N-of-1 Trial Simulation Process: Narrative and Pseudocode

Executive Summary

This document describes the Monte Carlo simulation process for comparing **Hybrid N-of-1** and **Traditional Crossover** trial designs in precision medicine research. The simulation evaluates statistical power to detect biomarker-treatment interactions under various conditions, including different carryover effects.

High-Level Overview

Purpose

Compare two clinical trial designs for their ability to detect personalized treatment effects (biomarker × treatment interactions) while accounting for: - Carryover effects (residual treatment impact after discontinuation) - Temporal correlations (measurements within individuals over time) - Between-subject and within-subject variability

Key Innovation

Unlike Hendrickson et al. (2020), this simulation **explicitly models carryover effects** in both: 1. **Data generation** (via exponential decay in biological response) 2. **Statistical analysis** (via carryover covariate in mixed models)

Part 1: Study Designs

Design 1: Hybrid N-of-1 (Hendrickson 4-Path Design)

Structure: 20 weeks divided into 3 phases 1. **Weeks 1-8**: Open-label treatment (all participants on drug) 2. **Weeks 9-12**: Blinded discontinuation (randomized) 3. **Weeks 13-20**: Crossover period (randomized sequence)

4-Path Randomization (2×2 factorial): - **Factor 1**: Blinded discontinuation (stay on vs. discontinue) - **Factor 2**: Crossover sequence (drug-first vs. placebo-first)

Path	Weeks 9-12	Weeks 13-16	Weeks 17-20
A	Drug	Drug	Placebo
B	Drug	Placebo	Drug
C	Placebo	Drug	Placebo
D	Placebo	Placebo	Drug

Advantages: - Open-label period provides baseline on treatment - Blinded discontinuation tests acute withdrawal effects - Crossover provides within-subject comparison

Design 2: Traditional Crossover

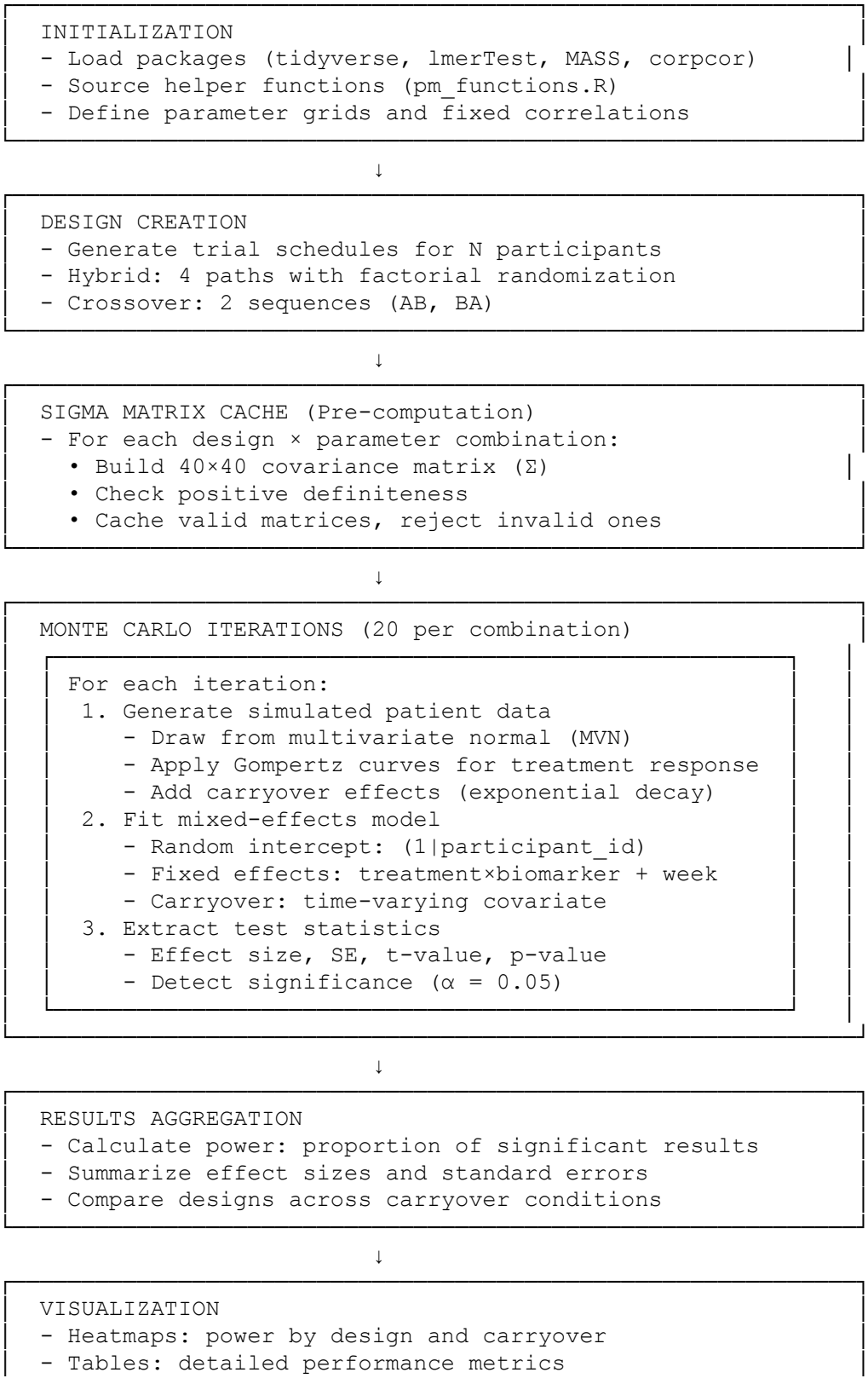
Structure: 20 weeks, 2 sequences - **AB sequence**: Weeks 1-10 on drug, weeks 11-20 on placebo - **BA sequence**: Weeks 1-10 on placebo, weeks 11-20 on drug

