is moderate/large; it becomes poor when β≈0 (energy with extra_m= 0), where we saw a large COV ($\sim$ 1.16). For very small p_f, finite N often yields *zero observed failures*—the code then prints p_f=0 and COV = 0, which is misleading; it actually indicates insufficient effective samples or a poor proposal. In such cases, report an upper bound (e.g., "rule of three": $p_f \lesssim 3/N$ at ≈ 95%).

3.5.2 Why SORM underestimates p_f

SORM (Breitung) adjusts FORM with a *local* curvature term at a single design point:

$$p_f^{\mathrm{SORM}} \approx \frac{p_f^{\mathrm{FORM}}}{\sqrt{1+\beta\,\kappa}}.$$

Underestimation arises when this local picture makes failure look "harder to reach" than it truly is:

1. **Strong nonlinearity away from the MPP.** A quadratic fit in the tangent plane can misrepresent a surface that bends toward failure in regions not captured locally. In our blade limit, many directions discover failure at moderate σ and higher C_{d0} that the local surrogate misses.

2. **Multiple near-optimal failure routes.** FORM/SORM lock onto one MPP; probability mass from other routes is ignored.

3. **Skewed inputs via lognormal transforms.** With $x = \exp(\mu + s\,u)$ for C_{d_0} and σ, the exponential map stretches u-space asymmetrically. Curvature at the MPP may not reflect the broader failure set, biasing SORM low.

4. **Numerical curvature bias.** Finite-difference Hessians are sensitive to step size; overly small steps inject round-off (random κ sign), overly large steps truncate curvature. Positive-biased κ depresses p_f^{SORM} below reality.

5. **Clipping effects.** Clipping u to $[-4, 4]$ stabilizes exponentials but can subtly distort gradients/Hessians if the stencil touches the clip.

6. **Extreme β.** When $\beta{\approx}0$ (our energy limit at extra_m=0), SORM $\approx$ FORM and inherits its linearization error; when β is large (blade), even small curvature errors are magnified by $(1 + \beta\kappa)^{-1/2}$, pushing SORM further down.

Practical guidance

- **Always report COV/CI with DS/IS.** Scale N so COV $\lesssim$ 5% for decision-quality results (e.g., DS: N=10,000 for blade $p_f \sim 0.05$).

- **Re-center IS per limit state.** Use the *blade* MPP for blade IS, *energy* MPP for energy IS; target $\sim$ 100–200 effective failures.

- **Validate SORM.** Recompute κ with $h \in \{2{\times}10^{-3}, 10^{-3}, 5{\times}10^{-4}\}$. If p_f^{SORM} shifts materially, prefer FORM + DS/IS.

- **For zero-failure IS runs,** quote an upper bound (e.g., $p_f \lesssim 3/N$ at $\sim$ 95%) rather than "0".

3.6 Conclusion

- With `extra_m`=0 (30% SOC only), the design sits roughly on the energy boundary ($\beta \approx 0$), giving $p_f \approx 0.5$ for g_3. This is *not* acceptable for reliability targets.

- Adding energy margin (e.g., `extra_m`=0.10) moves the energy limit to $\beta \approx 2.6$ ($p_f \approx$ 0.5% by FORM/SORM), at the cost of higher mass and a slightly different $V_\infty^\star$.

- For robust estimates when $\beta \approx 0$, prefer DS or increase IS samples / adjust centering/scaling. When β is large, expect IS to occasionally report $p_f = 0$ at finite N.

- Keep C_{d0} and σ lognormal. Using normal distributions induces negative draws and destabilizes the algorithms.

Chapter 4

Optimization Under Uncertainty

We perform Optimization Under Uncertainty using the Sequential Optimization and Reliability Assessment (SORA) framework for the quadrotor concept sized in Chapter 2 and analyzed in Chapter 3. The design variables are rotor radius r, cruise speed V_∞, and a continuous energy reserve knob `extra_m` $\in [0, 0.35]$ (extra energy margin beyond the fixed 30% SOC retained). Uncertainties are in zero-lift drag coefficient C_{d0} and rotor solidity σ, both modeled as *lognormal*, consistent with Chapter 3.

