Technical Specification: Algorithmic Implementation and Software Architecture of DyProp

DyProp Development Team

December 12, 2025

1 Overview

The `dyprop` package is an R/Bioconductor extension of the `propr` ecosystem. It implements the mathematical framework of Dynamic Proportionality using a high-performance C++ backend for vectorized template matching and an R frontend for S4 class management, event classification, and hierarchical validation.

2 Data Transformation Pipeline

2.1 Zero Handling Protocol

To enable log-ratio analysis on sparse genomic data without masking biological signals (e.g., gene silencing), we implement a rigorous two-step imputation strategy[Ref:140, 141].

1. **Step A: Technical Correction (kNN Smoothing)**
   - **Objective:** To correct technical dropouts (sporadic zeros) based on manifold proximity.
   - **Implementation:** To avoid circularity (library-size bias), we cannot perform standard PCA on raw counts with zeros. We utilize **GLM-PCA** (Generalized Linear Model PCA) or perform a temporary $\log(x + 1)$ transform solely to calculate the Euclidean distance matrix.
   - **Logic:** We identify the $K = 5$ nearest neighbors in this corrected space, but perform the pooling on the **raw counts** to preserve the integer nature required for CoDa imputation.
2. **Step B: Biological Imputation (Bayesian)**
   - **Objective:** To allow log-ratios involving Biological Zeros (e.g., silenced genes) to be computed[Ref:145].
   - **Implementation:** We utilize the `zCompositions::cmultRepl` function (Count Zero Multiplicative Replacement)[Ref:146].
   - **Logic:** Zeros are replaced by a probabilistic estimate δ derived from the limit of detection, ensuring that an "On/Off" switch is detected as a large, quantifiable change in the log-ratio, rather than a missing value[Ref:147, 148].

2.2 Coordinate Projection & Ratio-First Robustness

Following imputation, the data matrix $\mathbf{X}$ is projected into Euclidean space using the Centered Log-Ratio (CLR) transformation[Ref:378]. To ensure robustness against outliers (e.g., PCR artifacts) without violating Aitchison geometry, we apply Winsorization at the ratio level:

```
# R Pseudo-code
# 1. Full CLR Transform (Preserves Subcompositional Coherence)
clr_matrix <- propr::propr(imputed_counts)@logratio

# 2. Robust Ratio Calculation (e.g., Gene A vs Gene B)
ratio_vector <- clr_matrix[, A] - clr_matrix[, B]
clean_vector <- winsorize(ratio_vector, probs = c(0.01, 0.99))
```

3 Algorithmic Engine: The Vectorized Scanner

3.1 Phase 1: Just-In-Time Dictionary Generation

To bypass the computational intractability of iterative regression for $> 10^8$ pairs, we pre-compute a dictionary of archetypal Boundary Functions[Ref:379]. Upon initialization, the C++ backend generates a template matrix $\mathcal{M}$ ($K_{archetypes} \times T$) dynamically to match the user's pseudotime resolution. The grid resolution is strictly coupled to the detection limit via the Nyquist criterion ($\Delta\tau \leq \epsilon_{min}$)[Ref:380].

3.2 Phase 2: Block-Wise Vectorized Scanner (C++ / OpenMP)

We perform a global search for topological tipping points using dense linear algebra. To prevent memory overflow (storing 10^8 pairs is infeasible), we implement a **Map-Reduce** architecture with OpenMP parallelization.

1. **Chunking:** The gene matrix is processed in blocks (e.g., $B = 1000$ genes).
2. **Stream Processing:** For each block, we compute the $\binom{B}{2}$ log-ratio trajectories on the fly.
3. **Standardization:** Trajectories are Z-scored in-place. Crucially, to ensure the matrix product yields Pearson Correlation coefficients (Cosine Similarity), both $\mathbf{Y}$ and $\mathcal{M}$ are **row-wise centered and scaled** prior to multiplication.
4. **Filtering:** We retain *only* the top 0.1% of pairs (based on fit score R) and discard the raw trajectory data from memory.
5. **Operation:** We compute the convolution via matrix multiplication using `RcppArmadillo`.

```
// src/dyprop.cpp (Memory-Safe Block Scanner)
#include <RcppArmadillo.h>
// [[Rcpp::depends(RcppArmadillo)]]

// Uses OpenMP for parallel chunk processing
arma::mat scan_block(arma::mat Y_chunk, arma::mat M_t) {
    // Y_chunk: [Pairs x Time]
    // M_t: [Time x Archetypes] (Transposed Template)

    // 1. Row-wise Z-Score (In-place)
    // Ensures dot product == Pearson Correlation
    Y_chunk.each_row([](arma::rowvec& r) {
```

```
        r = (r - arma::mean(r)) / arma::stddev(r);
    });

    // 2. Dense Matrix Multiplication
    arma::mat scores = Y_chunk * M_t;

    return scores;
}
```

3.3 Phase 3: Quadratic Interpolation

To recover continuous parameter estimates from the discrete grid, we apply Parabolic Interpolation[Ref:386].

- **Logic:** Given the best matching template index i, we fit a local parabola to scores (R_{i-1}, R_i, R_{i+1}).
- **Output:** The vertex of the parabola yields the precise continuous Tipping Point $\hat{\tau}$ and Sharpness $\hat{\epsilon}$[Ref:387].