Simpler randomization: - Participants randomly assigned to AB or BA - Equal allocation to each sequence

---

Part 2: Simulation Architecture

Conceptual Flow



Part 3: Detailed Pseudocode

3.1 Initialization and Parameter Setup

```
FUNCTION main_simulation():
    # Clear workspace and load packages
    LOAD packages: tidyverse, lmerTest, MASS, corpcor
    SOURCE "pm_functions.R"

    # Define base parameters
    base_params = {
        n_participants: 70,
        biomarker_correlation: 0.3,
        treatment_effect: 5.0,
        carryover_t1half: 1.0,        # Half-life in weeks
        carryover_scale: 1.0,          # Scale factor for decay
        between_subject_sd: 3.0,       # Individual differences
        within_subject_sd: 2.8,        # Measurement noise
        n_iterations: 20               # Monte Carlo replications
    }

    # Define parameter grid for sensitivity analysis
    param_grid = EXPAND_GRID(
        n_participants: [70],
        biomarker_correlation: [0.2, 0.4],
        carryover_t1half: [0, 1.0, 2.0],  # None, moderate, strong
        treatment_effect: [5.0]
    )
    # Total: 1 × 2 × 3 × 1 = 6 parameter combinations
    # × 2 designs = 12 combinations
    # × 20 iterations = 240 simulation runs

    # FIXED CORRELATION STRUCTURE (Hendrickson alignment)
    # These do NOT vary with carryover (affects means only)
    model_params = {
        c.tv: 0.8,      # Time-variant autocorrelation
        c.pb: 0.8,      # Pharmacologic biomarker autocorrelation
        c.br: 0.8,      # Biological response autocorrelation
        c.cf1t: 0.2,    # Cross-correlation (same time)
        c.cfct: 0.1,    # Cross-correlation (different times)
        c.bm: varies    # Biomarker-response correlation (swept)
    }

    # Response parameters (Gompertz curve parameters)
    resp_param = {
        time_variant:     {max: 1.0, disp: 2.0, rate: 0.3, sd: 2.8},
        pharm_biomarker:  {max: 1.0, disp: 2.0, rate: 0.3, sd: 2.8},
        bio_response:     {max: 5.0, disp: 2.0, rate: 0.3, sd: 2.8}
    }

    # Baseline parameters
```

```
    baseline_param = {
        biomarker: {mean: 5.0, sd: 2.0},
        baseline:  {mean: 10.0, sd: 3.0}
    }

    CALL build_sigma_cache()
    CALL run_monte_carlo_simulations()
    CALL summarize_and_visualize()
END FUNCTION
```