Targets. Structural limits (disk, blade) use $p_f \leq 10^{-4}$, power/energy uses $p_f \leq 10^{-3}$. These translate to reliability indices $\beta_{\text{struct}} \approx 3.719$ and $\beta_{\text{power}} \approx 3.090$.

4.1 Problem statement

We minimize take-off mass $m(x)$ over $x = (r, V_\infty, \texttt{extra\_m})$ subject to reliability constraints on three limit states:

$$g_1 = DL_{\max} - \frac{T}{A}, \qquad g_2 = BL_{\max} - \frac{C_T}{\sigma}, \qquad g_3 = E_{\text{use}} - E_{\text{req}},$$
$$p_f[g_1] \leq 10^{-4}, \quad p_f[g_2] \leq 10^{-4}, \quad p_f[g_3] \leq 10^{-3}.$$

Uncertain inputs are C_{d_0} and σ with lognormal laws:

$$C_{d_0} = \exp(\mu_C + s_C u_1), \qquad \sigma = \exp(\mu_\sigma + s_\sigma u_2), \quad u \sim \mathcal{N}(0, I).$$

We keep $\Omega = 1110$ rad/s from Chapter 2 (benchmarked so $4P_{\text{cruise}} \in [0.5, 1.0]$ kW).

4.2 SORA algorithm

At iteration k:

1. **Reliability step.** At current design $x^{(k)}$, compute unit gradients $\alpha_i = \nabla_u g_i / \|\nabla_u g_i\|$ at the mean point $u = 0$ (chain-rule in u). Form equivalent points

$$u_i^* = -\,\beta_i\,\alpha_i,$$

with $\beta_1 = \beta_2 = \beta_{\text{struct}}$ and $\beta_3 = \beta_{\text{power}}$.

2. **Deterministic subproblem.** Solve

$$\min_x \; m(x) + \lambda\,\texttt{extra\_m} \quad \text{s.t.} \quad g_i\big(x,\, T^{-1}(u_i^*)\big) \geq 0,\; i = 1, 2, 3, \quad x \in \mathcal{X},$$

   where λ is a small regularizer that discourages overshooting the energy target and T^{-1} maps u to (C_{d0}, σ) via the lognormal transform.

3. Update $x^{(k+1)}$ and repeat until both x and $\{u_i^*\}$ stabilize.

4.3 Python Implementation

We show the key functions; the full script contains DS/IS checkers and a plain MC verifier.

Sizing model and limit states

Listing 4.1: Sizing and limit states

```python
@dataclass
class Design:  r: float; V: float; extra_m: float
@dataclass
class State:
    mass: float; T_per_rotor: float; A: float; CT: float
    E_use: float; P_hover_per_rotor: float

def sizing_model(des: Design, Cd0_nom=CD0_MEAN, sigma_nom=SIGMA_MEAN) -> State:
    # Fixed RPM sizing loop; battery sized so (30% + extra_m) is unused.
    # Computes P_hover, P_cruise (profile+mu), energy, installed power, and masses.
    ...

def limit_states(des: Design, par: tuple[float,float]) -> np.ndarray:
    Cd0, sigma = par
    st = sizing_model(des)                   # objective uses nominal (Cd0,sigma)
    DL = st.T_per_rotor / st.A
    g1 = DL_MAX - DL
    BL = st.CT / sigma
    g2 = BL_MAX - BL
    mu = des.V/(OMEGA*des.r)
    Pcr = (sigma*Cd0/8)*(1+4.65*mu**2)*RHO*st.A*OMEGA**3*des.r**3
    Ereq = st.P_hover_per_rotor*4*THOVER + 4*Pcr*(RANGE/des.V)
    g3 = st.E_use - Ereq
    return np.array([g1,g2,g3],float)
```

Chain-rule gradient and FORM (HL–RF)

We *do not* finite-difference directly in u (which can saturate); instead we FD in physical variables and apply the chain rule $dC_{d0}/du_1 = s_C C_{d0}$ and $d\sigma/du_2 = s_\sigma \sigma$.

