User Requirement Specification (URS)

Project: Sample Size Estimator (Python/Streamlit) **Version:** 1.0 **Classification:** QMS Software (ISO/TR 80002-2)

1. Introduction

The system is a Python-based web application designed to determine statistically valid sample sizes for medical device design verification and process validation. It utilizes the Streamlit framework to provide a user-friendly interface for advanced stochastic models. The application is critical for Quality Management Systems (QMS) to ensure compliance with risk-based statistical rationale requirements.

2. Functional Requirements (Statistical Engine)

2.1 Module A: Attribute Data Analysis

Context: Used for binary data (Pass/Fail, Go/No-Go) derived from the Binomial Distribution.

ID	Requirement Description	Acceptance Criteria / Formula Logic
URS-FUNC_A-01	The system shall accept user inputs for **Confidence (C)**, **Reliability (R)**, and optionally **Allowable Failures (c)**.	Inputs validated: $0 < C < 1$, $0 < R < 1$, $c \ge 0$ (integer).
URS-FUNC_A-02	If allowable failures are zero ($c=0$), the system shall calculate the minimum sample size (n) using the **Success Run Theorem**.	Formula: $n = \lceil \frac{\ln(1-C)}{\ln(R)} \rceil$.
URS-FUNC_A-03	If allowable failures are specified ($c > 0$), the system shall calculate n using the cumulative Binomial distribution.	Iteratively solve for smallest n where: $\sum_{k=0}^{c} \binom{n}{k} (1-R)^k R^{n-k} \le 1-C$.

URS-FUNC_A-04	**Sensitivity Analysis:** If the user leaves the Allowable Failures (c) input empty, the system shall automatically calculate and display sample sizes for $c=0, 1, 2, 3$.	Output must be a table with two columns: c (0 to 3) and required n.
URS-FUNC_A-05	If the user inputs a specific value for c, the system shall calculate the sample size only for that specific value.	Output is a single integer value n for the defined c.

2.2 Module B: Variable Data Analysis (Parametric)

Context: Used for continuous measurements assuming Normal Distribution.

ID	Requirement Description	Acceptance Criteria / Formula Logic
URS-FUNC_B-01	The system shall calculate the **One-Sided Tolerance Factor (k_1)** using the Non-Central t-Distribution.	Formula: $k_1 = \frac{t_{n-1, 1-\alpha}(\delta)}{\sqrt{n}}$ where $\delta = z_R\sqrt{n}$.
URS-FUNC_B-02	The system shall calculate the **Two-Sided Tolerance Factor (k_2)** using the Howe-Guenther Approximation.	Formula: $k_2 = \sqrt{\frac{(1 + 1/n) z^2_{(1+R)/2} (n-1)}{\chi^2_{1-C, n-1}}}$.
URS-FUNC_B-03	The system shall compute Upper (U_{tol}) and/or Lower (L_{tol}) Tolerance Limits based on Sample Mean ($\bar{X}$) and Standard Deviation (S).	$Limit = \bar{X} \pm (k \times S)$.

URS-FUNC_B-04	The system shall compare Tolerance Limits against User Specification Limits (LSL/USL) to determine PASS/FAIL status.	Pass if $L_{tol} \ge LSL$ and $U_{tol} \le USL$.
URS-FUNC_B-05	The system shall calculate the **Process Performance Index (P_{pk})** based on the sample statistics.	$P_{pk} = \min\left(\frac{USL - \bar{X}}{3S}, \frac{\bar{X} - LSL}{3S}\right)$. For one-sided specs, use only the relevant term.

2.3 Module C: Non-Normal Data Handling

Context: Used when data violates the normality assumption. Includes outlier detection, transformation, and non-parametric fallbacks.

ID	Requirement Description	Acceptance Criteria / Formula Logic
URS-FUNC_C-01	The system shall detect **outliers** in the raw data using the Interquartile Range (IQR) method.	Flag data points $< Q1 - 1.5 \times IQR$ or $> Q3 + 1.5 \times IQR$.
URS-FUNC_C-02	The system shall perform the **Shapiro-Wilk Test** for normality.	Return p-value. Reject normality if $p < 0.05$.
URS-FUNC_C-03	The system shall perform the **Anderson-Darling Test** for normality.	Return test statistic and critical values. Reject normality if statistic > critical value (at $\alpha=0.05$).
URS-FUNC_C-04	The system shall generate a **Probability Plot (Q-Q Plot)** visualization for the raw data.	Plot theoretical quantiles vs. ordered sample values.

URS-FUNC_C-05	The system shall offer multiple transformation modes: **Box-Cox**, **Logarithmic ($ln(x)$)**, and **Square Root ($\sqrt{x}$)**.	For Box-Cox, automatically optimize lambda (λ).
URS-FUNC_C-06	Upon transformation, the system must re-evaluate normality (via URS-FUNC_C-02/03) on the transformed dataset.	If $p > 0.05$ (on transformed data), proceed to parametric calculation.
URS-FUNC_C-07	The system shall calculate Tolerance Limits on the transformed scale and **back-transform** them to the original unit scale.	Log: $Limit_{orig} = \exp(Limit_{trans})$. Box-Cox: $Limit_{orig} = (\lambda \cdot Limit_{trans} + 1)^{1/\lambda}$.
URS-FUNC_C-08	If transformations fail, the system shall default to **Non-Parametric Analysis (Wilks' Method)**.	Limits defined by Min/Max values of the sample.