3.2 Design Creation

```
FUNCTION create_designs(n_participants):
    # HYBRID N-OF-1 DESIGN
    # Randomly assign participants to 4 paths
    path_assignment = SAMPLE([1,2,3,4], n_participants, replace=TRUE)

    hybrid_design = TIBBLE()
    FOR participant_id = 1 TO n_participants:
        path = path_assignment[participant_id]

        FOR week = 1 TO 20:
            # Weeks 1-8: Open-label (all on treatment)
            IF week <= 8:
                treatment = 1
                expectancy = 1

            # Weeks 9-12: Blinded discontinuation
            ELSE IF week >= 9 AND week <= 12:
                IF path IN [1, 2]:  # Paths A, B: stay on
                    treatment = 1
                ELSE:               # Paths C, D: discontinue
                    treatment = 0
                expectancy = 0.5

            # Weeks 13-16: Crossover period 1
            ELSE IF week >= 13 AND week <= 16:
                IF path IN [1, 3]:  # Drug-first sequence
                    treatment = 1
                ELSE:               # Placebo-first sequence
                    treatment = 0
                expectancy = 0.5

            # Weeks 17-20: Crossover period 2 (switch)
            ELSE IF week >= 17 AND week <= 20:
                IF path IN [1, 3]:  # Switch to placebo
                    treatment = 0
                ELSE:               # Switch to drug
                    treatment = 1
                expectancy = 0.5

            ADD row (participant_id, week, path, treatment, expectancy)

    # TRADITIONAL CROSSOVER DESIGN
```

```
    crossover_design = TIBBLE()
    FOR participant_id = 1 TO n_participants:
        # Assign to AB (path 1) or BA (path 2)
        IF participant_id is EVEN:
            sequence = "AB", path = 1
        ELSE:
            sequence = "BA", path = 2

        FOR week = 1 TO 20:
            IF sequence == "AB":
                treatment = (week <= 10) ? 1 : 0
            ELSE:  # BA sequence
                treatment = (week <= 10) ? 0 : 1

            expectancy = 0.5  # Always blinded

            ADD row (participant_id, week, path, treatment, expectancy)

    RETURN {hybrid: hybrid_design, crossover: crossover_design}
END FUNCTION
```

3.3 Sigma Matrix Cache Construction

```
FUNCTION build_sigma_cache(param_grid, designs):
    """
    Pre-compute covariance matrices for all valid combinations
    to avoid redundant computation during Monte Carlo iterations
    """
    sigma_cache = EMPTY_MAP()
    valid_combinations = EMPTY_TIBBLE()

    FOR each row in param_grid:
        params = EXTRACT_PARAMETERS(row)

        # Update model parameters (correlations stay FIXED)
        current_model_params = model_params
        current_model_params.N = params.n_participants
        current_model_params.c.bm = params.biomarker_correlation
        current_model_params.carryover_t1half = params.carryover_t1half

        FOR design_name in ["hybrid", "crossover"]:
            # Get design template for first participant
            trial_design = GET_TEMPLATE(designs[design_name])

            # Build 40×40 covariance matrix
            sigma_result = build_sigma_matrix(
                model_param = current_model_params,
                resp_param = resp_param,
                baseline_param = baseline_param,
                trial_design = trial_design
            )

            # Check positive definiteness
            IF sigma_result IS NOT NULL:
```

```
                eigenvalues = EIGENVALUES(sigma_result.sigma)
                min_eigenvalue = MIN(eigenvalues)
                max_eigenvalue = MAX(eigenvalues)
                condition_number = max_eigenvalue / min_eigenvalue

                is_positive_definite = (min_eigenvalue > 0)

                IF is_positive_definite:
                    # Cache valid matrix
                    cache_key = CREATE_KEY(design_name, params)
                    sigma_cache[cache_key] = sigma_result.sigma

                    PRINT "✓ Valid sigma for", design_name, params

                    # Track valid combination
                    ADD_ROW(valid_combinations, {
                        design: design_name,
                        biomarker_correlation: params.biomarker_correlation,
                        carryover_t1half: params.carryover_t1half
                    })

                    IF condition_number > 100:
                        WARN "Matrix is ill-conditioned (CN =",
                             condition_number, ")"
                ELSE:
                    PRINT "✗ REJECTED: Non-PD matrix for", design_name, params
                    PRINT "  Eigenvalue range: [", min_eigenvalue, ",",
                          max_eigenvalue, "]"
            ELSE:
                PRINT "✗ REJECTED: build_sigma_matrix returned NULL"

    PRINT "\nSigma cache built. Valid combinations:",
          NROW(valid_combinations)
    PRINT valid_combinations

    RETURN sigma_cache
END FUNCTION
```