Listing 4.2: Chain-rule gradient and HL–RF

```python
def g_of_u(des, u, idx0):
    Cd0, sigma = from_u_to_physical(u, clip=False)
    return float(limit_states(des, (Cd0, sigma))[idx0])

def grad_g_u(des, u, idx0):
    Cd0, sigma = from_u_to_physical(u, clip=False)
    rel = 1e-2; dCd0 = max(1e-12, rel*Cd0); dsig = max(1e-12, rel*
        sigma)
    # central differences in physical space
    dgdCd0 = (limit_states(des,(Cd0+dCd0,sigma))[idx0]
            - limit_states(des,(Cd0-dCd0,sigma))[idx0])/(2*dCd0)
    dgdsig = (limit_states(des,(Cd0,sigma+dsig))[idx0]
            - limit_states(des,(Cd0,sigma-dsig))[idx0])/(2*dsig)
    dCd0_du1   = CD0_LN.sig_log   * Cd0
    dsigma_du2 = SIGMA_LN.sig_log * sigma
    return np.array([dgdCd0*dCd0_du1, dgdsig*dsigma_du2])

def form_hlrf(des, idx0, tol=1e-5, Nmax=60):
    # HL-RF in u-space with chain-rule gradient (no clipping)
    u = np.zeros(2)
    for _ in range(Nmax):
        g = g_of_u(des, u, idx0)
        grad = grad_g_u(des, u, idx0); ng = np.linalg.norm(grad)
        if ng < 1e-12:  # insensitive
            beta = np.inf if g>0 else -np.inf; pf = 0.0 if g>0 else
                1.0
            return float(beta), float(pf), u.copy(), grad.copy()
        alpha = grad/ng; beta = -g/ng
        u_new = beta*alpha
        if np.linalg.norm(u_new-u) < tol: u = u_new; break
        u = u_new
    return float(np.linalg.norm(u)), float(norm.cdf(-np.linalg.norm(u
        ))), u.copy(), grad.copy()
```

SORA step and loop

Listing 4.3: Equivalent points and deterministic subproblem

```python
def sora_equivalent_point(des, idx0, beta_target):
    # u* = -beta_target * alpha at u=0
    grad = grad_g_u(des, np.zeros(2), idx0); ng = np.linalg.norm(grad
        )
    if ng < 1e-12: return np.zeros(2)
    return -beta_target * (grad/ng)

def sora_iteration(u_eq_points, x0):
    # Map each u*_i to physical (Cd0, sigma)
    z_eq = {i: from_u_to_physical(u, clip=False) for i,u in
        u_eq_points.items()}
```

```
    def objective(x):
        des = Design(float(x[0]), float(x[1]), float(x[2]))
        st  = sizing_model(des)
        return st.mass + 0.5*des.extra_m  # small penalty to avoid
            overshoot
    def cons_fun(x):
        des = Design(float(x[0]), float(x[1]), float(x[2]))
        g1 = limit_states(des, z_eq[0])[0]; g2 = limit_states(des,
            z_eq[1])[1]
        g3 = limit_states(des, z_eq[2])[2]
        return np.array([g1,g2,g3],float)
    # SLSQP with bounds and inequality constraints
    ...
    return res.x, float(res.fun)

def run_sora(max_iter=8, x0=np.array([0.18, 45.0, 0.08])):
    u_eq = {0:np.zeros(2), 1:np.zeros(2), 2:np.zeros(2)}
    betas= {0:BETA_STRUCT, 1:BETA_STRUCT, 2:BETA_POWER}
    x = x0.copy(); hist = {'x':[], 'mass':[], 'u_eq':[]}
    for k in range(max_iter):
        x_opt, m_opt = sora_iteration(u_eq, x)
        des = Design(float(x_opt[0]), float(x_opt[1]), float(x_opt
            [2]))
        hist['x'].append(x_opt.copy()); hist['mass'].append(m_opt)
        hist['u_eq'].append({i:u.copy() for i,u in u_eq.items()})
        u_eq = {i: sora_equivalent_point(des, i, betas[i]) for i in
            (0,1,2)}
        if np.linalg.norm(x_opt - x) < 1e-3: break
        x = x_opt
    return {'history':hist, 'final_design':des, 'final_state':
        sizing_model(des), 'u_eq':u_eq}
```

Reliability estimators used for validation

Directional Sampling (probability-weighted). For a random unit direction d, we find the radius $b^*(d)$ where $g_i(b\,d) = 0$; along that ray the failure probability is $1-\Phi(b^*)$. Averaging across directions estimates p_f and its CoV.