3.4 Phase 3.5: The Empirical Null Model (FDR)

To assign statistical significance to the template match scores (R) without the prohibitive cost of permuting 10^8 pairs:

1. We permute the cell labels (pseudotime) once to destroy biological signal.
2. We run the Vectorized Scanner on a random subset of $10,000$ pairs to generate a null distribution of correlation scores R_{null}.
3. Empirical p-values for real scores R_{obs} are calculated relative to this null distribution.

3.5 Adaptive Bandwidth Selection

The kernel width h is automated based on the dynamic range of the pseudotime coordinate:

$$h_{opt} = 0.05 \times (\max(t) - \min(t))$$

4 Software Architecture

4.1 Class Structure

We define the `dyprop` S4 class to manage the complex outputs of dynamic analysis[Ref:159].

```
setClass("dyprop",
    contains = "propr",     # Inherits basic CoDa slots from parent package
    slots = c(
        pseudotime = "numeric",
        design = "matrix",

        # MEMORY EFFICIENT STORAGE (Lazy Evaluation)
        # We replace dense 3D arrays with a sparse catalog.
        # Stores only significant events (FDR < 0.05).
        events = "data.frame", # Cols: GeneA, GeneB, Type, Tau, Epsilon, Score, FDR

        # Lazy Evaluation Cache
```

```
        # Stores computed trajectories only for requested plots (LRU Cache)
        metric_cache = "environment",

        # Validation
        networks = "list",
        glmm_fits = "list",
        dictionary_meta = "list"
    )
)
```

4.2 Core Methods

`prepareComposition(counts)` Wrapper for the kNN + Imputation workflow. Returns clean CLR matrix[Ref:161].

`scanDynamics(object)` Calls the C++ backend. Populates the `events` catalog, and performs the Template Match[Ref:162] based on Vectorized Z-Score Matching.

`classifyEvents(object)` **Lazy Evaluation Step.** Computes the expensive kernel-weighted metrics (Φ, ρ) *only* for the significant candidates in the `events` slot. This reduces computational complexity from $O(p^2)$ to $O(N_{significant})$. Applies the Multi-Evidence Decision Tree to label events as "Switch" or "Decoupling"[Ref:393].

`validateGLMM(object)` Runs the hierarchical validation module on significant candidates.

5 Phase 4: Network Reconstruction & Differential Topology

To translate statistical events into biological mechanisms, we implement a dedicated module for **Differential Topology Inference**[Ref:395].

5.1 Regime Definition (Temporal Slicing)

For gene pairs identified as a **Stoichiometric Switch** (Type 1), we utilize the estimated Tipping Point $\hat{\tau}$ and Sharpness $\hat{\epsilon}$ to rigorously partition the trajectory. We explicitly exclude the "Boundary Layer" (the transition region) to prevent transition-state noise from contaminating static estimates[Ref:397].

```
# R Logic: Define Stable Regimes
get_regimes <- function(pseudotime, tau, epsilon) {
    boundary_width <- 2 * epsilon
    idx_pre <- which(pseudotime < (tau - boundary_width)) # Stable State A
    idx_post <- which(pseudotime > (tau + boundary_width)) # Stable State B
    return(list(pre = idx_pre, post = idx_post))
}
```

5.2 Differential Topology (ΔP)

We compute the static proportionality matrix $\mathbf{P}$ (Rho) separately for the Pre- and Post-transition cell subsets. We define the **Rewiring Matrix**[Ref:400]:

$$\mathbf{\Delta P} = \mathbf{P}_{\text{post}} - \mathbf{P}_{\text{pre}}$$

- **Decoupling ($\Delta P_{ij} \ll 0$):** The regulatory constraint is relaxed. The gene pair moves from a coupled state to an independent state.

- **Coupling ($\Delta P_{ij} \gg 0$):** A new regulatory constraint is established. The gene pair locks into a fixed stoichiometric ratio.

5.3 Haywire Hub Analysis

We identify driver regulators by calculating the **Differential Degree Centrality** (Δk) for each gene. A high Δk indicates a gene that has simultaneously rewired its relationship with many targets—a signature of a master regulator or a "Haywire Hub"[Ref:32].

$$\Delta k_i = \sum_{j \neq i} |\Delta P_{ij}|$$

Downstream Integration: The package exports the list of top Δk genes to facilitate motif enrichment analysis (e.g., via `RCisTarget`), linking the detected stoichiometric switch to specific Transcription Factors (e.g., *MYC*, *TP53*)[Ref:405].

6 Phase 5: Hierarchical Validation Module

To support clinical experimental designs, DyProp includes a confirmatory GLMM module utilizing `glmmTMB`. This framework is chosen over standard `lme4` to leverage the Student's t-distribution for outlier robustness and to explicitly model heteroscedasticity in pseudotime[Ref:406, 407].

- **Objective:** To separate biological trajectory dynamics from random batch/patient effects while controlling for heavy-tailed residual noise[Ref:408].
- **Input:** The subset of significant gene pairs identified in Phase 2.
- **Model:** Robust Generalized Linear Mixed Model[Ref:410].

$$y_{ijk} \sim \mathcal{T}(\mu_{ijk}, \sigma_{ij}, \nu)$$

$$\mu_{ijk} = \beta_0 + f(\text{Pseudotime}_{ij}) + \text{Condition}_k + (1|\text{Patient}_k)$$

$$\log(\sigma_{ij}) = \gamma_0 + \gamma_1 \text{Pseudotime}_{ij} \quad \text{(Dispersion Model)}$$