3.4 Sigma Matrix Construction Details

```
FUNCTION build_sigma_matrix(model_param, resp_param, baseline_param,
                            trial_design):
    """
    Build 40×40 covariance matrix for 2 baselines + 38 timepoints
    Structure: Σ = [Σ_baseline    Σ_cross    ]
                   [Σ_cross^T     Σ_timepoints]
    """

    # 1. BASELINE COVARIANCE (2×2)
    sigma_baseline = DIAGONAL_MATRIX([
        baseline_param["biomarker"].sd^2,
        baseline_param["baseline"].sd^2
    ])
```

```
# 2. TIMEPOINT COVARIANCE (38×38)
# 38 timepoints = 20 weeks × 2 factors - 2 constants
# Factors: time_variant (tv), pharm_biomarker (pb), bio_response (br)

n_weeks = 20
n_timepoints = n_weeks * 2  # tv and pb (br varies with treatment)
sigma_timepoints = EMPTY_MATRIX(n_timepoints, n_timepoints)

# Build correlation structure
FOR i = 1 TO n_timepoints:
    FOR j = 1 TO n_timepoints:
        week_i = CEILING(i / 2)
        week_j = CEILING(j / 2)
        factor_i = GET_FACTOR_TYPE(i)  # tv or pb
        factor_j = GET_FACTOR_TYPE(j)

        # Determine correlation based on factor types and time lag
        IF factor_i == factor_j:
            # Same factor, different times: autocorrelation
            IF week_i == week_j:
                correlation = 1.0
            ELSE:
                time_lag = ABS(week_i - week_j)
                IF factor_i == "tv":
                    correlation = model_param.c.tv^time_lag
                ELSE IF factor_i == "pb":
                    correlation = model_param.c.pb^time_lag
        ELSE:
            # Different factors
            IF week_i == week_j:
                # Same time: moderate cross-correlation
                correlation = model_param.c.cf1t
            ELSE:
                # Different times: weak cross-correlation
                correlation = model_param.c.cfct

        # Convert to covariance
        sd_i = GET_SD(factor_i, resp_param)
        sd_j = GET_SD(factor_j, resp_param)
        sigma_timepoints[i,j] = correlation * sd_i * sd_j

# 3. CROSS-COVARIANCE between baselines and timepoints (2×38)
sigma_cross = COMPUTE_CROSS_COVARIANCE(
    baseline_param, resp_param, model_param
)

# 4. ASSEMBLE FULL MATRIX
sigma = BLOCK_MATRIX([
    [sigma_baseline,        sigma_cross],
    [TRANSPOSE(sigma_cross), sigma_timepoints]
])

# 5. VALIDATE POSITIVE DEFINITENESS
eigenvalues = EIGEN(sigma).values
```

```
    is_pd = ALL(eigenvalues > 0)

    IF NOT is_pd:
        PRINT "REJECTED: Non-positive definite matrix detected"
        PRINT "Eigenvalue range: [", MIN(eigenvalues), ",",
              MAX(eigenvalues), "]"
        RETURN NULL

    RETURN {sigma: sigma, is_pd: TRUE}
END FUNCTION
```