Listing 4.4: Probability-weighted DS (per limit)

```
def ds_weighted(des, N=1000, limits=[0,1,2]):
    weights = {i: [] for i in limits}
    for _ in range(N):
        d = np.random.randn(2); d /= np.linalg.norm(d)
        for i in limits:
            # grow bracket until failure then bisect
            ...
            b_star = bH
            weights[i].append(1.0 - norm.cdf(b_star))
    pf = np.zeros(3); cov = np.zeros(3)
    for i in limits:
```

```
        w = np.array(weights[i]); pf[i] = w.mean()
        var_i = w.var(ddof=1)/max(N,1)
        cov[i] = np.sqrt(var_i)/pf[i] if pf[i]>0 else 0.0
    return pf, cov
```

Importance Sampling (centered at MPP). Standard IS around each limit's FORM MPP with masked COV and a 95% upper bound when zero failures are seen.

Listing 4.5: Importance sampling (per limit)

```
def importance_sampling(des, N=8000):
    pf = np.zeros(3); var = np.zeros(3); ub95 = np.zeros(3)
    u_mpp = [form_hlrf(des, j)[2] for j in range(3)]   # design
        points
    for j in range(3):
        u0 = u_mpp[j]; w_sum = w2_sum = 0.0; nf = 0
        for _ in range(N):
            u = u0 + np.random.randn(2)  # N(u0, I)
            w = np.exp(0.5*np.dot(u-u0,u-u0) - 0.5*np.dot(u,u))
            if g_of_u(des, u, j) <= 0: nf += 1; w_sum += w; w2_sum +=
                w*w
        pf[j] = w_sum/N; var[j] = max(w2_sum/N - pf[j]**2, 0.0)
        ub95[j] = 3.0/N if nf==0 else np.nan
    cov = np.zeros_like(pf); mask = pf>0; cov[mask] = np.sqrt(var[
        mask])/pf[mask]
    return pf, cov, ub95
```

4.4 Results

SORA iteration trace (final run)

```
[Iter 1] r=0.150 m, V=65.4 m/s, extra_m=0.000, mass=4.909 kg
          g(mean) = [DL: 79.68, BL: 0.101, Energy: 0.0] J
[Iter 2] r=0.150 m, V=77.2 m/s, extra_m=0.328, mass=6.587 kg
          g(mean) = [DL: 21.47, BL: 0.088, Energy: 399717.2] J
[Iter 3] r=0.150 m, V=77.2 m/s, extra_m=0.328, mass=6.587 kg
          g(mean) = [DL: 21.47, BL: 0.088, Energy: 399717.2] J
```

The algorithm increases V_∞ and adds energy reserve `extra_m` until energy reliability is comfortably satisfied; r remains at its lower bound because blade/disk limits already have large margins.

Final design and reliability

Consistency and targets.

- **Energy is now safe:** $\beta_{\text{FORM}} \approx 4.22$ ($p_f \sim 1.2\times10^{-5}$) and DS-weighted $\hat{p}_f \sim 1.2\times10^{-4}$ with 7% CoV. Both satisfy the 10^{-3} target with ample margin.