2.4 Module D: Reliability Life Testing

Context: Used for time-dependent testing (aging, fatigue).

ID	Requirement Description	Acceptance Criteria / Formula Logic
URS-FUNC_D-01	The system shall calculate required test duration/units for	Duration $\propto \chi^2_{1-C, 2(r+1)}$.

	Zero-Failure ($r=0$) demonstration.	
URS-FUNC_D-02	The system shall calculate the **Acceleration Factor (AF)** using the Arrhenius Equation.	$AF = \exp\left[\frac{E_a}{k} \left(\frac{1}{T_{use}} - \frac{1}{T_{test}}\right)\right]$.

---

3. User Interface (UI) Requirements

ID	Requirement Description	Implementation Detail
URS-UI-01	The application shall use a **Tab-based layout** to separate statistical methods.	Tabs: "Attribute (Binomial)", "Variables (Normal)", "Non-Normal Distribution", "Reliability".
URS-UI-02	The system shall provide **Contextual Tooltips** for all statistical input fields.	Hovering over "Confidence" displays: "Probability that the population parameter lies within the calculated interval (typically 95%)."
URS-UI-03	The system shall provide a dedicated **Help/Instruction Section** (Expander) at the top of the UI.	Contains brief instructions on how to select the correct module based on data type.
URS-UI-04	Inputs for Confidence and Reliability must use numeric input widgets with clearly labeled units (%).	Restrict range 0.0–100.0 (if %) or 0.0–1.0 (if fraction).

---

4. Reporting & Data Integrity Requirements

ID	Requirement Description	Acceptance Criteria

URS-REP-01	**User Calculation Report:** The system shall generate a downloadable PDF report summarizing the current session.	**Content:** Date/Time, User Inputs (C, R, c), Calculated Results ($n, k, Limits, P_{pk}$), and Method Used.
URS-REP-02	**Validation State Reference:** The User Calculation Report must display the **SHA-256 Hash** of the current calculation engine file (`calculations.py`).	The report must clearly state: "Engine Hash: [HashValue]" to prove the code has not been altered since validation.
URS-REP-03	**Integrity Check:** The User Calculation Report must compare the current Engine Hash against a stored "Validated Hash".	If hashes match: Print "VALIDATED STATE: YES". If mismatch: Print "VALIDATED STATE: NO - UNVERIFIED CHANGE".
URS-REP-04	**Automated Validation Report:** The IQ/OQ/PQ test suite must generate a self-contained PDF report ("Validation Certificate").	**Content:** Test Date, Tester Name, System Info (OS, Python Ver), List of all URS IDs tested, Pass/Fail status for each, and the final "Validated Hash" of the engine.

---

5. Automated Validation Requirements (IQ/OQ/PQ)

ID	Requirement Description	Acceptance Criteria
URS-VAL-01	**Installation Qualification (IQ):** Dependencies must be strictly version-locked using a hash-based lockfile.	`uv sync` or `pip install` must succeed without conflict. Environment check script confirms `scipy==1.x.x` version.
URS-VAL-02	**Operational Qualification (OQ):** A `pytest` suite shall verify all mathematical	Tests must carry markers linking to URS IDs (e.g., `@pytest.mark.urs("URS`

ID	Requirement Description	Acceptance Criteria
	models against known standard values.	`-FUNC_A-02")`. All tests must PASS.
URS-VAL-03	**Performance Qualification (PQ):** An automated UI test (using `Streamlit AppTest` or `Playwright`) shall simulate a user workflow.	Script simulates: Open Tab -> Enter Data -> Click Calculate -> Verify Output Text appears.

---

6. Verification Traceability Matrix (VTM) Structure

To ensure automated verification, test scripts must utilize the following mapping strategy:

Example VTM Mapping in Code:

Python

```
import pytest
from calculations import calculate_attribute_sample_size

# Testing URS-FUNC_A-04 (Sensitivity Table Logic)
@pytest.mark.urs("URS-FUNC_A-04")
def test_attribute_sensitivity_analysis():
    """
    Verifies that if c is None, the system returns results for c=0,1,2,3.
    """
    results = calculate_attribute_sample_size(confidence=0.95,
reliability=0.90, c=None)
    assert len(results) == 4
    assert results[0]['c'] == 0
    assert results[3]['c'] == 3

# Testing URS-FUNC_C-07 (Back-Transformation)
@pytest.mark.urs("URS-FUNC_C-07")
def test_log_back_transformation():
    """
    Verifies that limits calculated on log-data are correctly exponentiated
back.
    """
    # ... setup code ...
    assert calculated_limit_orig == pytest.approx(expected_limit, rel=1e-3)
```