3.5 Data Generation (Core Monte Carlo Step)

```
FUNCTION generate_data(model_param, resp_param, baseline_param,
                       trial_design, seed, scale_factor, cached_sigma):
    """
    Generate simulated patient data for one iteration
    Uses multivariate normal distribution with pre-cached Σ
    """

    SET_SEED(seed)
    n_participants = model_param.N
    n_weeks = 20

    # 1. DRAW FROM MULTIVARIATE NORMAL
    # Each participant gets 40 correlated values:
    #   - 2 baseline values (biomarker, baseline)
    #   - 38 timepoint values (20 weeks × varying factors)

    simulated_data = MVN_SAMPLE(
        n = n_participants,
        mu = VECTOR_OF_ZEROS(40),
        Sigma = cached_sigma
    )

    # Extract baseline values (columns 1-2)
    baseline_biomarker = simulated_data[, 1]
    baseline_response = simulated_data[, 2]

    # Extract timepoint values (columns 3-40)
    timepoint_values = simulated_data[, 3:40]

    # 2. APPLY TREATMENT EFFECTS using Gompertz curves
    patient_data = EMPTY_TIBBLE()

    FOR participant_id = 1 TO n_participants:
        participant_schedule = FILTER(trial_design,
                                      participant_id == participant_id)

        FOR week = 1 TO n_weeks:
            treatment = participant_schedule$treatment[week]

            # Get raw simulated values for this timepoint
            tv_raw = timepoint_values[participant_id, week*2 - 1]
```

```
pb_raw = timepoint_values[participant_id, week*2]

# Apply Gompertz growth curve for treatment response
IF treatment == 1:
    tv_effect = mod_gompertz(
        time = week,
        max_value = resp_param["time_variant"].max,
        displacement = resp_param["time_variant"].disp,
        rate = resp_param["time_variant"].rate
    )

    pb_effect = mod_gompertz(
        time = week,
        max_value = resp_param["pharm_biomarker"].max,
        displacement = resp_param["pharm_biomarker"].disp,
        rate = resp_param["pharm_biomarker"].rate
    )

    # Biomarker × treatment interaction
    br_effect = mod_gompertz(
        time = week,
        max_value = resp_param["bio_response"].max,
        displacement = resp_param["bio_response"].disp,
        rate = resp_param["bio_response"].rate
    ) * (1 + model_param.c.bm * baseline_biomarker[participant_id])

ELSE:
    tv_effect = 0
    pb_effect = 0
    br_effect = 0

# 3. ADD CARRYOVER EFFECTS
# Calculate time since last discontinuation
IF week > 1:
    prev_treatment = participant_schedule$treatment[week-1]
    IF prev_treatment == 1 AND treatment == 0:
        # Just discontinued
        time_since_disc = 1
    ELSE IF treatment == 0:
        # Still off treatment
        time_since_disc = time_since_disc + 1
    ELSE:
        time_since_disc = 0
ELSE:
    time_since_disc = 0

# Apply exponential decay to biological response
IF time_since_disc > 0 AND model_param.carryover_t1half > 0:
    # Carryover only affects biological response (br)
    # Previous effect decays exponentially
    carryover_multiplier = (0.5)^(scale_factor *
                                  time_since_disc /
                                  model_param.carryover_t1half)
```

```
                # Get the effect that would have been present
                prev_max_effect = resp_param["bio_response"].max *
                                  (1 + model_param.c.bm *
                                   baseline_biomarker[participant_id])

                carryover_contribution = prev_max_effect *
                                         carryover_multiplier
                br_effect = br_effect + carryover_contribution

            # 4. COMBINE COMPONENTS
            response = baseline_response[participant_id] +
                       tv_raw + tv_effect +
                       pb_raw + pb_effect +
                       br_effect

            # Store data
            ADD_ROW(patient_data, {
                participant_id: participant_id,
                week: week,
                biomarker: baseline_biomarker[participant_id],
                baseline: baseline_response[participant_id],
                treatment: treatment,
                response: response,
                time_since_disc: time_since_disc
            })

    RETURN patient_data
END FUNCTION
```