Table 4.1: *Final SORA design and loads.*

Variable	Value	Unit
r	0.150	m
V_∞	77.2	m/s
`extra_m`	0.328	–
Mass	6.587	kg
T/rotor	16.15	N
Disk loading T/A	228.5	N/m^2
C_T	0.0067	–

Table 4.2: *Reliability at the final design.*

Limit	FORM β	FORM p_f	DS-weighted p_f (CoV)	IS p_f (UB95)
Disk load	∞	0	0 (−−)	0 (3.75×10^{-4})
Blade load	8.319	4.42×10^{-17}	3.55×10^{-18} (0.17)	0 (3.75×10^{-4})
Energy	4.216	1.24×10^{-5}	1.16×10^{-4} (0.07)	0 (3.75×10^{-4})

- **Blade/disk are extremely safe:** $\beta \gg \beta_{\text{struct}}$; DS-weighted/IS corroborate negligible risk.
- **IS zeros:** At $N = 8000$ we saw zero weighted failures, so IS prints 0 and reports a conservative 95% upper bound $3/N = 3.75 \times 10^{-4}$.

FORM design point for energy

The MPP in u-space is $u^*_{\text{energy}} = [-3.61, -2.179]$ with $\beta \approx 4.216$. Negative components mean the MPP lies in the *lower-tail* of both C_{d_0} and σ under our transform; the sign depends on the gradient convention and has no physical meaning by itself. What matters is $\beta = \|u^*\|$ and $g(u^*) \approx 0$.

4.5 Discussion and Comparison to Chapters 2–3

- In Chapter 3 the baseline deterministic design sat near the *energy* boundary ($p_f \sim 50\%$ at `extra_m` = 0). SORA responds by raising `extra_m` to ≈ 0.33 (and V_∞ to 77.2 m/s), which increases mass but drives energy risk well below the 10^{-3} target.
- Blade loading remains non-critical in this regime; pushing r above its minimum would add weight with little reliability benefit.
- If desired, a slightly stronger penalty on `extra_m` or a tighter upper bound could land closer to the *target* β (e.g., $\beta \approx 3.1$ for energy) and shave mass.
- For reporting, increasing DS directions to $N = 10\,000$ would tighten the energy COV to ≈ 2–3%.

4.6 Optional Plain Monte Carlo (Energy Only)

For completeness, we include a vectorized MC estimator for the energy limit; this can be run once at the final design as a belt-and-braces validation.

Listing 4.6: Plain Monte Carlo for energy (optional)

```python
def mc_energy(des: Design, N: int = 100000) -> float:
    U = np.random.randn(N, 2)
    Cd0 = np.exp(CD0_LN.mu_log   + CD0_LN.sig_log   * U[:,0])
    sig = np.exp(SIGMA_LN.mu_log + SIGMA_LN.sig_log * U[:,1])
    st = sizing_model(des); r, V, A = des.r, des.V, st.A
    mu = V/(OMEGA*r)
    Pcr = (sig*Cd0/8.0) * (1.0 + 4.65*mu*mu) * RHO * A * (OMEGA**3) *
        (r**3)
    Ereq = st.P_hover_per_rotor*4*THOVER + 4*Pcr*(RANGE/V)
    g3 = st.E_use - Ereq
    return float(np.mean(g3 <= 0))
```

4.7 A Critical Reflection

How do your optimal parameters change if the uncertainty in drag increases 20%? We interpret "uncertainty in drag increases by 20%" as a relative increase in the *coefficient of variation* (CoV) of the zero-lift drag coefficient:

$$\mathrm{CoV}[C_{d_0}]:\ 0.20 \ \longrightarrow\ 0.24.$$

For a lognormal variable $X = \exp(\mu_{\ln} + s\,U)$ with $U \sim \mathcal{N}(0,1)$, the log–standard deviation is

$$s = \sqrt{\ln\big(1 + \mathrm{COV}^2\big)}.$$

Thus, for C_{d_0} the transformation scales from

$$s_C^{\text{base}} = \sqrt{\ln(1+0.20^2)} \approx 0.198 \qquad \longrightarrow \qquad s_C^{+20\%} = \sqrt{\ln(1+0.24^2)} \approx 0.237,$$

which thickens the right tail of C_{d_0} and increases dispersion in cruise profile power.

Qualitative impact on the SORA optimum

Energy is the active reliability driver near the deterministic design (Tasks 1–3). Increasing only the *drag* uncertainty affects primarily the energy limit $g_3 = E_{\text{use}} - E_{\text{req}}$; disk/blade limits are almost unchanged.

