# Loop Closure with Least Squares Algorithm - Ariane

This document describes the loop closure algorithm used in Ariane to distribute errors in cave survey loops. The implementation uses a Least Squares adjustment method, treating the survey network as a spring-mass system.

## 1. Conceptual Model

The cave survey is modeled as a graph where:

- **Vertices (Nodes)** represent survey stations (points in 3D space).
- **Edges (Links)** represent survey shots (measurements between stations).

The algorithm seeks to find the set of station coordinates $(x, y)$ that minimizes the total weighted squared error between the measured shot vectors and the calculated vectors between stations.

The problem is solved independently for the X and Y dimensions (or jointly depending on the solver implementation), assuming uncorrelated errors in X and Y. The Z component is generally handled separately or excluded from the 2D least squares adjustment in this specific implementation context.

## 2. Mathematical Formulation

The objective is to minimize the following cost function:

$$E = \sum_{(i,j) \in Edges} w_{ij} \left( (p_j - p_i) - d_{ij} \right)^2$$

Where:

- $p_i, p_j$ are the positions (coordinates) of station $i$ and $j$.
- $d_{ij}$ is the measured vector component (e.g., $\Delta x$ or $\Delta y$) from station $i$ to $j$.
- $w_{ij}$ is the weight (stiffness) of the shot.

This leads to a system of linear equations of the form $Ax = B$, where $A$ is a symmetric positive-definite matrix.

# 3. Algorithm Steps

## 3.1. Graph Construction

1. **Vertices**:
    - All stations in the survey are added as vertices.
    - Stations designated as **Start** (e.g., entrance points, GPS fixes) are marked as **FIXED**.
    - All other stations are marked as **ADJUSTABLE** (or ESTIMATED).
    - Initial positions are populated from the raw traverse calculation.
2. **Edges**:
    - Every survey shot (Real or Virtual) connects two vertices.
    - **Closure** shots (shots explicitly marked to close loops) are collapsed.

## 3.2. Weight Calculation

The “stiffness” of a shot determines how much it resists stretching or compressing. Ariane uses the inverse of the shot length as the primary weighting factor, based on the assumption that shorter shots are more accurate than long shots.

For an edge $e_{ij}$ with length $L$:

$$w_{ij} = \begin{cases} 10.0 & \text{if } L = 0 \\ \min(\frac{1.0}{L}, 10.0) & \text{if } L > 0 \end{cases}$$

*Note: Weights are clamped to a maximum of 10.0 to prevent numerical instability with extremely short shots.*

## 3.3. Matrix Assembly

The system $Ax = B$ is assembled for each dimension (X and Y). For a graph with $N$ vertices:

- **Matrix A ($N \times N$)**:
    - For each edge connecting $i$ and $j$ with weight $w_{ij}$:
        - Add $w_{ij}$ to diagonal elements $A_{ii}$ and $A_{jj}$.
        - Subtract $w_{ij}$ from off-diagonal elements $A_{ij}$ and $A_{ji}$.
- **Vector B ($N \times 1$)**:
    - Represents the “force” applied by measurements.
    - For edge $i \to j$ with measured delta $d$:

- Subtract $w_{ij} \cdot d$ from $B_i$.
- Add $w_{ij} \cdot d$ from $B_j$.

## 3.4. Applying Constraints

To fix the graph in space, constraints (Fixed Stations) must be applied. For every **FIXED** station $f$ with known position $P_{known}$:

1. Identify all neighbors $n$ connected to $f$.
2. Move the known terms to the Right Hand Side (RHS) vector $B$:
    - The term $A_{nf} \cdot x_f$ becomes a constant since $x_f$ is known ($P_{known}$).
    - Update $B_n \leftarrow B_n - A_{nf} \cdot P_{known}$.
3. Remove the row and column corresponding to $f$ from matrix $A$ and row $f$ from vector $B$.
    - *Implementation Check*: This effectively reduces the system size by the number of fixed stations.

## 3.5. Solving

The resulting linear system is predominantly sparse and symmetric positive-definite. Ariane employs two solving strategies:

1. **CUDA Solver (GPU)**:
    - Uses cuSolvers (Cholesky decomposition) for high-performance solving on compatible hardware.
    - Handles large matrices efficiently.
2. **Rust Native Solver (CPU)**:
    - Uses an iterative solver (e.g., Conjugate Gradient or similar iterative approach suitable for sparse systems) or a direct solver via the `ariane-native` library.
    - Fallback when CUDA is not available.

## 3.6. Applying Corrections

Once the global solution vectors $X_{new}$ and $Y_{new}$ are computed:

1. For every non-fixed station $i$:
    - The new position is updated: $P_i \leftarrow (X_{new}[i], Y_{new}[i])$.
    - A **Correction Vector** is stored: $\vec{C}*i* = *P*{new} - P_{original}$.

This correction vector represents the shift required to close the loops according to the least squares distribution.

# 4. Code References

- **Graph Construction & Solver Interface**: `com.arianesline.graph.Graph`
  - `solveLeastSquare()`: Main entry point.
  - `solveRustGraph()`: Bridge to native CPU solver.
  - `solveCudaPath()`: Setup for GPU solving.
- **Data Preparation**: `com.arianesline.cavelib.CaveGeo`
  - `buildGraph()`: Converts cave survey data into the graph structure.
  - `buildDisplayData()`: Manages the high-level construction workflow.