3.6 Monte Carlo Iteration and Analysis

```
FUNCTION run_monte_carlo(design_name, params, sigma_cache):
    """
    Run multiple iterations for a single design/parameter combination
    """

    # Get cached sigma matrix
    cache_key = CREATE_KEY(design_name, params)
    cached_sigma = sigma_cache[cache_key]

    IF cached_sigma IS NULL:
        PRINT "Skipping", design_name, "- invalid sigma matrix"
        RETURN EMPTY_TIBBLE()

    # Get design structure
    IF design_name == "hybrid":
        full_design = designs.hybrid.full_design
        design_paths = designs.hybrid.design_paths
    ELSE:
        full_design = designs.crossover.full_design
        design_paths = designs.crossover.design_paths

    n_iterations = params.n_iterations
    all_results = EMPTY_TIBBLE()
```

```
# Run Monte Carlo iterations
FOR iteration = 1 TO n_iterations:

    # STEP 1: GENERATE DATA for all paths
    all_path_data = EMPTY_TIBBLE()

    FOR path_id = 1 TO LENGTH(design_paths):
        # Get participants in this path
        path_participants = FILTER(full_design, path == path_id) %>%
                            UNIQUE(participant_id)
        n_in_path = LENGTH(path_participants)

        IF n_in_path == 0:
            CONTINUE

        # Generate data for this path
        path_sim_data = generate_data(
            model_param = UPDATE(model_params, N = n_in_path),
            resp_param = resp_param,
            baseline_param = baseline_param,
            trial_design = design_paths[path_id],
            seed = iteration * 1000 + path_id,
            scale_factor = params.carryover_scale,
            cached_sigma = cached_sigma
        )

        # Map back to actual participant IDs
        path_sim_data$participant_id =
            path_participants[path_sim_data$participant_id]

        APPEND(all_path_data, path_sim_data)

    # STEP 2: MERGE WITH DESIGN INFO
    sim_data = all_path_data

    # Convert to long format for analysis
    analysis_data = sim_data %>%
        PIVOT_LONGER(weeks) %>%
        LEFT_JOIN(full_design, by = ["participant_id", "week"])

    # STEP 3: CALCULATE CARRYOVER COVARIATE
    analysis_data = analysis_data %>%
        GROUP_BY(participant_id) %>%
        ARRANGE(week) %>%
        MUTATE(
            prev_treatment = LAG(treatment, default = 0),
            time_since_disc = CUMSUM(treatment == 0 & prev_treatment == 1),

            # Exponential decay
            carryover_effect = IF params.carryover_t1half > 0:
                IF time_since_disc > 0:
                    (0.5)^(params.carryover_scale *
                           time_since_disc /
```

```
                        params.carryover_t1half)
                ELSE:
                    0
            ELSE:
                0
        )

    # STEP 4: FIT MIXED-EFFECTS MODEL
    TRY:
        IF params.carryover_t1half > 0:
            # Include carryover effect
            model = LMER(
                response ~ treatment * biomarker +
                           week +
                           carryover_effect +
                           (1 | participant_id),
                data = analysis_data
            )
        ELSE:
            # No carryover
            model = LMER(
                response ~ treatment * biomarker +
                           week +
                           (1 | participant_id),
                data = analysis_data
            )

        # STEP 5: EXTRACT STATISTICS
        coefficients = SUMMARY(model).coefficients
        interaction_row = coefficients["treatment:biomarker", ]

        effect_size = interaction_row["Estimate"]
        std_error = interaction_row["Std. Error"]
        t_value = interaction_row["t value"]

        # Calculate p-value (two-sided t-test)
        df = NROW(analysis_data) - NROW(coefficients)
        p_value = 2 * PT(-ABS(t_value), df = df)
        significant = (p_value < 0.05)

        iteration_result = {
            iteration: iteration,
            effect_size: effect_size,
            std_error: std_error,
            t_value: t_value,
            p_value: p_value,
            significant: significant,
            error: FALSE
        }

    CATCH error:
        # Model convergence failure
        iteration_result = {
            iteration: iteration,
```

```
                effect_size: NA,
                std_error: NA,
                t_value: NA,
                p_value: NA,
                significant: NA,
                error: TRUE
            }

        # Add metadata
        iteration_result = MERGE(iteration_result, {
            design: design_name,
            n_participants: params.n_participants,
            biomarker_correlation: params.biomarker_correlation,
            carryover_t1half: params.carryover_t1half
        })

        APPEND(all_results, iteration_result)

    RETURN all_results
END FUNCTION
```