- **Cruise speed V_∞:** stays essentially *unchanged.* The energy-optimal speed that minimizes cruise leg energy follows

  $$E_{\text{cruise}} \propto \frac{R}{V} + \kappa\,V \ \ \Rightarrow\ \ V^\star = \frac{\Omega r}{\sqrt{4.65}},$$

  which is independent of C_{d_0} and σ (they factor out), so the SORA solution continues to sit near $V^\star$.

- **Rotor radius r:** remains at its *lower bound*. Increasing r raises profile power (via $A\,r^3$ with fixed Ω), which hurts the energy margin while structural reliabilities already have large safety margins.
- **Energy reserve `extra_m`:** *increases* to buy back energy reliability (larger battery $\Rightarrow$ larger E_{use}). In our baseline run `extra_m` ≈ 0.328; with +20% CoV on C_{d_0} SORA typically pushes `extra_m` upward, often to the cap if the target is tight.
- **Mass:** *increases* accordingly due to the heavier battery.

Table 4.3: *Direction of change in optimal parameters with +20% drag CoV (qualitative).*

Parameter	**Baseline**	+20% **COV on C_{d_0}**	**Reason**
r	0.150 m	$\approx$ same	Structural safe; larger r worsens energy
V_∞	≈ 77 m/s	$\approx$ same	$V^\star = \Omega r/\sqrt{4.65}$ (independent of C_{d_0})
`extra_m`	0.328	$\uparrow$	More dispersion in E_{req} requires more reserve
Mass	~ 6.6 kg	$\uparrow$	Heavier battery for reliability target

Feasibility note. If `extra_m` saturates at its upper bound and the energy target ($\beta_{\text{power}} \geq 3.09$) is still unmet, viable levers are: (i) allow a slightly higher reserve cap, or (ii) gently restrict the upper speed bound around $V^\star$ to avoid drifting to higher V that amplifies the μ^2 term in cruise power. Increasing r is generally counter-productive for energy with fixed Ω.

Chapter 5

Epistemic Uncertainty Quantification

5.1 What is Epistemic UQ?

Uncertainty quantification (UQ) separates two fundamentally different kinds of uncertainty:

- **Aleatory** (randomness you *cannot* reduce): flight-to-flight variability. Here we model C_{d_0} and σ as lognormal with fixed coefficients of variation (CoV): $\mathrm{CoV}[C_{d_0}] = 0.20$, $\mathrm{CoV}[\sigma] = 0.12$.
- **Epistemic** (lack of knowledge you *can* reduce): bias or dispersion due to limited data or model-form choices. In this task we vary two epistemics:

$$b_{C_{d0}} \in [0.8, 1.2] \quad \text{(drag mean bias factor)}, \qquad m_w \in [4, 12] \quad \text{(Weibull shape for blade fatigue)}. \tag{5.1}$$

Goal: quantify how these epistemics move the *mission* probability of failure p_f^{mission}, and decide whether to prioritize **data collection** (reduce epistemic uncertainty) or **design changes**.

5.2 Modeling: Inputs, Outputs, and Assumptions

We evaluate at the final SORA design (Chapter 4): $r = 0.150\,\mathrm{m}$, $V_\infty = 77.2\,\mathrm{m/s}$, `extra_m`= 0.328.

Aleatory transforms (kept from Chapter 3). For a lognormal $X = \exp(\mu_{\ln} + s\,U)$ with $U \sim \mathcal{N}(0, 1)$ and CoV,

$$s = \sqrt{\ln\!\big(1 + \mathrm{CoV}^2\big)}, \qquad \mu_{\ln} = \ln(\text{mean}) - \tfrac{1}{2}s^2. \tag{5.2}$$

Energy failure probability. At fixed design x, given epistemic $b_{C_{d0}}$ we *shift only the mean* of C_{d_0} (CoV fixed) and compute $p_f^{(\text{energy})}$ by FORM (HL–RF) in standard normal space u:

$$p_f^{(\text{energy})} = \Phi(-\beta_{\text{energy}}). \tag{5.3}$$

The limit state is $g_3 = E_{\text{use}} - E_{\text{req}}(C_{d0}, \sigma)$, where E_{req} includes hover and cruise (profile-power model with $\mu = V_\infty/(\Omega r)$). The gradient uses a *chain rule*:

$$\frac{\partial g_3}{\partial u_1} = \frac{\partial g_3}{\partial C_{d0}}\,(s_C C_{d0}), \qquad \frac{\partial g_3}{\partial u_2} = \frac{\partial g_3}{\partial \sigma}\,(s_\sigma \sigma). \tag{5.4}$$