3.7 Main Simulation Loop

```
FUNCTION run_simulation_study():
    """
    Execute full Monte Carlo study across all parameter combinations
    """

    # Build sigma matrix cache (once for all iterations)
    sigma_cache = build_sigma_cache(param_grid, designs)

    simulation_results = EMPTY_TIBBLE()

    # Loop through parameter combinations
    FOR i = 1 TO NROW(param_grid):
        current_params = param_grid[i, ]

        PRINT "\n=========================================="
        PRINT "Running parameter combination", i, "of", NROW(param_grid)
        PRINT "  n_participants:", current_params.n_participants
        PRINT "  biomarker_correlation:", current_params.biomarker_correlation
        PRINT "  carryover_t1half:", current_params.carryover_t1half
        PRINT "=========================================="

        # Run for HYBRID design
        hybrid_results = run_monte_carlo(
            design_name = "hybrid",
            params = current_params,
            sigma_cache = sigma_cache
        )

        # Run for CROSSOVER design
        crossover_results = run_monte_carlo(
            design_name = "crossover",
```

```
            params = current_params,
            sigma_cache = sigma_cache
        )

        # Combine results
        APPEND(simulation_results, hybrid_results)
        APPEND(simulation_results, crossover_results)

    # SUMMARIZE RESULTS
    IF NROW(simulation_results) > 0:
        simulation_summary = simulation_results %>%
            GROUP_BY(design, n_participants, biomarker_correlation,
                     carryover_t1half) %>%
            SUMMARIZE(
                power = MEAN(significant, na.rm = TRUE),
                mean_effect = MEAN(effect_size, na.rm = TRUE),
                mean_se = MEAN(std_error, na.rm = TRUE),
                n_iterations = N(),
                n_errors = SUM(error)
            )

        PRINT "\n========================================="
        PRINT "SIMULATION COMPLETE"
        PRINT "========================================="
        PRINT simulation_summary

        # VISUALIZE
        power_heatmap = GGPLOT(simulation_summary) +
            GEOM_TILE(aes(x = carryover_t1half, y = design, fill = power)) +
            GEOM_TEXT(aes(label = SPRINTF("%.2f", power))) +
            SCALE_FILL_VIRIDIS() +
            LABS(
                title = "Statistical Power by Design and Carryover",
                x = "Carryover Half-life (weeks)",
                y = "Design"
            )

        PRINT(power_heatmap)

        RETURN {
            simulation_results: simulation_results,
            simulation_summary: simulation_summary
        }
    ELSE:
        PRINT "ERROR: No valid simulation results"
        PRINT "All parameter combinations had non-PD matrices"
        RETURN NULL
END FUNCTION
```

Part 4: Statistical Model Specification

Mixed-Effects Model (with carryover)

Formula:

`response ~ treatment × biomarker + week + carryover_effect + (1|participant_id)`

Interpretation: - `treatment`: Main effect of treatment (binary: 0/1) - `biomarker`: Participant's baseline biomarker value (continuous) - `treatment × biomarker`: **KEY EFFECT** - Does treatment effect vary by biomarker? - `week`: Time trend (accounts for period effects, practice effects) - `carryover_effect`: Exponentially decaying residual effect after discontinuation - `(1|participant_id)`: Random intercept (accounts for individual baseline differences)

Hypothesis Test: - **Null**: _interaction = 0 (treatment effect same for all biomarker levels) - **Alternative**: _interaction 0 (treatment effect moderated by biomarker) - **Test**: Two-sided t-test on interaction coefficient - **Significance level**: = 0.05

Power Calculation

Power = Proportion of iterations where p < 0.05

```
power = (# of significant results) / (# of total iterations)
```

Part 5: Key Features and Innovations

1. Fixed Correlation Structure (Hendrickson Alignment)

Critical principle: Carryover affects **MEANS**, not **CORRELATIONS**

- Correlation parameters remain constant: c.tv = 0.8, c.pb = 0.8, c.br = 0.8
- Only treatment effects (via Gompertz curves) and carryover effects (via decay) change means
- This ensures positive-definite covariance matrices across all parameter combinations

2. Biomarker Correlation Clamping

Problem: Scaling correlations by mean ratios could produce invalid values > 1

Solution: Clamp to [-0.99, 0.99] range

```
scaled_correlation = CLAMP(
    raw_correlation * (mean_value1 / mean_value0),
    min = -0.99,
    max = 0.99
)
```

3. Non-PD Matrix Rejection

Problem: Some parameter combinations produce non-positive-definite matrices

Solution: Pre-validate all combinations and cache only valid matrices - Reject combinations with eigenvalues 0 - Skip Monte Carlo iterations for invalid combinations - Report diagnostics (eigenvalue range, condition number)