Blade-fatigue probability. We use a Weibull per-flight model in the normalized load ratio

$$u = \frac{(C_T/\sigma_{\text{mean}})}{BL_{\max}} \quad \text{(evaluated at the design)}, \qquad p_f^{(\text{blade})} = 1 - \exp\Big[-(u/\theta)^{m_w}\Big], \tag{5.5}$$

with θ calibrated so that at $u = 1$ and $m_{\text{ref}} = 8$ the per-flight failure probability is 10^{-6}.

Mission probability of failure. Assuming independence between energy and fatigue (distinct drivers),

$$p_f^{\text{mission}} = 1 - \big(1 - p_f^{(\text{energy})}\big)\big(1 - p_f^{(\text{blade})}\big).$$

Disk-load risk is negligible at this design and omitted.

5.3 Global Sensitivity via Sobol Indices

The question to be answered is how much of the variance of $Y = p_f^{\text{mission}}$ comes from each epistemic input $X = [b_{C_{d_0}}, m_w]$. We estimate first-order S_1^i (main effect) and total-effect S_T^i (main+interactions):

$$S_1^i = \frac{\mathrm{Var}\big(\mathbb{E}[Y \mid X_i]\big)}{\mathrm{Var}(Y)}, \qquad S_T^i = 1 - \frac{\mathrm{Var}\big(\mathbb{E}[Y \mid X_{\sim i}]\big)}{\mathrm{Var}(Y)}. \tag{5.6}$$

We use Saltelli sampling over $[0.8, 1.2] \times [4, 12]$.

5.3.1 Key Code Snippets

Lognormal parameterization from mean & CoV

Listing 5.1: Lognormal parameters from (mean, CoV).

```python
@dataclass
class LogNormalParam: mu_log: float; sig_log: float

def logn_from_mean_cov(mean: float, cov: float) -> LogNormalParam:
    s2 = np.log(1.0 + cov**2)
    return LogNormalParam(mu_log=np.log(mean) - 0.5*s2, sig_log=np.
        sqrt(s2))
```

FORM ingredients for energy limit (chain rule)

Listing 5.2: Energy limit in u-space: $g_3(u)$ and $\nabla_u g_3(u)$ (sketch).

```python
def g_energy_of_u(des, u, cd_ln, sig_ln):
    Cd0   = np.exp(cd_ln.mu_log  + cd_ln.sig_log  * u[0])
```

```python
    sigma = np.exp(sig_ln.mu_log + sig_ln.sig_log * u[1])
    g = limit_states(des, Cd0, sigma)[2]   # E_use - E_req
    return float(g)

def grad_g_energy_u(des, u, cd_ln, sig_ln):
    Cd0   = np.exp(cd_ln.mu_log  + cd_ln.sig_log  * u[0])
    sigma = np.exp(sig_ln.mu_log + sig_ln.sig_log * u[1])
    # central FD in physical space
    rel=1e-2; dC=max(1e-12, rel*Cd0); ds=max(1e-12, rel*sigma)
    dgdC = (limit_states(des, Cd0+dC, sigma)[2] - limit_states(des,
        Cd0-dC, sigma)[2])/(2*dC)
    dgds = (limit_states(des, Cd0, sigma+ds)[2] - limit_states(des,
        Cd0, sigma-ds)[2])/(2*ds)
    # chain rule to u-space
    return np.array([dgdC * (cd_ln.sig_log*Cd0),  dgds * (sig_ln.
        sig_log*sigma)])
```

Blade-fatigue Weibull model.

Listing 5.3: Per-flight fatigue probability from normalized load ratio u.

```python
def blade_pf_weibull(des, m_w, m_ref=8.0, p0_at_u1=1e-6):
    st = sizing_model(des, Cd0_nom=CD0_MEAN, sigma_nom=SIGMA_MEAN)
    u = (st.CT / SIGMA_MEAN) / BL_MAX       # normalized load ratio
    theta = (-np.log(1.0 - p0_at_u1))**(-1.0/m_ref)
    return float(1.0 - np.exp(- (u/theta)**m_w ))
```

Mission failure and Sobol wrapper.

Listing 5.4: Compose mission p_f; evaluate on Sobol samples.

```python
def mission_pf(des, b_Cd0, m_w):
    # shift mean of Cd0 by epistemic bias; keep COV the same
    cd_ln = logn_from_mean_cov(mean=b_Cd0*CD0_MEAN, cov=COV_CD0)
    pf_en = form_energy_pf(des, cd_ln)    # HL-RF using g_energy_of_u
        , grad_g_energy_u
    pf_bl = blade_pf_weibull(des, m_w)
    return 1.0 - (1.0 - pf_en) * (1.0 - pf_bl)

# SALib: sample over [0.8,1.2] x [4,12], evaluate Y = mission_pf(...)
# then sobol.analyze(...) yields S1 and ST.
```

5.4 Results and Interpretation

A representative run produced

$$S_1 = [0.694,\ 0.303], \qquad S_T = [0.697,\ 0.306], \qquad \overline{p_f^{\text{mission}}} \approx 1.447 \times 10^{-5},$$

for parameters $[b_{C_{d_0}},\, m_w]$.

- **Dominant driver: drag mean bias.** About 70% of the variance in mission risk is due to $b_{C_{d_0}}$, which shifts the mean C_{d_0} and thus cruise power (energy risk dominates overall).
- **Fatigue shape matters but less.** m_w contributes ∼30%: when m_w is small (e.g. 4), fatigue probability increases noticeably; for larger m_w it falls off rapidly.
- **Minimal interactions.** $S_T \approx S_1$ for both inputs ⇒ near-additive effects (energy and fatigue act through different mechanisms).
- **Mean level.** The mean mission risk across the epistemic box is close to the energy FORM value at the final design (order 10^{-5}), consistent with energy being the main contributor.

Data Collection vs. Design Changes? If you're near a requirement boundary: prioritize drag data (polars, CFD/tunnel) over geometry changes, because $b_{C_{d_0}}$ dominates the variance. If m_w dominates in a different regime (high normalized blade load), prioritize fatigue characterization (S–N testing).

5.5 Key Remarks

5.5.1 Assumptions and Limitations

- **Energy p_f via FORM.** We used FORM to keep the Sobol sweep fast. For absolute p_f calibration at a few anchor points you can cross-check with DS/IS.
- **Independence.** Mission p_f assumes independence between energy and fatigue failure events (reasonable here).
- **Epistemic ranges.** The boxes $b_{C_{d_0}} \in [0.8, 1.2]$, $m_w \in [4, 12]$ are engineering choices and should be tightened as data becomes available.

5.5.2 Bayesian Neural Networks

Bayesian neural networks (BNNs) represent model parameters with priors and return predictive posteriors, which *can* separate epistemic from aleatory (with care). However, for this problem they are not the right tool:

- **No surrogate needed.** Our physics model is fast and low-dimensional (two epistemics). Sobol+FORM/IS gives answers without training a surrogate.
- **Data scarcity.** BNNs need many labeled samples spanning the epistemic space; here we have limited data but good physics. Posteriors would be prior-dominated.
- **Tail calibration.** Reliability hinges on rare-event tails ($\beta \sim$ 3–4). Common BNN approximations (e.g. variational inference) are often miscalibrated in the tails, yielding over-confident/optimistic p_f.
- **Interpretability.** Sobol indices and FORM points are auditable and easy to explain; BNN priors/posteriors are harder to defend in certification contexts.

BNNs make sense when the simulator is *expensive* (CFD/FEA), the dimension is high, and there is *ample data*. Here, classical global sensitivity with physics-based reliability is simpler, more transparent, and more reliable for decision-making.