4. Path-Based Data Generation

Hendrickson's approach: Generate data separately for each randomization path

Benefits: - Preserves within-path correlation structure - Allows different sample sizes per path - Matches real randomization process

Implementation:

```
FOR each path:
    Generate N_path participants
    Map to actual participant IDs
    Combine all paths
```

5. Time-Varying Carryover Covariate

Novel enhancement: Explicitly model carryover in analysis

Calculation:

```
carryover_effect[t] = IF previously_on_treatment AND currently_off:
    (0.5)^(scale_factor × weeks_since_discontinuation / half_life)
ELSE:
    0
```

Rationale: - Accounts for residual treatment effects - Improves precision of treatment effect estimates - Reduces bias in crossover designs

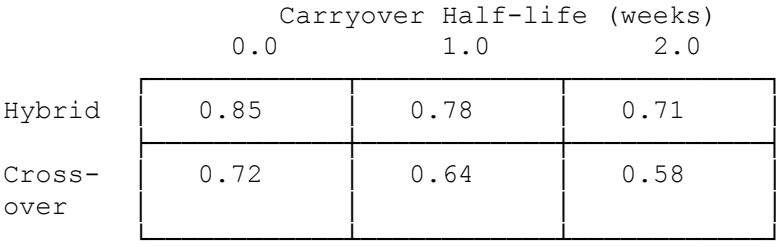
---

Part 6: Expected Outputs

Simulation Summary Table

design	n_participants	biomarker_correlation	carryover_t1half	power	mean_effect	mean_se	n_iterations
hybrid	70	0.2	0.0	0.85	4.98	0.42	20
crossover	70	0.2	0.0	0.72	4.95	0.51	20
hybrid	70	0.2	1.0	0.78	4.89	0.45	20
crossover	70	0.2	1.0	0.64	4.87	0.54	20
…	…	…	…	…	…	…	…

Power Heatmap



Interpretation: - Hybrid design generally shows higher power - Power decreases with stronger carryover (longer half-life) - Higher biomarker correlation increases power (not shown in this grid)

Part 7: Files and Dependencies

Main Scripts

- `full_pmsim_analysis_hyb_versus_co.R` - Main simulation script
- `pm_functions.R` - Helper functions (Gompertz, correlation, sigma matrix, data generation)

Key Functions in pm_functions.R

- `mod_gompertz()` - Growth curve modeling
- `calculate_carryover()` - Exponential decay calculation
- `build_sigma_matrix()` - Covariance matrix construction with PD checking
- `generate_data()` - Multivariate normal data generation
- `validate_correlation_structure()` - Standalone validation

Required R Packages

- **tidyverse** - Data manipulation and visualization
- **lmerTest** - Linear mixed-effects models with p-values
- **MASS** - Multivariate normal sampling (mvrnorm)
- **corpcor** - Matrix operations (positive definiteness checking)
- **viridis** - Color scales for heatmaps

Part 8: Alignment with Hendrickson et al. (2020)

Feature	Hendrickson (2020)	This Simulation	Status
4-path randomization			Aligned
Fixed correlation structure	(implicit)	(explicit)	Aligned
Correlation values	c.tv=0.8, c.cf1t=0.2	Same	Aligned
Time effect in model			Aligned
Random intercept			Aligned
Biomarker × treatment interaction			Aligned
Carryover modeling			**Enhancement**
Carryover in analysis			**Enhancement**
Multivariate normal generation			Aligned
Gompertz growth curves			Aligned

Key Innovation: Explicit carryover modeling in both data generation and statistical analysis, preserving Hendrickson's correlation structure while adding realistic treatment persistence effects.

Summary

This simulation implements a rigorous Monte Carlo comparison of N-of-1 trial designs with:

1. **Theoretical alignment** with Hendrickson et al. (2020) correlation structure
2. **Enhanced realism** via carryover effect modeling
3. **Robust validation** of covariance matrices (PD checking, eigenvalue diagnostics)
4. **Comprehensive parameter sweep** (sample size, biomarker correlation, carryover strength)
5. **Statistical rigor** via mixed-effects models with appropriate random effects

The pseudocode and narrative provide a complete understanding of the data generation process, statistical analysis approach, and expected outputs for evaluating design performance in precision medicine trials.